Rational Verification in Multi-Agent Systems

Muhammad Najib

Linacre College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

September 2019

Declaration

I declare that the work presented in this thesis is my own except where explicitly stated otherwise in the text, and that no portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Acknowledgements

First, I thank my supervisors, Julian Gutierrez and Michael Wooldridge, for all of the suggestions, help, encouragement, and, most importantly, tremendous support; I could not ask for better supervisors.

Second, thanks to LPDP (Indonesia Endowment Fund for Education) for financial support.

Third, I am immensely grateful to Giuseppe Perelli (for being such an awesome collaborator), Franco Raimondi (for the discussions on part of this thesis); also to Marta Kwiatkowska, Alessio Lomuscio, and Luke Ong, for agreeing to be the examiners of my reports and thesis.

Fourth, special thanks also go to those who have made the past four years in Oxford to be such a wonderful experience, especially to (in no particular order): Paul Harrenstein (for constant supply of coffee), Jiarui Gan (for afternoon trips to Covered & Gloucester Green Market), Norbert Nthala (for the walks through University Park and lunches at Linacre), and all brilliant people I met in RACE project, room 017, and the Indonesian community in Oxford.

Finally, I thank my family for always supporting my decisions.

Abstract

Rational verification problem is concerned with checking which temporal logic properties will hold in a system composed of multiple agents which are assumed to behave rationally and strategically in pursuit of individual objectives. Unfortunately, the problem is generally hard from computational point of view, and for the purpose of practical implementations, usually requires specialised techniques.

This thesis aims to develop algorithms and study computational complexity results for rational verification in multi-agent systems. Firstly, a practically amenable technique which relies on a reduction to the solution of a collection of parity games is proposed. The technique in this thesis uses a model of strategies that is bisimulation invariant—that is, in which individual strategies for system components are valid across all bisimilar systems, and which satisfy the same temporal logic properties in equilibrium. This approach has been implemented in the Equilibrium Verification Environment (EVE) system. Secondly, some cases in which the problem of rational verification is computationally tractable are investigated. In particular, it is shown that the complexity of rational verification can be reduced from 2EXPTIME-complete to fixed-parameter tractable. Furthermore, improved complexity results when considering quantitative goals, namely mean-payoff utility functions, are also studied. In doing so, a concept called *equilibrium design* is proposed. This concept is concerned with the design of incentives so that a desirable equilibrium is obtained.

Contents

List of Figures

List of Notations

2^S	where S is a set, denotes the powerset of S
$S \times S'$	cross product of the set S and S'
$\models$	semantic satisfaction relation
$Sub(\varphi)$	subformula set of formula φ
$Sat(\varphi)$	the set of states satisfying φ
S_{-A}	the set $S \setminus A$
$\pi, \pi(\vec{\sigma})$	a path, the unique path induced by $\vec{\sigma}$
$\mathrm{Inf}(\pi)$	the set of states occuring infinitely often in the path π
W, L	the set of winners and losers in a game
σ_i	strategy for player i
$\vec{\sigma}$	strategy profile $(\sigma_1, \ldots, \sigma_n)$ for the set of players N
$\succeq_i$	preference relation over outcomes for player i
$(\vec{\sigma}_{-i}, \sigma_i')$	the strategy profile where the strategy of player i in $\vec{\sigma}$ is replaced by σ_i'
$\mathrm{NE}(\mathcal{G})$	the set of Nash equilibria in game $\mathcal{G}$
$\mathrm{Pun}_i(\mathcal{G})$	the set of states in $\mathcal{G}$ from which player i can be punished
$\mathtt{pun}_i(s)$	the punishment value for player i in s
$\mathrm{Win}_i(\mathcal{G})$	winning region of player i ini $\mathcal{G}$
η	an action profile run
$\mathsf{mp}(r)$	mean-payoff value of $r \in \mathbb{R}^\omega$
λ	labelling function
α	priority function for parity condition
$\mathcal{A}, \mathcal{S}$	deterministic parity & Streett automaton
w	weight function
$\kappa, \mathcal{K}, \beta$	subsidy scheme, set of subsidy schemes, budget

Chapter 1

Introduction

Many software systems and applications can naturally be understood as collections of interacting (semi-)autonomous agents. This has given rise to the paradigm of multi-agent systems [Fagin et al., 1995, Weiß, 1999, Wooldridge, 2002, Shoham and Leyton-Brown, 2008]. The field of multi-agent systems is largely concerned with the theory and practice of systems composed of multiple interacting software components (agents), which are assumed to be acting autonomously in pursuit of delegated goals or preferences. The aim of this thesis is to develop practical algorithms and study computational complexity results for rational verification in multi-agent systems.

This chapter discusses the motivations for developing techniques that can solve the verification problem for multi-agent systems. The relation between multi-agent systems and concurrent games is then presented. Finally, the main contributions and the structure of the remainder of the thesis are outlined.

1.1 Motivation

This section describes a high-level motivation for the research presented in this thesis. More specifically, this section presents two relevant issues underlying the motivation of this thesis: *computer bugs* and *artificial intelligence safety*.

Computer Bugs. Computer systems play an important role in our lives and it is not difficult to see that this role will become more significant in the future. It is inevitable that these systems are getting more complex and integrated to our daily life via mobile phones, portable computers, internet, and various kinds of embedded systems. But as our dependency grows, we are more exposed to the risk of malfunctioning systems. One of the "canonical" examples of this risk is the launch of the Ariane-5 rocket on 4th June 1996 [ari, 1996]. The rocket exploded just 36 seconds

after take-off due to a software error in converting different data-types. This "simple" bug costed the European Space Agency more than $500 million. Indeed, there are other well-known examples [lis, 2016] of software bugs that not only costed money, but sadly, also lives.

Clearly, in the setting where failure is unacceptable, the need for reliable systems is crucial. During the last two decades, research in formal verification has led to the emergence of techniques for detecting defects early in the development stage of systems. Model checking is one of the most successful approaches to formal verification, in which a system is represented as a formal model such that its behaviour is described in a mathematically precise and unambiguous manner. The designer then can specify a property using a temporal logic formula and automatically check with a model checker whether it is reflected in the system behaviour. The usage of temporal logic has a rather nice advantage, since one can specify a property such as "the system will never give a wrong response" or "every request will eventually be served"[1] in an intuitive and natural way.

Artificial Intelligence and Safety. The recent revival and rapid progress of artificial intelligence (AI) has sparked some concern regarding its (possibly dangerous) consequence to humanity. A complex system composed of many (intelligent) components may appear not only as a reactive concurrent system, but to some extent, a "black box" to humans of which behaviour is difficult to predict. A relatively recent example of this problem happened on 6th May 2010 of what is known as the "2010 Flash Crash" [Kirilenko et al., 2017] where stock indexes in the United States collapsed and rebounded in a span of 36 minutes. High-frequency traders use highly sophisticated algorithms for stock transactions at high volumes and speeds, and this contributed to market volatility in the Flash Crash. This event suggested that we need new tools for analysing and verifying a system composed of multiple interacting autonomous, intelligent, and self-interested agents, to avoid potential undesireable behaviour. It is even more relevant when the system in consideration is used in safety-critical situations. The Future of Life Institute (FLI) listed the problems of verification and validation as two of the research priorities in AI [Russell et al., 2016].

Beyond Classical Model Checking. The most successful approach to prove the correctness of some system with respect to temporal logic specifications is model

[1] In reactive concurrent systems, these properties are commonly known, respectively, as *safety* and *liveness*.

checking [Clarke et al., 2002]. The basic idea of model checking is to represent the behaviour of a finite state program using a Kripke structure or transition system. However, in the context of multi-agent systems, the approach of determining correctness using classical model checking is not appropriate [Wooldridge et al., 2016]. The classical model checking approach is missing an important ingredient of such systems: agents may be assumed to pursue their preferences rationally and strategically. Thus, some runs of the system that in principle might be possible, in practice, will never arise from rational choices by agents within the system. Thus, the question that would be more appropriate to check for the correctness of a multi-agent system against a proprety φ is as follows: "would φ be satisfied in some run that would be sustained by a Nash equilibrium collection of choices by agents in the system?" This problem is called equilibrium checking, and the general paradigm is known as rational verification [Wooldridge et al., 2016].

1.2 Multi-agent Systems and Games

Concurrent and multi-agent systems can be naturally understood and modelled as multi-player games [Shoham and Leyton-Brown, 2008, Gutierrez et al., 2017b]. In this framework, concurrent/multi-agent systems correspond to games, system components (agents) correspond to players, computation runs of the system correspond to plays of the game, and individual component behaviours correspond to player strategies, which define how players make choices in the system over time. Game theory provides a number of solution concepts through which to analyse such systems, of which Nash equilibrium [Osborne and Rubinstein, 1994] stands out as the most fundamental and important in noncooperative settings. A profile of strategies, one for each player in a game, is said to be a Nash equilibrium if no player could benefit by unilaterally changing its strategy assuming the other players' strategies remain unchanged. Previous work on the game theoretic analysis of concurrent/multi-agent systems has taken Nash equilibrium, and refinements of it, as the central solution concept. Our main interest is the development of the theory and tools for the automated game theoretic analysis of concurrent/multi-agent systems, and in particular, the analysis of temporal logic properties that will hold in a system under the assumption that players choose strategies which form a Nash equilibrium[2].

[2]In this work we focus on Nash equilibrium; however, a similar methodology may be applied using refinements of Nash equilibrium and other solution concepts.

At this point, it is useful to state the assumptions that are used for the rest of this thesis. Some of the assumptions are very natural and, indeed, we put forward some argument why it is the case. Otherwise, they are placed out of necessity—either there is some fundamental restriction, or simply because it is aimed as an early approximation for further research.

Firstly, we assume that the players are playing pure strategies, and as such, the solution concept used is *pure* Nash equilibrium. This is because in mixed strategies, the existence of (mixed) Nash equilibrium is guaranteed [Nash, 1951], which makes some problem considered in this thesis become trivial.

Secondly, we assume that the games are in complete information and perfect recall setting. Clearly, this is ultimately too strong an assumption in general setting, but it will suffice as a first approximation for further research[3]. Nevertheless, there are some classes of multi-agent systems in which complete information is relevant, such as in some non-antagonistic (non-zero-sum) environment wherein a centralised coordination is not feasible. In such a system, agents are not gaining (significant) advantage by keeping their strategies and goals private. To see this, consider a system populated with autonomous car agents. If we model the interaction between the agents (autonomous cars), it is not appropriate to model it as a zero-sum game, since, despite the fact that the goal (*e.g.*, destination) of each agent is most likely unique, an agent will not be better off by preventing other agents to reach their destinations. Indeed, in some papers such as [Fisac et al., 2019, Stefansson et al., 2019], the authors consider some autonomous car systems in which the setup is of (some type of) complete information.

Thirdly, we implicitly assume that the strategies have finite memory. Although, it is important to point out that we impose no bound on memory size. We argue that this is a natural assumption, since in practice, any implementation of some agent will always have this restriction (*e.g.*, RAM size).

1.3 Structure of the Thesis

This thesis presents the development of algorithmic techniques and the study of computational complexity results for rational verification in multi-agent systems. The thesis is structured as follows:

[3]Weakening this assumption may make some decision problems become undecidable in general [Gutierrez et al., 2018b, Dima and Tiplea, 2011, Berthon et al., 2017].

- Chapter 2 summarises some backgorund material on modal logics, temporal logics, game theoretical representations of multi-agent systems, formal verification via model checking, and recent developments of model checking tools for multi-agent systems. The material in this chapter enables the introduction of some technical background for the next chapter which presents an approach to rational verification.

- Chapter 3 presents an approach to solve rational verification problems using MCMAS. The results in this chapter are later used in performance evaluation for the proposed approach in this thesis.

- Chapter 4 presents the main approach to solve rational verification problems. The algorithmic techniques presented in this chapter are the underlying foundations for the implementation/tool: "Equilibrium Verification Environment" (EVE).

- Chapter 5 studies some cases in which rational verification problems are (relatively) computationally tractable. This chapter also introduces games with quantitative goals (mean-payoffs), extending the previously qualitative objectives (LTL formulae).

- Chapter 6 introduces the concept of equilibrium design for multi-agent systems. It also studies the computational complexity of the proposed problems.

- Chapter 7 presents the implementation of the equilibrium checker EVE, analysis of various applications of EVE to some protocols and examples from the literatures, and reports its performance in comparison with two other tools: MCMAS and PRALINE.

- Chapter 8 discusses the contribution of the thesis, evaluation results of the proposed approach, and outlines possible extensions of this work.

Chapter 2

Background

This chapter presents some fundamental concepts on which verification tools are built, especially the frameworks that enable the development of different techniques used in the tools. More specifically, this chapter discusses the role of different logics in providing conceptual structures for modelling and reasoning about systems.

Formal logic has been used widely in philosophy and computer science as a framework for reasoning about systems. It provides a way to express and analyse the properties of systems in a formal and rigorous fashion. Sometimes, it even allows us to build a powerful machinery to carry out the analysis automatically, which is the foundation of all verification tools. Readers' familiarity with (classical) propositional logic is assumed, thus this chapter starts from *modal logic* and proceeds to the more recent formalisms. Later in this chapter, the argument of how game theory fits as an appropriate tool for reasoning about systems with multiple agents is also presented.

2.1 Modal Logics for Multi-agent Systems

C. I. Lewis founded modern *modal logic* through his works in the beginning of 19th century [Lewis, 1918] and his book [Lewis and Langford, 1932] (co-authored with C. H. Langford). Modal logic extends classical propositional logic with two operators expressing modality: $\Box$ (*necessity*) and $\Diamond$ (*possibility*). Formula $\Box\varphi$ says that φ is *necessarily true* and similarly, $\Diamond\varphi$ says that φ is *possibly true*. The current standard interpretation of modal logic is using *Kripke semantics*.

To describe *Kripke semantics*, we need the definition of Kripke models (due to Saul Kripke) [Kripke, 1963]. A Kripke model is a kind of transition system modelled as a graph whose nodes represent the states of the system and edges represent the reachability relations. A Kripke model also has a labelling function that maps each

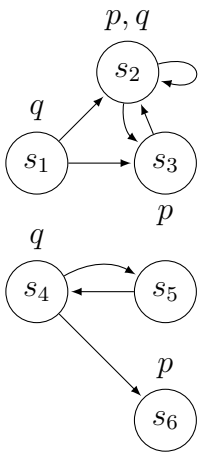

Figure 2.1: Graphical representation of Example 1.

node to a set of properties (usually, atomic propositions) that hold in that node. Formally, a Kripke model is defined as follows.

Definition 1 (Kripke model)**.** Let AP be a set of atomic propositions. Kripke models are the models of modal logics which include the set of (possibly empty) worlds (or states) St, a modal accessibility relation $\mathcal{R} \subseteq \mathrm{St} \times \mathrm{St}$, and a valuation of the atomic propositions $\mathcal{V} : \mathrm{St} \to 2^{\mathrm{AP}}$.

Example 1. Let $M = (\mathrm{St}, \mathcal{R}, \mathcal{V})$, where $\mathrm{St} = \{s_1, \ldots, s_6\}$,
$\mathcal{R} = \{(s_1, s_2), (s_1, s_3), (s_2, s_2), (s_3, s_2), (s_4, s_5), (s_5, s_4), (s_5, s_6)\}$,
and $\mathcal{V} = \{(s_1, \{q\}), (s_2, \{p, q\}), (s_3, \{p\}), (s_4, \{q\}), (s_5, \{\}), (s_6, \{p\})\}$. The graphical representation is shown in Figure 2.1.

Now we can define the Kripke semantics of standard modal logic. Formally, it is defined as follows [Kripke, 1963]. Let $M = (\mathrm{St}, \mathcal{R}, \mathcal{V})$ be a Kripke model, $p \in \mathrm{AP}$, and $s \in \mathrm{St}$. The truth value of formula φ at s in M is given by the semantic relation $\models$, and defined by:

$$
\begin{array}{lll}
M, s \models p & \text{iff} & p \in \mathcal{V}(s), \\
M, s \models \neg\varphi & \text{iff} & M, s \not\models \varphi \\
M, s \models \varphi \vee \psi & \text{iff} & M, s \models \varphi \text{ or } M, s \models \psi \\
M, s \models \Box\varphi & \text{iff} & \text{for all } s' \in \mathrm{St}, \text{ if } s\mathcal{R}s' \text{ then } M, s' \models \varphi
\end{array}
$$

Observe that modal possibility and conjunction can be defined as combinations of other operators: $\Diamond\varphi \equiv \neg\Box\neg\varphi$ and $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$. We can also extend modal logic with multiple modal operators: $\Box_i$ and $\Diamond_i$, each of them interpreted over a corresponding modal accessibility relation $\mathcal{R}_i \subseteq \mathrm{St} \times \mathrm{St}$. For further exposition to modal logic, we refer to the work by Blackburn *et.al.* [Blackburn et al., 2001].

2.2 Temporal Specification and Verification of Systems

This section presents a brief overview of logics that capture the dynamics of systems, namely *linear temporal logic* and *branching time/computation tree logic*, as well as the basic notion of model checking and the underlying algorithms.

Temporal logic was first proposed by Arthur Prior [Prior, 1957, Prior, 1967, Prior, 1968] in an attempt to facilitate reasoning about statements such as "Liverpool Football Club will win the Premier League this season". This statement might be true if evaluated in the first month of the Premier League season, but false if evaluated just a month before the end of the season. To analyse such a sentence, Prior proposed the idea of extending modal logic by adding four *temporal* modal operators:

$\mathbf{P}\varphi$ "It *has previously been* the case that φ is true"
$\mathbf{F}\varphi$ "It *will eventually be* the case that φ is true"
$\mathbf{H}\varphi$ "It *has always been* the case that φ is true"
$\mathbf{G}\varphi$ "It *will always be* the case that φ is true"

This later was extended by Hans Kamp in his doctoral dissertation [Kamp, 1968] by introducing two *binary* temporal operators:

$\varphi\mathbf{S}\psi$ "φ has been true since a time when ψ was true"
$\varphi\mathbf{U}\psi$ "φ will be true until a time when ψ is true"

This proposed language clearly cannot be interpreted in the static, non-changing models of classical logic. One simple approach in order to give the semantics of this language is by interpreting time as a linear sequence of time-steps:

$$t_0, t_1, t_2, \ldots$$

A straightforward view is that the flow of time is regarded as the natural numbers $\mathbb{N}$ in which each time-step corresponds to an element in $\mathbb{N}$ ordered by the "less than" relation, "$<$". Of course there are other posible interpretations of the flow of time

which will yield different properties, such as dense model[1] and interval model[2]. In the following section, we will discuss an extension of modal logic with respect to linear sequences of time-steps called *linear temporal logic* (LTL).

2.2.1 Linear Temporal Logic

Prior's proposal has led to the development of a formalism used in computer science [Pnueli, 1977] namely *linear temporal logic*, in which the basic operators are: **X** ("next"), **F** ("eventually"), and **G** ("always"), and **U** ("until"). It was developed as a tool for formal verification of computer programs and based on a linear-time perspective which allows the specification of total relative orders of events. The intuitive description of these standard temporal operators is as follows:

$\mathbf{X}\varphi$	"φ will be true in the next moment"
$\mathbf{F}\varphi$	"φ is true now or will eventually be true in the future"
$\mathbf{G}\varphi$	" φ is true now and will always be true in the future"
$\varphi\,\mathbf{U}\,\psi$	"ψ will eventually be true in the future and φ will always be true until the moment before ψ becomes true."

These operators are useful for expressing properties of infinite computations in reactive systems, namely *safety, liveness*, and *fairness* [Manna and Pnueli, 1992].

A *safety property* asserts that nothing bad will happen. It corresponds to maintenance of goals throughout the lifespan of the system. For example the statement $\mathbf{G}\neg crash$ says that the system will never crash.

A *liveness property* asserts that something good eventually happens. It corresponds to achievements of goals, that is, the goals should be achieved at some point in the future. For instance the statement $\mathbf{F}terminate$ says that the system will eventually terminate in the future.

A *fairness property* corresponds to services that should be provided sufficiently often. For instance, the statement $\mathbf{G}(request \rightarrow \mathbf{F}delivered)$ is translated as "it is always the case that if one requests something then eventually it will be delivered".

[1]The dense model put a one-to-one correspondence between time points and real numbers. This interpretation arguably gives a powerful model to reason about the "real world" time, as well as comes with an expensive computational cost. In fact, a variant of logic built on top of this interpretation is highly undecidable [Alur and Henzinger, 1994].

[2]The interval model does not view time as points, but as intervals. This modelling approach gives rise to more expressive kinds of logic. However, the intractability of the logics based on this interpretation [Halpern et al., 1983, Halpern and Shoham, 1991, Venema, 1991, Chaochen et al., 1993, Chaochen et al., 1991] contributes to their unpopularity in practical usage.

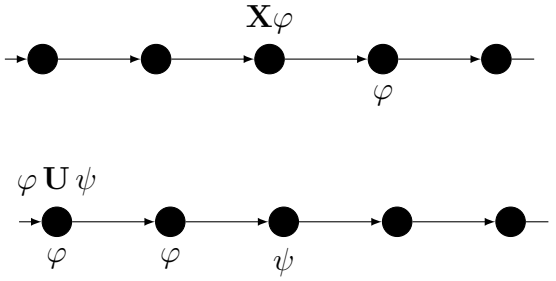

Figure 2.2: Examples for $\mathbf{X}\varphi$ and $\varphi \,\mathbf{U}\, \psi$.

Formally, the syntax of LTL, where $p \in AP$, is formed according to the following grammar:

$$\varphi ::= \text{true} \mid p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi \,\mathbf{U}\, \varphi.$$

Other Boolean operators such as disjunction, implication, etc., are defined as usual and can be represented by the operators $\wedge$ and $\neg$. The until operator also allows us to derive the temporal operators **F** and **G** as follows:

$$\mathbf{F}\varphi \equiv \text{true}\,\mathbf{U}\,\varphi \qquad \mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$$

Definition 2 (Kripke transition model)**.** A Kripke transition model is a tuple $M = (\text{St}, \rightarrow, \mathcal{V})$ where St is a non-empty set of states, $\rightarrow \subseteq \text{St} \times \text{St}$ is a transition relation, and $\mathcal{V} : \text{St} \rightarrow 2^{\text{AP}}$ is a valuation of atomic propositions.

Definition 3 (Path)**.** A path π of M is an infinite sequence $\pi = s_0, s_1, s_2, \ldots$ of sequentially accessible states that can be generated by the transition relation of a system M.

Semantically, LTL is defined by using infinite paths drawn from executions of a Kripke transition model M. Let $Paths(M)$ be the set of all infinite paths in M and $\pi \in Paths(M)$. Write $\pi[j]$ to denote the jth state of π, $\pi[...j]$ to denote the jth prefix of π, $\pi[j...]$ to denote the suffix of π after jth state, and $\pi[j...k]$ to denote the fragment between j and k on π. Let $\pi \in Paths(M)$, the semantics of LTL over paths is given as:

$$\begin{array}{lll}
M, \pi \models p & \text{iff} & M, \pi[0] \in \mathcal{V}(p) \\
M, \pi \models \neg\varphi & \text{iff} & M, \pi \not\models \varphi \\
M, \pi \models \varphi_1 \wedge \varphi_2 & \text{iff} & M, \pi \models \varphi_1 \text{ and } M, \pi \models \varphi_2 \\
M, \pi \models \mathbf{X}\varphi & \text{iff} & M, \pi[1...] \models \varphi \\
M, \pi \models \varphi_1 \,\mathbf{U}\, \varphi_2 & \text{iff} & M, \pi[j...] \models \varphi_2 \text{ for some } j \geq 0, \text{ and } M, \pi[i] \models \varphi_1 \text{ for all } 0 \leq i < j
\end{array}$$

The next section presents another formalism that extends modal logic with respect to time. In this formalism, time is modelled as a tree-like structure which resembles non-determinism, i.e., there may be more than one path in the future, any one of which might be realised.

2.2.2 Computation Tree Logic

The previous section presents LTL, a kind of temporal logic that allows us to reason about single infinite paths of a system. This language is clearly not an appropriate tool to reason about reactive systems where there may be several different possible futures. Thus, each moment of time may split into several possible futures. To capture this kind of behaviour, *branching-time temporal logic* was proposed [Clarke and Emerson, 1981]. CTL* extends LTL with path quantifiers E ("some paths") and A ("all paths") which express a quantification on alternative possible infinite paths. Intuitively, given a property φ, the formula $\mathsf{E}\varphi$ means that there exists an infinite path where φ is true, while $\mathsf{A}\varphi$ states that φ is true on all possible infinite paths. Formulae in CTL* are classified into state and path formulae. State formulae are interpreted in the states of a model, while path formulae are interpreted on the paths of a model. The syntax of state formulae is given by the following grammar:

$$\varphi ::= \text{true} \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{E}\psi \mid \mathsf{A}\psi$$

where $p \in AP$ and ψ is a path formula. The path formulae are formed according to the following grammar:

$$\psi ::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \psi \,\mathbf{U}\, \psi$$

Let $M = (\mathrm{St}, \rightarrow, \mathcal{V})$ be a Kripke transition model, $s \in \mathrm{St}$, and $p \in \mathrm{AP}$. The semantics of CTL* state formulae are defined as follows:

$M, s \models p$ iff $s \in \mathcal{V}(p)$

$M, s \models \neg\varphi$ iff $M, s \not\models \varphi$

$M, s \models \varphi_1 \wedge \varphi_2$ iff $M, s \models \varphi_1$ and $M, s \models \varphi_2$

$M, s \models \mathsf{E}\psi$ iff $\exists \pi \in Paths(M)$, starting from s, for which we have $M, \pi \models \psi$

$M, s \models \mathsf{A}\psi$ iff $\forall \pi \in Paths(M)$, starting from s, we have $M, \pi \models \psi$

The semantics of CTL* path formulae is given by:

$$
\begin{array}{lll}
M, \pi \models \varphi & \text{iff} & M, \pi[0] \models \varphi \\
M, \pi \models \neg\psi & \text{iff} & \pi \not\models \psi \\
M, \pi \models \psi_1 \wedge \psi_2 & \text{iff} & \pi \models \psi_1 \text{ and } \pi \models \psi_2 \\
M, \pi \models \mathbf{X}\psi & \text{iff} & \pi[1...] \models \psi \\
M, \pi \models \psi_1 \,\mathbf{U}\, \psi_2 & \text{iff} & \pi[j...] \models \psi_2 \text{ for some } j \geq 0, \text{ and } \pi[i] \models \psi_1 \text{ for all } 0 \leq i < j
\end{array}
$$

CTL is a subset of CTL* where every path quantifier must be immediately followed by a temporal operator. It was originally proposed by Clarke and Emerson [Clarke and Emerson, 1981] and used in slightly different form by Queille and Sifakis [Queille and Sifakis, 1982]. CTL precedes CTL*; in fact CTL* was defined several years later in [Emerson and Halpern, 1986]. The syntax of CTL is slightly different to the one used in CTL* due to the restriction of coupling a path quantifier with a temporal operator. Let $p \in \mathrm{AP}$, the set of CTL state formulae in *existential normal form* (ENF) is given by:

$$\varphi ::= \text{true} \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{E}\mathbf{X}\varphi \mid \mathsf{E}\mathbf{G}\varphi \mid \mathsf{E}(\varphi \,\mathbf{U}\, \varphi).$$

Theorem 1. For each CTL formula φ, there exists an equivalent CTL formula $\widehat{\varphi}$ in ENF.

Proof. The proof follows from the duality laws. □

The semantics of CTL is given by:

$$
\begin{array}{lll}
M, s \models p & \text{iff} & s \in \mathcal{V}(p) \\
M, s \models \neg\varphi & \text{iff} & M, s \not\models \varphi \\
M, s \models \varphi_1 \wedge \varphi_2 & \text{iff} & M, s \models \varphi_1 \text{ and } M, s \models \varphi_2 \\
M, s \models \mathsf{E}\mathbf{X}\varphi & \text{iff} & \exists \pi \in \mathit{Paths}(M), \text{ starting from } s, \text{ s.t. } M, \pi[1] \models \varphi \\
M, s \models \mathsf{E}\mathbf{G}\varphi & \text{iff} & \exists \pi \in \mathit{Paths}(M), \text{ starting from } s, \text{ s.t } M, \pi[j] \models \varphi \text{ for all } j \geq 0 \\
M, s \models \mathsf{E}(\varphi_1 \,\mathbf{U}\, \varphi_2) & \text{iff} & \exists \pi \in \mathit{Paths}(M), \text{ starting from } s, \text{ s.t } M, \pi[j] \models \varphi_2 \text{ for some } j \geq 0 \\
 & & \quad \text{and } M, \pi[k] \models \varphi_1 \text{ for all } 0 \leq k < j
\end{array}
$$

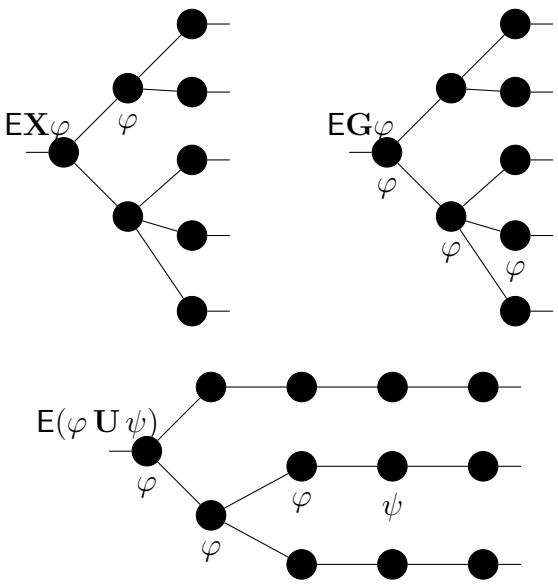

Figure 2.3: Examples for $\mathsf{EX}\varphi$, $\mathsf{EG}\varphi$ and $\mathsf{E}(\varphi\,\mathbf{U}\,\psi)$.

This section presents CTL* and some of its fragments[3]. Observe that the semantics of CTL, one of the branching-time temporal logic's fragments, may be presented entirely with respect to the states. Later, when model checking algorithms are presented, it will become apparent that this makes CTL model checking substantially easier than CTL* model checking. One might ask about the expresiveness of LTL and CTL, both of which are fragments of CTL*. In what follows, it is shown that they are not comparable.

2.2.3 LTL vs CTL

Both LTL and CTL are subsets of CTL*, however, the logical expresiveness of LTL and CTL cannot be compared [Lamport, 1980]. For instance, the ability of CTL to explicitly put existential quantification over paths is more expressive when we want to specify about the possibility of the existence of a specific path in a transition model M, where M is best unfolded as a computation tree. Say we have a statement (in CTL) "$\mathsf{AGEF}p$". Figure 2.4 satisfies the statement, however there is no LTL formula exists that satisfies it because the horizontal sequence of states containing $\neg p$ will falsify the (LTL) formula.

[3]For a deeper exposition of temporal logic, one can refer to [Emerson, 1990, Ohrstrom and Hasle, 1995].

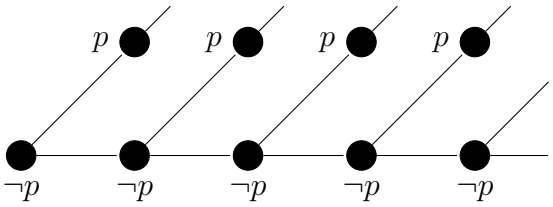

Figure 2.4: Example model for $\mathsf{AGEF}p$

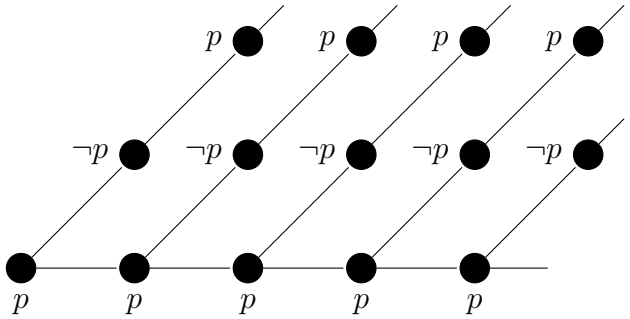

Figure 2.5: Example model for $\mathbf{FG}p$

The other way round also applies, *i.e.*, there are conditions that can be expressed in LTL but not in CTL. For instance, consider the LTL formula $\mathbf{FG}p$ and Figure 2.5. $\mathbf{FG}p$ holds in the model in Figure 2.5. However, any attempt to specify the same condition in CTL fails. For instance, consider the formula $\mathsf{AFAG}p$. This formula clearly has a different meaning to the LTL formula $\mathbf{FG}p$. Looking at this particular example, it seems that LTL allows us a greater ability to describe individual paths.

2.2.4 Temporal Logic Model Checking

Model checking [Clarke et al., 2002] is arguably the best-known and most successful approach for verifying the correctness of a system with respect to some temporal logic formulae. Model checking starts with the fact that it is possible to capture the behaviour of a finite state system P using a Kripke model or (labelled) transition system M. This means that transition system can be interpreted as models for temporal logic. Thus, given an LTL formula φ, checking whether P satisfies φ boils down to checking whether φ is satisfied on all infinite paths in M. In essence, model checking is a decision problem that takes as the inputs, model M representing a finite state abstraction of a system and a temporal logic property φ, and output true if and only if property φ is satisfied on M—that is, M is a model of φ.

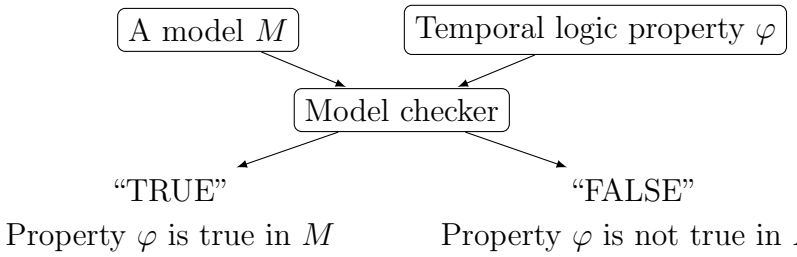

Figure 2.6: Basic structure of model checking; most of model checkers will provide a counter example for a false instance.

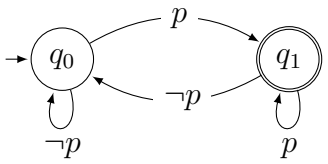

Figure 2.7: NBA for $\mathbf{GF}p$; q_1 is the accepting state.

The classical method for LTL model checking is based on automata theory[4] and was originally proposed by Vardi and Wolper [Vardi and Wolper, 1986]. It exploits the fact that each LTL formula φ can be translated into a *non-deterministic Büchi automaton* (NBA) [Büchi, 1962]. For instance, the LTL formula $\mathbf{GF}p$ can be translated into the NBA shown in Figure 2.7. This construction then can be used to find a witness or counterexample that falsifies φ in transition model M by constructing an NBA $\mathcal{A}$ from $\neg\varphi$ and *combine* it with M to form M'. We then can disprove $M \models \varphi$ by finding a path $\pi \in Paths(M')$ such that $M', \pi \models \neg\varphi$. If such a path is found, a prefix of π is returned as counterexample. Otherwise, it is concluded that $M \models \varphi$.

In the following LTL model checking algorithm, instead of a Kripke transition model, (labelled) *Transition Systems* ($\mathcal{TS}$) are used to model the behaviour of a system. One can, however, translate from one domain to another (*e.g.*, $\mathcal{TS}$ to Kripke transition model) as proposed by De Nicola and Vaandrager in their seminal work [De Nicola and Vaandrager, 1990], extended by Reiners and Willemse in [Reniers and Willemse, 2010].

Definition 4 (Transition System ($\mathcal{TS}$))**.** A *transition system* is a tuple $\mathcal{TS} = (\mathrm{St}, \mathrm{Ac}, \rightarrow, \mathcal{I}, \mathrm{AP}, L)$ where St is a set of states, Ac is a set of actions, $\rightarrow \subseteq \mathrm{St} \times \mathrm{Ac} \times \mathrm{St}$ is a

[4]There exist newer approaches based on alternating automata, *e.g.*, [Vardi, 1995], but popular model checkers such as SPIN [Holzmann, 1997] are based on standard non-deterministic automata.

transition relation, $\mathcal{I} \subseteq \mathrm{St}$ is a set of initial states, AP is a set of atomic propositions, and $L : \mathrm{St} \to 2^{\mathrm{AP}}$ is a labeling function.

Definition 5 (Non-deterministic Büchi automaton)**.** A non-deterministic Büchi automaton (NBA) $\mathcal{A}$ is a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to S^Q$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accepting states.

Definition 6 (Non-blocking NBA)**.** Let $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ be an NBA. $\mathcal{A}$ is called *non-blocking* if $\delta(q, A) \neq \varnothing$ for all states q and all symbols $A \in \Sigma$.

Definition 7 (Product of $\mathcal{TS}$ and NBA[5])**.** Let $\mathcal{TS} = (\mathrm{St}, \mathrm{Ac}, \to, \mathcal{I}, \mathrm{AP}, L)$ be a non-terminating transition system, and $\mathcal{A} = (Q, 2^{\mathrm{AP}}, \delta, Q_0, F)$ a non-blocking NBA. Let $\mathcal{TS} \otimes \mathcal{A} = (\mathrm{St} \times Q, \mathrm{Ac}, \to', \mathcal{I}', \mathrm{AP}', L')$ where $\to'$ is the minimum relation given by:

$$\frac{s \xrightarrow{\alpha} t \wedge q \xrightarrow{L(t)} p}{\langle s, q\rangle \xrightarrow{\alpha}{}' \langle t, p\rangle}$$

and where $\mathcal{I}' = \{\langle s_0, q\rangle | s_0 \in \mathcal{I} \wedge \exists q_0 \in Q_0.\ q_0 \xrightarrow{L(s_0)} q\}$, $\mathrm{AP}' = Q$, and $L' : \mathrm{St} \times Q \to 2^Q$ is defined by $L'(\langle s, q, \rangle) = \{q\}$.

Definition 8 (ω-Regular Expression)**.** An ω-regular expression G over the alphabet Σ has the form

$$\mathsf{G} = \mathsf{E}_1.\mathsf{F}_1^\omega + \cdots + \mathsf{E}_n.\mathsf{F}_n^\omega$$

where $n \geq 1$ and $\mathsf{E}_1, \ldots, \mathsf{E}_n, \mathsf{F}_1, \ldots, \mathsf{F}_n$ are regular expressions over Σ such that $\varepsilon \notin \mathcal{L}(\mathsf{F}_i)$, for all $1 \leq i \leq n$.

Definition 9 (ω-Regular Language)**.** A language $\mathcal{L} \subseteq \Sigma^\omega$ is called ω-regular if $\mathcal{L} = \mathcal{L}_\omega(\mathsf{G})$, where G is some ω-regular expression over Σ.

Let $\mathcal{L}_\omega(\mathcal{A})$ be the ω-regular language accepted by NBA $\mathcal{A}$, and let $Words(\varphi)$ contain all infinite ω-regular words over 2^{AP} that satisfy φ. The basic steps of the LTL model checking algorithm are shown in Algorithm 1. The computational complexity of LTL model checking is shown in the following theorem.

Theorem 2 ([Vardi and Wolper, 1986, Sistla and Clarke, 1985])**.** Model checking in LTL is PSPACE-complete and can be performed in time $\mathbf{O}(|\mathcal{TS}|) \cdot 2^{\mathbf{O}|\varphi|}$.

Algorithm 1 NBA-based LTL model checking

1: **input:** $\mathcal{TS}$ and LTL formula φ
2: **output:** if $\mathcal{TS} \models \varphi$, "yes", otherwise, "no" and counterexample
3: Build NBA $\mathcal{A}$ s.t. $\mathcal{L}_\omega(\mathcal{A}) = Words(\neg\varphi)$;
4: Build the product transition system $\mathcal{TS} \otimes \mathcal{A}$;
5: **if** there exists $\pi \in Paths(\mathcal{TS} \otimes \mathcal{A})$ satisfying the accepting condition of $\mathcal{A}$ **then**
6: **return** "no" and counterexample (prefix of π);
7: **else**
8: **return** "yes";
9: **end if**

The idea of model checking for CTL is rather different than LTL, since in CTL we are concerned about the satisfaction of a formula φ in the states. That is, we need to check whether the formula φ is valid in each initial state of the transition system. The basic procedure of CTL model checking is relatively easy compared to LTL. Let $Sat(\varphi)$ be the set of all states satisfying φ and $\mathcal{I}$ the set of all initial states of transition system $\mathcal{TS}$, the procedure is given by the following two steps:

- compute $Sat(\varphi)$ recursively, and
- check whether $\mathcal{TS} \models \varphi$ if and only if $\mathcal{I} \subseteq Sat(\varphi)$.

Notice that by computing $Sat(\varphi)$ we solve a more general problem, that is, we check not only the initial states, but any state in $\mathcal{TS}$ that satisfies φ.

Computing $Sat(\varphi)$ recursively involves transforming φ into an equivalent ENF formula $\widehat{\varphi}$ (where we only use operators $\neg$, $\wedge$, EX, $\mathsf{E}\,\mathbf{U}$, EG)[6] and then construct a parse tree for formula $\widehat{\varphi}$, where the nodes of the parse tree contain the subformulae of $\widehat{\varphi}$. We compute $Sat(\psi)$, where $\psi \in Sub(\widehat{\varphi})$ and $Sub(\widehat{\varphi})$ is the set of subformulae of $\widehat{\varphi}$, in a bottom-up manner.

Example 2. Consider the following formula:

$$\widehat{\varphi} = \underbrace{\mathsf{EX}p}_{\psi} \wedge \underbrace{\mathsf{E}(q\,\mathbf{U}\,\underbrace{\mathsf{EG}\neg r}_{\psi''})}_{\psi'}.$$

The parse tree of $\widehat{\varphi}$ is shown in Figure 2.8. We can check the satisfaction of the leaves as they correspond directly to the labelling function L. We can then move upwards

[5] We can also use the product of two Kripke transition models M_1, M_2, where M_1 is the description of the system, M_2 is the description of properties to be checked. The product of these two models is given by $M_1 \cap M_2$.

[6] Recall that the previous definition of the syntax of CTL formulae uses only these operators.

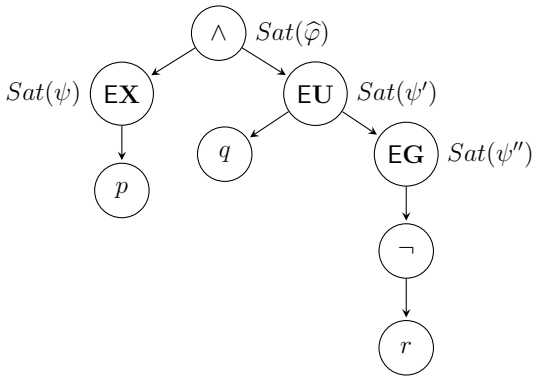

Figure 2.8: Parse tree constructed from formula $\widehat{\varphi}$.

to the parents until we reach the root of the tree. This procedure is described in Algorithm 2. Note that this algorithm works recursively on the structure of $\widehat{\varphi}$. For formulae without temporal operators, $Sat(\psi)$ can be computed directly or combined in a certain way from sets of $Sat(\psi)$. However, interesting cases arise when we deal with formulae that involve temporal operators. We need to prove the termination and correctness of proposed the algorithm.

Algorithm 2 Basic algorithm of CTL model checking

1: **input:** $\mathcal{TS}$ and CTL formula φ
2: **output:** if $\mathcal{TS} \models \varphi$, "yes", otherwise, "no" and counterexample
3: **for all** $i = 1$ to $|\widehat{\varphi}|$ **do**
4: **for all** $\psi \in Sub(\widehat{\varphi})$ s.t. $|\psi| = i$ **do**
5: compute $Sat(\psi)$ from $Sat(\psi')$ for maximal proper $\psi' \in Sub(\psi)$;
6: **end for**
7: **end for**
8: **return** $\mathcal{I} \subseteq Sat(\widehat{\varphi})$;

Without lost of generality, we may assume that the CTL formula to be verified is in ENF (see Theorem 1). Let St' be a set of states and $pre_{\mathsf{E}}(\mathrm{St}')$ the set of predecessors of St', that is $pre_{\mathsf{E}}(\mathrm{St}') = \{s \in \mathrm{St} | \exists s' \in \mathrm{St}', s \to s'\}$. It is obvious that $Sat(\mathbf{EX}\psi) = pre_{\mathsf{E}}(Sat(\psi))$, and it follows that $Sat(\mathbf{EG}\varphi) = Sat(\varphi) \cap pre_{\mathsf{E}}(Sat(\mathbf{EG}\varphi))$. The last statement seems to be circular, because we need to first compute $Sat(\mathbf{EG}\varphi)$ in order to obtain $Sat(\mathbf{EG}\varphi)$. This is where *fixpoints* play an important role as later we see in Theorem 3. But first, it is helpful to introduce some definitions.

Definition 10 (Monotone functions)**.** Let St be a non-empty set of states and $f : 2^{\mathrm{St}} \to 2^{\mathrm{St}}$ a function on the power set of St. We say that f is monotone if and only

Algorithm 3 Computation of the satisfaction sets

1: **input:** $\mathcal{TS}$ and CTL formula φ
2: **output:** $Sat(\varphi)$
3: **switch** φ **do**
4: **case** p:
5: **return** $L(p)$;
6: **case** $\neg\psi$:
7: **return** $\mathrm{St} \backslash Sat(\psi)$;
8: **case** $\psi_1 \wedge \psi_2$:
9: **return** $Sat(\psi_1) \cap Sat(\psi_2)$;
10: **case** $\mathsf{EX}\psi$:
11: **return** $pre_{\mathsf{E}}(Sat(\psi))$;
12: **case** $\mathsf{EG}\psi$: (*computing greatest fixpoint*)
13: $S_1 := \mathrm{St};\ S_2 := Sat(\psi)$;
14: **while** $S_1 \not\subseteq S_2$ **do**
15: $S_1 := S_2$;
16: $S_2 := pre_{\mathsf{E}}(S_1) \cap S_1$;
17: **end while**
18: **return** S_1;
19: **case** $\mathsf{E}(\psi_1 \,\mathbf{U}\, \psi_2)$: (*computing least fixpoint*)
20: $S_1 := \varnothing;\ S_2 := Sat(\psi_1);\ S_3 := Sat(\psi_2)$;
21: **while** $S_3 \not\subseteq S_1$ **do**
22: $S_1 := S_1 \cup S_3$;
23: $S_3 := pre_{\mathsf{E}}(S_1) \cap S_2$;
24: **end while**
25: **return** S_1;

if, for all $X, Y \in 2^{\mathrm{St}}$, $X \subseteq Y$ implies $f(X) \subseteq f(Y)$.

Definition 11 (Fixpoint)**.** Let $X \in 2^{\mathrm{St}}$ and $f : 2^{\mathrm{St}} \to 2^{\mathrm{St}}$. We say that X is a *fixpoint* of f if and only if $X = f(X)$.

Definition 12 (Iterated function)**.** Let St be a non-empty set of states and $f : 2^{\mathrm{St}} \to 2^{\mathrm{St}}$. The finite iteration of f on an input X is $f^{n+1} = f(f^n(X))$.

With some definitions above, we can then state the following theorem.

Theorem 3 (Knaster-Tarski Theorem (special case) [Tarski, 1955])**.** Let St be a finite set, $|\mathrm{St}| = n - 1$, $f : 2^{\mathrm{St}} \to 2^{\mathrm{St}}$ is a monotone function. Then:

- $f^n(\varnothing)$ is the least fixpoint of f, and
- $f^n(\mathrm{St})$ is the greatest fixpoint of f.

Proof. Since f is monotone, we have $f^1(\varnothing) \subseteq f^2(\varnothing)$. Using mathematical induction, we show that $f^i(\varnothing) \subseteq f^{i+1}(\varnothing)$ for $0 \leq i \leq n$. We claim that $\exists i$ s.t. $0 \leq i < n$ and $f^i(\varnothing)$ is a fixpoint of f. Suppose that it is not the case, then $f^1(\varnothing)$ must contain at least one element since $\varnothing \neq f^1(\varnothing)$. By the same argument, $f^2(\varnothing)$ must contain at least two elements since $f^1(\varnothing) \neq f^2(\varnothing)$. Continuing this, $f^{n+1}(\varnothing)$ must contain at least $n + 1$ elements, which leads to a contradiction.

Now suppose that X is a fixpoint of f. We show that $f^i(\varnothing) \subseteq X$. Since $\varnothing \subseteq X$, X is a fixpoint of f, and f is monotone, we have $f(\varnothing) \subseteq f(X) = X$. By induction, $\forall i,\ i \geq 0$ we get $f^i(\varnothing) \subseteq X$. So, for $i = n$, we obtain $f^n(\varnothing) \subseteq X$. Therefore, $f^n(\varnothing)$ is the least fixpoint of f.

The proof for greatest fixpoint is analogous, we simply need to replace $\subseteq$ with $\supseteq$, $\varnothing$ with St, and "greater" with "less". □

Theorem 3 not only shows that such fixpoints exist, but also provides a technique to compute them correctly. For instance, computing the least fixpoint of f, which corresponds to computing $Sat(\mathsf{E}(\varphi\,\mathbf{U}\,\psi))$ (as we later see in Theorem 5), can be done by repeatedly applying it to the empty set $\varnothing$ until the result does not change. The same procedure is needed for computing the greatest fixpoint, which corresponds to computing $Sat(\mathsf{EG}\varphi)$ (Theorem 4), but here we start from the set of all states. The procedure is guaranteed to terminate since we are working with a finite number of states. Such procedure is included in Algorithm 3. Note that, we only need the cases for EX, EG, and $\mathsf{E}\,\mathbf{U}$ since we can always express any CTL formula in ENF (Theorem 1).

Theorem 4 ([Emerson and Clarke, 1980])**.** Let $F(X) = Sat(\varphi) \cap pre_{\mathsf{E}}(X)$. Then F is monotone and $Sat(\mathsf{EG}\varphi)$ is the greatest fixpoint of F.

Proof. The proof consists of two parts:

- To show that F is monotone, let $X, Y \subseteq \mathrm{St}$ such that $X \subseteq Y$ then show that $F(X) \subseteq F(Y)$. Take $s \in X$ such that there exist some $s' \in X$ with $s' \to s$. Certainly, we also have $s' \to s$ where $s \in Y$, since $X \subseteq Y$. Thus, F is monotone.

- We have already seen that $Sat(\mathsf{EG}\varphi)$ is a fixpoint of F. Now we need to show that it is the greatest fixpoint. In order to do this, we need to show that for any set X with $F(X) = X$, we have $X \subseteq Sat(\mathsf{EG}\varphi)$. Let $s_0 \in X$, we have $s_0 \in X = F(X) = Sat(\varphi) \cap pre_{\mathsf{E}}(X)$, then $s_0 \in Sat(\varphi)$ and there exists a state $s_1 \in X$ with $s_0 \to s_1$. Since $s_1 \in X$ we may apply the previous argument to obtain $s_1 \in Sat(\varphi)$ and $s_1 \to s_2$ for some $s_2 \in X$. By induction, we can therefore construct an infinite path $s_0 \to s_1 \to s_2 \to \ldots$ such that $s_i \in Sat(\varphi)$ for all $i \geq 0$. It follows that $s_0 \in Sat(\mathsf{EG}\varphi)$ and since this applies to any $s \in X$, we have $X \subseteq Sat(\mathsf{EG}\varphi)$.

□

Theorem 5 ([Emerson and Clarke, 1980])**.** Let $G(X) = Sat(\psi) \cup (Sat(\varphi) \cap pre_{\mathsf{E}}(X))$. Then G is monotone and $Sat(\mathsf{E}(\varphi\,\mathbf{U}\,\psi))$ is the least fixpoint of G.

Proof. The proof consists of two parts:

- To show that G is monotone, let $X, Y \subseteq \mathrm{St}$ such that $X \subseteq Y$ then show that $G(X) \subseteq G(Y)$. The argument is essentially similar to the first part of proof for Theorem 4, since G performs the intersection and union of pre_{E} with constant sets $Sat(\varphi)$ and $Sat(\psi)$ respectively, and pre_{E} itself is monotone.

- Observe that $\mathsf{E}(\varphi\,\mathbf{U}\,\psi) \equiv \psi \vee (\varphi \wedge \mathsf{EXE}(\varphi\,\mathbf{U}\,\psi))$, then we have $Sat(\mathsf{E}(\varphi\,\mathbf{U}\,\psi)) = Sat(\psi) \cup (Sat(\varphi) \cap pre_{\mathsf{E}}(Sat(\mathsf{E}(\varphi\,\mathbf{U}\,\psi))))$. That means $Sat(\mathsf{E}(\varphi\,\mathbf{U}\,\psi))$ is a fixpoint of G. Let $Y = G(Y)$, we need to show that $Sat(\mathsf{E}(\varphi\,\mathbf{U}\,\psi)) \subseteq Y$. To prove this, let $s \in Sat(\mathsf{E}(\varphi\,\mathbf{U}\,\psi))$. If $s \in Sat(\psi)$ then $s \in Y$, otherwise if $s \notin Sat(\psi)$, there exists a path $\pi = s_0, s_1, s_2, \ldots$ starting in $s = s_0$ such that $\pi \models (\varphi\,\mathbf{U}\,\psi)$. Let $n > 0$, such that $s_i \models \varphi$, $0 \leq i < n$, and $s_n \models \psi$. Then we have:

  - $s_n \in Sat(\psi) \subseteq Y$,
  - $s_{n-1} \in Y$, since $s_{n-1} \in pre_{\mathsf{E}}(s_n)$ and $s_{n-1} \in Sat(\varphi)$,

– $s_{n-2} \in Y$, since $s_{n-2} \in pre_{\mathsf{E}}(s_{n-1})$ and $s_{n-2} \in Sat(\varphi)$,
  $\vdots$
– $s_0 \in Y$, since $s_0 \in pre_{\mathsf{E}}(s_1)$ and $s_0 \in Sat(\varphi)$

Therefore it follows that $s = s_0 \in Y$. □

The theorems above ensure the correctness of Algorithm 2 and 3. For the complexity of CTL model checking, we refer to the following theorem.

Theorem 6 ([Clarke et al., 1986, Arnold and Crubille, 1988])**.** The model checking problem for CTL can be solved in time $\mathbf{O}(|\mathcal{TS}| \cdot |\varphi|)$.

This section presents some model checking techniques for LTL and CTL. These approaches are arguably the most intuitive ones. There are, of course, other various alternative approaches that provide more efficient techniques. For instance, some alternatives approaches for LTL model checking can be found in [Gerth et al., 1995, Etessami and Holzmann, 2000, Daniele et al., 1999, Gastin and Oddoux, 2001, Giannakopoulou and Lerda, 2002, Fritz and Wilke, 2002]. For CTL model checking, the most widely used technique is symbolic model checking using *binary decision diagrams* [Burch et al., 1990, McMillan, 1993b].

At this point we have discussed about some (of the most common) techniques for model checking LTL and CTL properties. These approaches provide some underpinning techniques that will be used to solve some problems that emerge from reasoning about strategies in multiplayer games. The importance of these approaches will be made clear in the following section.

2.3 Strategic Ability of Agents

This section presents different logics that can be used for reasoning about strategies of players/agents in game-like settings. Previous sections present LTL and CTL (and CTL*) for reasoning about computations of a system which behaviour is completely deterministic with respect to the state of the system. However, in the domain of multi-agent systems, a system is viewed as a non-deterministic system, where agents in the system have different choices available to them. More importantly, agents are assumed to act strategically and rationally in order to satisfy their goals or preferences. Thus, it is important to know which agent (or set of agents) can make the system behave in a particular way.

2.3.1 Strategic Behaviour of Players

Game theory is a field in mathematical economics that is used to study strategic interactions between self-interested entities [Maschler et al., 2013]. It is argued in [Nisan et al., 2007, Shoham and Leyton-Brown, 2008] that game theory provides an appropriate underlying concept and analytical framework for strategic reasoning of players in multi-agent systems.

Generally, the models of games used in game theory can be divided into two categories: *cooperative* and *non-cooperative*. The two categories differ in how they define interaction and interdependence among players. In cooperative game theory, the outcomes of the game come from the possible joint actions of sets of players. In non-cooperative game theory, players decide their actions independently and these actions may affect the outcomes directly.

Definition 13 (Strategic game form)**.** A strategic game form is a tuple $\Gamma = (\mathrm{N}, \{\mathrm{Ac}_1, \ldots, \mathrm{Ac}_n\}, \Omega, \mathsf{o})$, where $\mathrm{N} = \{1, \ldots, n\}$ is a finite set of agents or players, Ac_i is a set of actions (or choices) for player $i \in \mathrm{N}$, $\Omega = \{\omega_1, \ldots, \omega_n\}$ is a set of outcomes, and $\mathsf{o} : \mathrm{Ac}_1 \times \cdots \times \mathrm{Ac}_n \to \Omega$ is an outcome function.

Agents are assumed to act strategically and rationally in order to satisfy their goals, therefore it is natural to assume that agents have preferences over the outcomes of the game. To express the preferences, each agent i is associated with a *utility function* $u_i : \Omega \to \mathbb{R}$ assigning every outcome to a real number. The larger the number means the better the outcome from an agent i's perspective. We can then define a preference relation $\preceq_i$ over outcomes for each player i as follows:

$$\omega \preceq_i \omega' \iff u_i(\omega) \leq u_i(\omega').$$

This definition implies that the relation is complete, reflexive, and transitive.

2.3.2 Strategies in Concurrent Games

In the real world, agents interactions often happen in a repeated setting, where a base game $\mathcal{G}$ is played a number of times and the payoffs are totalled. This type of setting is usually called *iterated game* or *repeated game*. *Concurrent game structures* allow multi-player games to be played repeatedly with different strategies at different stages. The outcome of every round affects the strategies to be played in the next round.

A concurrent game structure allows us to model interactions of the players concurrently instead of sequentially (such as in an *extensive-form game* [Kuhn, 2003]). A state transition in a concurrent game structure is the result of choices made (simultaneously) by the players in the system. In this way, we can describe open systems properly.

Definition 14 (Concurrent game structure (CGS))**.** A *concurrent game structure* (CGS) is a tuple

$$\mathcal{M} = (\mathrm{N}, (\mathrm{Ac}_i)_{i\in\mathrm{N}}, \mathrm{St}, s_0, \mathsf{tr}, \lambda)$$

where $\mathrm{N} = \{1, \ldots, n\}$ is a set of *players*, each Ac_i is a set of *actions*, St is a set of *states*, with a designated *initial* state s_0. With each player $i \in \mathrm{N}$ and each state $s \in \mathrm{St}$, we associate a non-empty set $\mathrm{Ac}_i(s)$ of *available* actions that, intuitively, i can perform when in state s. We refer to a profile of actions $\vec{a} = (a_1, \ldots, a_n) \in \vec{\mathrm{Ac}} = \mathrm{Ac}_1 \times \cdots \times \mathrm{Ac}_n$ as a *direction*. We also consider *partial* directions. A direction $\vec{a}$ is available in state s if for all i we have $a_i \in \mathrm{Ac}_i(s)$. Write $\vec{\mathrm{Ac}}(s)$ for the set of available directions in state s. For a given set of players $A \subseteq \mathrm{N}$ and an action profile $\vec{a}$, we let $\vec{a}_A$ and $\vec{a}_{-A}$ be two tuples of actions, respectively, one for each player in A and one for each player in $\mathrm{N} \setminus A$. We also write $\vec{a}_i$ for $\vec{a}_{\{i\}}$ and $\vec{a}_{-i}$ for $\vec{a}_{\mathrm{N}\setminus\{i\}}$. Furthermore, for two directions $\vec{a}$ and $\vec{a}'$, we write $(\vec{a}_A, \vec{a}'_{-A})$ to denote the direction where the actions for players in A are taken from $\vec{a}$ and the actions for players in $\mathrm{N} \setminus A$ are taken from $\vec{a}'$. Finally, tr is a *deterministic transition function*, which associates a state s and every available direction $\vec{a}$ in s a state $s' \in \mathrm{St}$, and $\lambda : \mathrm{St} \to 2^{\mathrm{AP}}$ is a labelling function. Whenever there is $\vec{a}$ such that $\mathsf{tr}(s, \vec{a}) = s'$, we say that s' is *accessible* from s. By π_k we refer to the $(k+1)$-th state in π and by $\pi_{\leq k}$ to the (finite) prefix of π up to the $(k+1)$-th element. An *action profile run* is an infinite sequence $\eta = \vec{a}_0, \vec{a}_1, \ldots$ of action profiles. Note that, since $\mathcal{M}$ is deterministic (*i.e.*, the transition function tr is deterministic), for a given state s_0, an action profile run uniquely determines the path π in which, for every $k \in \mathbb{N}$, $\pi_{k+1} = \mathsf{tr}(\pi_k, \vec{a}_k)$.

The definition of path in CGS is similar to the one in temporal models. A *finite* path is a finite sequence $h = s_0, s_1, \ldots, s_n$. A strategy of a player i in a CGS $\mathcal{M}$ is a plan that tells what action should be taken by i in each possible situation. Strategies may be classified into three kinds: *memoryless*, *perfect recall*, and intermediate types between the former two.

Formally, a *memoryless* strategy can be represented by a function $\sigma_i : \mathrm{St} \to \mathrm{Ac}_i$ such that $\sigma_i(s) \in \mathrm{Ac}_i(s)$. Informally, memoryless strategy is a model of strategy that depends *only* on the current state of play. A *perfect recall* strategy is represented

by a function $\sigma_i : \mathrm{St}^+ \to \mathrm{Ac}_i$, where St^+ is a finite sequence of states (i.e., a finite path), such that $\sigma_i(\langle \ldots, s \rangle) \in \mathrm{Ac}_i(s)$. This means that an agent with a perfect recall strategy acts based on the full history of a play, from the initial state until the current state of the play. An exposition and definition of types of strategies that lie between the previous two can be (among the others) found in [Agotnes and Walther, 2009] and [Vester, 2013]. The former studies intermediate types of strategies for agents with (bounded) finite memory, whereas the latter discusses finite memory strategies both in bounded and unbounded settings.

2.4 Logics for Strategies

This section presents some formalisms that allow us to reason about strategies of players. Specifically, this section presents two of the most influential logics for reasoning about strategies in multi-agent settings: *Alternating-time Temporal Logic* and *Strategy Logic*.

2.4.1 Alternating-time Temporal Logic

Alternating-time Temporal Logic (ATL^*) is one of the most important logics in which one can reason about time and strategic abilities of players [Alur et al., 2002]. ATL^* is closely related to CTL^*, in which path quantifiers are replaced with the strategic quantifiers $\langle\langle A \rangle\rangle$ in which $A \subseteq \mathrm{N}$ denotes a set of agents who act as a *coalition*, where N is the set of all agents. A formula $\langle\langle A \rangle\rangle \varphi$ expresses that *coalition* A has a strategy to ensure that the temporal property φ holds irrespective of how other players outside of A proceed. The strategic quantifier $\langle\langle A \rangle\rangle$ is basically a generalisation of branching-time temporal logics described as follows:

- The universal path quantifier A correponds to $\langle\langle \varnothing \rangle\rangle$,
- The existential path quantifier E corresponds to $\langle\langle \mathrm{N} \rangle\rangle$,

where N denotes the set of all agents and $\varnothing$ is the empty set.

Formally, ATL^* is defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\langle A \rangle\rangle \psi,$$
$$\psi ::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \psi \,\mathbf{U}\, \psi$$

where $A \subseteq \mathrm{N}$ and $p \in \mathrm{AP}$.

Write $out(s, \sigma_A)$ to denote the set of all paths $\pi \in St^\omega$ that the players in A enforce when they execute strategy σ_A from state s onward. Let Σ be the set of all strategies, the semantics of ATL*, interpreted over a CGS $\mathcal{M}$, is given by:

$$
\begin{array}{lll}
\mathcal{M}, s \models p & \text{iff} & p \in \lambda(s) \\
\mathcal{M}, s \models \neg\varphi & \text{iff} & \mathcal{M}, s \not\models \varphi \\
\mathcal{M}, s \models \varphi_1 \wedge \varphi_2 & \text{iff} & \mathcal{M}, s \models \varphi_1 \text{ and } \mathcal{M}, s \models \varphi_2 \\
\mathcal{M}, s \models \langle\langle A \rangle\rangle \psi & \text{iff} & \text{there is a collective strategy } \sigma_A \in \Sigma \text{ s.t. } \mathcal{M}, \pi \models \psi \\
 & & \text{for all } \pi \in out(s, \sigma_A)
\end{array}
$$

The semantics of path formulae is essentially the same as to the one of LTL:

$$
\begin{array}{lll}
\mathcal{M}, \pi \models \varphi & \text{iff} & \mathcal{M}, \pi[0] \models \varphi \\
\mathcal{M}, \pi \models \neg\psi & \text{iff} & \pi \not\models \psi \\
\mathcal{M}, \pi \models \psi_1 \wedge \psi_2 & \text{iff} & \mathcal{M}, \pi \models \psi_1 \text{ and } \mathcal{M}, \pi \models \psi_2 \\
\mathcal{M}, \pi \models \mathbf{X}\psi & \text{iff} & \pi[1...] \models \psi \\
\mathcal{M}, \pi \models \psi_1 \,\mathbf{U}\, \psi_2 & \text{iff} & \pi[j...] \models \psi_2 \text{ for some } j \geq 0, \text{ and } \pi[i...] \models \psi_1 \text{ for all } 0 \leq i < j
\end{array}
$$

ATL (sometimes called "vanilla" ATL) is a strict subset of ATL*. It is the alternating-time extension of CTL (in the analogous way of ATL* to CTL*). A model checking procedure of ATL properties is discussed in a later section.

2.4.2 Strategy Logic

Strategy Logic (SL) was first introduced by Chatterjee *et al.* in [Chatterjee et al., 2010b] with the aim of developing a simple and natural way for reasoning explicitly about strategies. However, it was defined and investigated only under the framework of two-players games. Soon afterwards, Mogavero *et al.* [Mogavero et al., 2010] introduced a more general framework of SL to reason about multi-player concurrent games.

SL quantifies over strategies explicitly in formulae instead of over agents. This allows us to express properties such as *"agent i and j share the same strategy"*. It is even strong enough to express game-theoretic concepts like Nash equilibrium. SL extends the syntax of LTL with three new operators: an existential strategy quantifier $\langle\langle x \rangle\rangle$, a universal strategy quantifier $[\![x]\!]$, and an agent binding operator (i, x), where

i is an agent and x a strategy variable. Intuitively, these new operators can be read as *"there exists a strategy x"*, *"for all strategies x"*, and *"bind agent i to the strategy associated with variable x"*, respectively.

SL formulae are built inductively from a set of Boolean variables Φ, strategy variables Var, and a set of agents N, by using the following BNF:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \langle\langle x\rangle\rangle\varphi \mid [\![x]\!]\varphi \mid (i,x)\varphi$$

where $p \in \Phi$, $x \in Var$, and $i \in \mathrm{N}$.

Write $free(\varphi)$ to denote the set of *free agents and variables* of a formula φ. For instance, let $\varphi = \langle\langle y\rangle\rangle(\alpha,x)(\beta,y)\mathbf{G}p$ be a formula on agents $\mathrm{N} = \{\alpha,\beta,\gamma\}$. Since agent γ is not used in the formula and x is not quantified, then we have $free(\varphi) = \{\gamma, x\}$.

Let Var be a fixed set of variables. An *assignment* $\chi : \mathrm{N} \cup Var \to \Sigma$ is a partial function mapping every agent and variable to a strategy. Write $dom(\chi)$ for the subset of $\mathrm{N} \cup Var$, then an assignment χ is complete if and only if $\mathrm{N} \subseteq dom(\chi)$. SL formulae are evaluated at a state s of a CGS $\mathcal{M}$, under an assignment χ. Write $\mathcal{M}, \chi, s \models \varphi$ to indicate that formula φ holds at s under assignment χ.

Let s be a state of $\mathcal{M}$, χ be an assignment, $x \in Var$, and $\varphi \in$ SL, such that $free(\varphi)\backslash\{x\} \subseteq dom(\chi)$. The semantics of the SL formulae involving boolean and LTL temporal operators is defined as usual (see Section 2.2.1). The semantics of SL that involves quantifications and bindings is defined as follows:

$$\begin{array}{lll} \mathcal{M}, \chi, s \models \langle\langle x\rangle\rangle\varphi & \text{iff} & \exists\sigma \in \Sigma \text{ such that } \mathcal{M}, \chi[x \mapsto \sigma], s \models \varphi \\ \mathcal{M}, \chi, s \models [\![x]\!]\varphi & \text{iff} & \forall\sigma \in \Sigma \text{ it holds that } \mathcal{M}, \chi[x \mapsto \sigma], s \models \varphi \end{array}$$

Additionally, if $free(\varphi) \cup \{x\} \subseteq dom(\chi) \cup \{a\}$ where $i \in \mathrm{N}$, then:

$$\mathcal{M}, \chi, s \models (i,x)\varphi \qquad \text{iff} \qquad \mathcal{M}, \chi[i \mapsto \chi(x)], s \models \varphi$$

2.5 Verifying Strategies

This section presents a model checking procedure for (vanilla) ATL and SL. ATL is a subset of ATL* in which every occurence of the cooperation modality is coupled with a temporal operator.

2.5.1 Model Checking ATL

The basic idea of ATL model checking is to verify an ATL formula $\langle\langle A\rangle\rangle\varphi$ by constructing a winning strategy for coalition A, that is, a strategy that guarantees the satisfaction of φ irrespective of what other agents outside coalition A do. This is carried out by using similar algorithms for CTL model checking where we start from set of all states for **G** and empty set for **U**. Thus, model checking an ATL property has similar complexity to model checking a CTL property, as shown in the following theorem.

Algorithm 4 Computation of the satisfaction sets

function ATLMC($\mathcal{M}, \varphi$)
  switch φ **do**
    case p:
      return $\lambda(p)$;
    case $\neg\psi$:
      return $\mathrm{St}\backslash\textsc{atlmc}(\mathcal{M}, \psi)$;
    case $\psi_1 \wedge \psi_2$:
      return $\textsc{atlmc}(\mathcal{M}, \psi_1) \cap \textsc{atlmc}(\mathcal{M}, \psi_2)$;
    case $\langle\langle A\rangle\rangle\mathbf{X}\psi$:
      return $\textsc{pre}(\mathcal{M}, A, \textsc{atlmc}(\mathcal{M}, \psi))$;
    case $\langle\langle A\rangle\rangle\mathbf{G}\psi$: (*computing greatest fixpoint*)
      $S_1 := \mathrm{St};\ S_2 := \textsc{atlmc}(\mathcal{M}, \psi);\ S_3 := S_2;$
      while $S_1 \not\subseteq S_2$ **do**
        $S_1 := S_2;$
        $S_2 := \textsc{pre}(\mathcal{M}, A, S_1) \cap S_3;$
      end while
      return S_1;
    case $\langle\langle A\rangle\rangle(\psi_1\,\mathbf{U}\,\psi_2)$: (*computing least fixpoint*)
      $S_1 := \varnothing;\ S_2 := \textsc{atlmc}(\mathcal{M}, \psi_1);\ S_3 := \textsc{atlmc}(\mathcal{M}, \psi_2);$
      while $S_3 \not\subseteq S_1$ **do**
        $S_1 := S_1 \cup S_3;$
        $S_3 := \textsc{pre}(\mathcal{M}, A, S_1) \cap S_2;$
      end while
      return S_1;
end function

function PRE($\mathcal{M}, A, S$)
  return $\{s \mid \exists a_A \forall a_{\mathrm{N}\backslash A}, \mathsf{tr}(s, (a_A, a_{\mathrm{N}\backslash A})) \in S\}$ where a_A is collective action of coalition A;
end function

Theorem 7 ([Alur et al., 2002])**.** Model checking ATL is P-complete, and can be performed in time $\mathbf{O}(|\mathcal{M}| \cdot |\varphi|)$, where $|\mathcal{M}|$ is the number of transitions in $\mathcal{M}$, and $|\varphi|$ is the number of subformulae in φ.

The model checking procedure, essentially consists in, finding the greatest and least fixpoint for ATL formula with the temporal operators $\mathbf{G}$ and $\mathbf{U}$, respectively. The fixpoint characterisation of ATL operators is given by:

$$\langle\langle A\rangle\rangle \mathbf{G}\varphi \iff \varphi \wedge \langle\langle A\rangle\rangle \mathbf{X}\langle\langle A\rangle\rangle \mathbf{G}\varphi$$
$$\langle\langle A\rangle\rangle (\varphi_1 \,\mathbf{U}\, \varphi_2) \iff \varphi_2 \vee (\varphi_1 \wedge \langle\langle A\rangle\rangle \mathbf{X}\langle\langle A\rangle\rangle (\varphi_1 \,\mathbf{U}\, \varphi_2))$$

Note that the fixpoint characterisation for ATL is similar to CTL, thus it is a straightforward adaptation of the CTL fixpoint algorithms presented in Section 2.2.4. However, there are some important differences compared to the previous implementation, especially in function PRE where transition function tr of $\mathcal{M}$ that respects the collective actions is used as a criterion for states reachability. The complete procedure for model checking an ATL formula is shown in Algorithm 4. Note that formulae with operators other than $\mathbf{X}$, $\mathbf{G}$, and $\mathbf{U}$ can be derived from these.

2.5.2 Model Checking SL

The model checking procedure for SL extends the ones used for temporal logic in two aspects. First, it takes as input a formula to be checked along with *bindings*, which assign agents to some variables. Second, it returns not only sets of states, but sets of pairs $\langle s, \chi\rangle$ consisting of state s and an assignment of variables to strategies χ. A pair $\langle s, \chi\rangle \in Ext$ is called an *extended state*, which can be read as: with the strategy assignment χ, the formula holds at state s.

Given an SL formula φ and a binding $\flat \in Bnd$, where $Bnd : \mathrm{N} \to Var$, the model checking algorithm $Sat : \mathsf{SL} \times Bnd \to 2^{Ext}$, returning a set of extended states, is shown in Algorithm 5, where $i \in \mathrm{N}$ is an agent, $A \subseteq \mathrm{N}$ a set of agents, $x \in Var$ a variable, $h(p)$ the set of global states where p is true, and $pre(S, \flat)$ is the set of extended states that temporally precede S under binding $\flat$.

The complexity of model checking for an SL formula φ (under perfect recall and complete information) with k number of alternation is k-EXPSPACE-hard and $(k+1)$-EXPTIME [Mogavero et al., 2010]. However, it is PTIME with respect to the size of the model [Mogavero et al., 2010]. This means that, given the computational power of today's computers, model checking a simple SL formula in a large model is feasible, but model checking a complex SL formula in a relatively small model would

Algorithm 5 Computation of the satisfaction sets (SL)

1: **function** SLMC($\mathcal{M}, \varphi, b$)
2: **switch** φ **do**
3: **case** p:
4: **return** $\{\langle s, \chi\rangle | s \in h(p), \chi \in Asg\}$;
5: **case** $\neg\psi$:
6: **return** NEG(SLMC($\mathcal{M}, \varphi, b$));
7: **case** $\psi_1 \wedge \psi_2$:
8: **return** SLMC($\mathcal{M}, \psi_1, b$) $\cap$ SLMC($\mathcal{M}, \psi_2, b$);
9: **case** $(i, x)\psi$:
10: **return** SLMC($\mathcal{M}, \psi, b[i \mapsto x]$);
11: **case** $\langle\langle x\rangle\rangle\psi$:
12: **return** $\{\langle s, \chi\rangle | \exists \sigma \in \Sigma_{sharing(\psi,x)}.\langle s, \chi[x \mapsto \sigma]\rangle \in$ SLMC($\mathcal{M}, \psi, b$)$\}$;
13: **case** $\mathbf{X}\psi$:
14: **return** pre(SLMC($\mathcal{M}, \psi, b$), b);
15: **case** $(\psi_1 \,\mathbf{U}\, \psi_2)$: (*computing least fixpoint*)
16: $S_1 := \varnothing$; $S_2 :=$ SLMC($\mathcal{M}, \psi_1, b$); $S_3 :=$ SLMC($\mathcal{M}, \psi_2, b$);
17: **while** $S_3 \not\subseteq S_1$ **do**
18: $S_1 := S_1 \cup S_3$;
19: $S_3 := pre(S_1, b) \cap S_2$;
20: **end while**
21: **return** S_1;
22: **end function**
23:
24: **function** NEG(S)
25: **return** $\{\langle s, \chi\rangle \mid \forall \langle s, \chi'\rangle \in S, \exists x \in dom(\chi) \cap dom(\chi'), \exists s' \in$ St.
$\chi(x)(s') \neq \chi'(x)(s')\}$;
26: **end function**

be impractical. In response to this, five different syntactic fragments of SL have been proposed [Mogavero et al., 2010, Mogavero et al., 2012]. To make it clearer, consider the following Boolean-Goal Strategy Logic [Mogavero et al., 2014] formula:

$$\psi = \underbrace{[\![x]\!]\langle\!\langle y\rangle\!\rangle}_{\wp}\underbrace{(a,x)(b,y)}_{\flat_1}\underbrace{\mathbf{F}p}_{\varphi_1} \wedge \underbrace{(a,y)(b,x)}_{\flat_2}\underbrace{(\mathbf{G}q)}_{\varphi_2}$$

where ψ is an SL formula, φ_i and φ_2 LTL formulae, $\wp$ a strategy quantification, and $\flat$ an agent binding prefix. The syntactic fragments of SL differ in which operators can occur between $\wp$ and $\flat\varphi$. More specifically, the syntax of each fragment [Mogavero et al., 2012, Mogavero et al., 2014] is as follows:

- Nested-Goal Strategy Logic (SL[NG]).
  Syntax: $\psi ::= \neg\psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \psi\,\mathbf{U}\,\psi \mid \flat\psi \mid \varphi$
- Boolean-Goal Strategy Logic (SL[BG]).
  Syntax: $\psi ::= \neg\psi \mid \psi \wedge \psi \mid \flat\varphi$
- Conjunctive-Goal Strategy Logic (SL[CG]).
  Syntax: $\psi ::= \psi \wedge \psi \mid \flat\varphi$
- Disjunctive-Goal Strategy Logic (SL[DG]).
  Syntax: $\psi ::= \psi \vee \psi \mid \flat\varphi$
- One-Goal Strategy Logic (SL[1G]).
  Syntax: $\psi ::= \flat\varphi$

where ψ is an SL formula, φ an LTL formula, $\wp$ a strategy quantification, and $\flat$ an agent binding prefix. Observe that SL[CG] and SL[DG] are essentially duals of each other.

The model checking complexity for each fragment of SL is summarised in Table 2.1, k is the maximum number of quantification alternation [Mogavero et al., 2014, Bouyer et al., 2015b]. There is also a variant of SL called Strategy Logic with Knowledge (SLK) [Cermák et al., 2014], which is defined on imperfect recall semantics with incomplete information.

2.6 Recent Developments of Model Checking Tools

This section presents some multi-agent systems verification tools that are actively being developed.

fragment	model checking complexity
SL[NG]	k-EXPSPACE-hard (lower bound), $(k+1)$-EXPTIME (upper bound)
SL[BG]	TOWER-complete
SL[CG]	2EXPTIME-complete
SL[DG]	2EXPTIME-complete
SL[1G]	2EXPTIME-complete

Table 2.1: Overview of complexity SL fragments with respect to the size of specification.

2.6.1 PRISM-games

PRISM-games [Chen et al., 2013b, PRISM Web, 2019b] is an extension of PRISM (Probabilistic Symbolic Model Checker) [Kwiatkowska et al., 2011, PRISM Web, 2019a] for modelling, verifying, and synthesising strategies in probabilistic systems that incorporate competitive or collaborative behaviour. Systems are modelled as *stochastic multi-player games* (SMGs) [Shapley, 1953]. PRISM-games uses a probabilistic extension of the Reactive Modules Language (RML) [Alur and Henzinger, 1999] and an extension of ATL called rPATL [Chen et al., 2012]. rPATL combines the coalition operator $\langle\langle A \rangle\rangle$ of ATL, the probabilistic operator $\mathcal{P}_{\sim\lambda}$ from PCTL [Hansson and Jonsson, 1994], and an operator $\mathcal{R}^r_{\sim x}$, which corresponds to the cumulative "reward" that a player can achieve along a path of the game. Intuitively, the rewards are represented by *reward functions* that map each possible path in the game to a cumulative reward value (see [Chen et al., 2012, Chen et al., 2013a]).

The current release of PRISM-games is built upon the explicit model checker version of PRISM which implements explicit-state data structures. However, it currently only supports turn-based, perfect-information SMGs which nevertheless is sufficient to model and analyse particular kinds of energy management and collective decision making in autonomous systems [Chen et al., 2012].

Figure 2.9 shows a toy example[7] of an SMG described in the PRISM modelling language. Notice the keyword `smg` at the beginning of the code, which differentiates from PRISM's usual modelling of Markov decision processes (MDPs). This particular SMG has two players: `p1` and `p2`. The former controls asynchronous transitions from

[7]Taken from [PRISM Web, 2019c].

```
1   smg
2
3   player p1
4   host, [send1], [send2]
5   endplayer
6
7   player p2
8   client
9   endplayer
10
11  module host
12  h : [0..2] init 0;
13  [send1] h=0 -> (h'=1); // send message 1
14  [send2] h=0 -> (h'=2); // send message 2
15  [] c=0 -> (h'=0); // restart
16  endmodule
17
18  module client
19  c : [0..2] init 0;
20  [send1] c=0 -> 0.85 : (c'=1) + 0.15 : (c'=0); // receive message 1
21  [send2] c=0 -> 0.85 : (c'=2) + 0.15 : (c'=0); // receive message 2
22  [] c!=0 -> (c'=0); // request another message
23  [] c!=0 -> true; // wait
24  endmodule
```

Figure 2.9: Example of an SMG described in PRISM-games modelling language.

```
1   Model checking: <<p1>> P>=0.99 [ F<=5 c=2 ]
2
3   Starting bounded probabilistic reachability...
4   Bounded probabilistic reachability (maxmin) took 5 iterations and 0.0
        seconds.
5
6   Number of states satisfying <<p1>> P>=0.99 [ F<=5 c=2 ]: 2
7
8   Property satisfied in 1 of 1 initial states.
9
10  Time for model checking: 0.0 seconds.
11
12  Result: true (property satisfied in the initial state)
```

Figure 2.10: PRISM-games verification output of property φ.

module host and synchronous transitions `send1` and `send2`. The latter controls asynchronous transitions from module client. Property $\varphi =$ `<<p1>> P>=0.99 [ F<=5 c=2 ]` states that `p1` has a strategy to ensure that the probability of reaching a state satisfying `c=2` in 5 time-steps is at least 0.99 which PRISM-games verifies that it is indeed true as shown in the output log in Figure 2.10.

2.6.2 MCK

MCK is a tool for model-checking the logic of knowledge of multi-agent systems developed at the University of New South Wales [Gammie and van der Meyden, 2004, MCK Web, 2019a]. MCK supports *explicit-state* model checking (as described in Section 2.2.4), symbolic model checking via *binary decision diagrams* (BDD) [McMillan, 1993a], and *bounded model checking* (BMC) [Biere et al., 2003].

MCK models the environment as a finite-state transition system where the transitions are labelled by agents' actions. Let $\mathrm{N} = \{1, \ldots, n\}$ be a set of agents, ACT_i a set of of actions associated with agent i. To model non-determinism, the environment also has a set of action ACT_e. A joint action $ACT = ACT_e \times ACT_1 \times \cdots \times ACT_n$ is the product of environment's and each agent's actions.

Definition 15 (Interpreted environment)**.** A finite interpreted environment for n agents is a tuple $E = (S_e, I_e, P_e, \tau, O_1, \ldots, O_n, L_e)$ where the components are as follows:

- S_e is a finite set of states of the environment.
- $I_e \subseteq S_e$ is the possible initial states of the environment.
- $P_e : S_e \to \mathcal{P}(ACT_e)$ is a function *protocol of the environment* mapping states to subset of ACT_e.
- τ is a function mapping joint actions $a \in ACT$ to transition function $\tau(a) : S_e \to S_e$.
- O_i is the *observation function of agent* i, mapping the set of state S_e to some set $\mathcal{O}$. That is, $O_i(s)$ will be called the *observation* of agent i in state $s \in S_e$.
- $\lambda_e : S_e \to 2^{AP}$ is an *labelling* function mapping each state to an assignment of truth values for the atomic propositions AP.

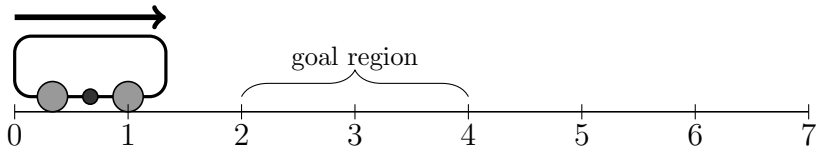

Figure 2.11: The illustration of Example 3.

A path r in E is an infinite sequence $r = s_0, s_1, \ldots$ such that $s_0 \in I_e$ and for all $m \geq 0$ there exists a joint action $a = \langle a_e, a_1, \ldots, a_n \rangle$ such that $a_e \in P_e(s_m)$ and $s_{m+1} = \tau(a)(s_m)$. A *point* is a tuple (r, m) that identifies a particular instant of time in r. MCK follows [Fagin et al., 1995] and defines a system to be a set $\mathcal{R}$ of runs and an interpreted system a tuple $\mathcal{I} = (\mathcal{R}, \lambda)$ consisting of a system $\mathcal{R}$ and labelling function λ mapping the points of runs $r \in \mathcal{R}$ to assignments of atomic proposition AP.

MCK supports several logics for specifying properties, namely LTL, CTL*, and the μ-calculus, as well as incorporating some knowledge modalities such as *Knows* and *CK* (*common knowledge*). To make the mechanism of MCK clearer consider Example 3 which is taken from [MCK Web, 2019b] (originally from [Brafman et al., 1997]). Figure 2.12 shows the example described in MCK input language.

Example 3. This example is a summary of [Engelhardt et al., 2000] which was originally introduced in [Brafman et al., 1997]. Consider a robot travelling along an endless corridor, which is identified with natural numbers (see Figure 2.11). The robot starts at 0 and has the goal of stopping in the goal region {2,3,4}. To determine when to stop, the robot has a sensor between its front and rear axles. It is assumed that the sensor is inaccurate with error of at most 1. The robot can only perform the action "halt", which results in instantaneous stopping. Unless it performs action halt, the robot cannot control its movement for each time-step.

The solution of this problem is straightforward: we have to set the robot's control policy into: *Do nothing while the sensor has a value of less than 3, and halt as soon as it reads value 3 or more.* The code in Figure 2.12 sets the arena to have 8 distinct locations[8], hence `type Pos={0..7}`. Agent Robot has a non-deterministic transition, at each time-step, it is either staying in the same position (Line 16) or traveling one step forward (Line 21) while always generating sensor reading with error bounds captured (non-deterministically) in Line 26-28. Line 32 rules out the traces where the environment never tries to move the robot forward. Line 33 asserts the

[8]The choice of 8 is arbitrary, any number greater than 4 is sufficient.

```
type Pos = {0. .7}

incpos : Bool
position: Pos
sensor : Pos
halted : Bool

init_cond = incpos /\ position == 0 /\ sensor == 0 /\ neg halted

agent Robot "robot" ( sensor )

transitions
begin
if neg halted /\ neg Robot.Halt >
begin
position := position;
incpos := False
end
[] neg halted /\ neg Robot.Halt >
begin
position := position + 1;
incpos := True
end
[] Robot.Halt  > halted := True
fi;
if True   > sensor := position   1
[] True   > sensor := position
[] True   > sensor := position + 1
fi
end

fairness = incpos \/ halted
spec obs = AG (Robot.test <=> Knows Robot position in {2..4})
protocol "robot" (sensor : observable Pos)
define test = sensor >= 3

begin
do neg test   > skip
[] break   > <<Halt>>
od
end
```

Figure 2.12: MCK input for Example 3.

```
1  Wed Sep 21 2016 13:00:16
2  /localhost/MCK/mck-Linux-1.1.0/examples/robot
3  spec_obs_ctl =
4  AG (Robot.test (Knows Robot (position in {2,3,4})))
5
6  >> Spec holds.
7  Timing (in seconds):
8  real 0.01
9  user 0.00
10 sys 0.00
```

Figure 2.13: MCK output for Example 3.

specification of the agent `Robot`, that is, it is always the case `Robot.test` is true if and only if the Robot knows it is in between 2 to 4, where `Robot.test` is defined as true when the reading of the sensor is greater or equal than 3. Indeed, MCK confirms the specification holds as shown in Figure 2.13.

2.6.3 MCMAS

MCMAS [Lomuscio and Raimondi, 2006, MCMAS Web, 2019] adopts interpreted systems [Fagin et al., 1995] as the formal language to represent systems comprised of multiple entities. In MCMAS, interpreted systems are extended to incorporate game theoretic notions such as those provided by ATL modalities [Lomuscio et al., 2017]. Although both MCK and MCMAS adapted the formalisation in [Fagin et al., 1995], there are some subtle differences between the two adaptations. The approach used in MCK is "top-down", where local states (MCK uses *observations*) of agents are defined by an *observation function* that maps the set of environment/global state S_e to local states/observations of the agents. On the other hand, MCMAS approach is "bottom-up", where the global state is defined as a tuple of the local states of the agents. In this setting, global states are given as the composition of local states of the agents and environment. The definition of *interpreted systems* in MCMAS is as follows.

Definition 16 (Interpreted systems (IS) [Fagin et al., 1995])**.** An interpreted system is a tuple $IS = (\{L_i, Act_i, P_i, \tau_i\}_{i\in\mathrm{N}}, I, h)$, where L_i is a finite set of possible local states for agent i, Act_i is a finite set of possible actions for agent i, $P_i : L_i \to 2^{Act_i \setminus \varnothing}$ is a local protocol function for agent i mapping possible actions for each local state, $\tau_i : L_i \times Act_1 \times \cdots \times Act_n \to L_i$ is a deterministic local transition function associating a successor local state for agent i after executing a joint action at a local state, $I \subseteq L_0 \times L_1 \times \cdots \times L_n$ is the set of initial global states, and $h \subseteq L_0 \times L_1 \times \cdots \times L_n \times AP$.

```
Agent m0
Vars:
u0 : boolean;
d0 : boolean;
end Vars
Actions = {val00,val01,val10,val11};
Protocol:
(u0=true and d0=true) : {val01,val10};
(u0=true and d0=false) : {val01,val10};
(u0=false and d0=true) : {val01,val10};
(u0=false and d0=false) : {val01,val10};
end Protocol
Evolution:
(u0=false and d0=false) if Action=val00;
(u0=true and d0=false) if Action=val10;
(u0=false and d0=true) if Action=val01;
(u0=true and d0=true) if Action=val11;
end Evolution
end Agent
```

Figure 2.14: Example of an agent in ISPL

```
Evaluation
u0 if m0.u0=true;
d0 if m0.d0=true;
u1 if m1.u1=true;
d1 if m1.d1=true;
end Evaluation
```

Figure 2.15: Evaluation of atomic variables

We say that $G = L_0 \times \cdots \times L_n$ is the set of possible global states for the system and $ACT = Act_1 \times \cdots \times Act_n$ the set of joint actions.

MCMAS uses a dedicated programming language called *Interpreted Systems Programming Language* (ISPL)[9] to describe the specification of IS. ISPL specifies a multi-agent system as an *Environment agent* and a set of *normal agents* similar to Definition 16. The Environment and normal agents have two sets of variables: *local* and *observable*. Local variables are private, while observable variables are visible to all agents in the system. Every agent also has a set of *local actions*, which are performed in accordance with a *protocol function* representing their decision-making process. The assignment of local states is given by a local *evolution function* which determines the next local state based on the current local state and the joint actions performed by all agents in the system at a given time step.

[9]User manual is accessible via `https://vas.doc.ic.ac.uk/software/mcmas/manual.pdf`

```
1  InitStates
2  ((m0.u0=true and m0.d0=false) or (m0.u0=false and m0.d0=true)) and
3  ((m1.u1=true and m1.d1=false) or (m1.u1=false and m1.d1=true));
4  end InitStates
5
6  Formulae
7  #PR <<x>><<y>>[[z]](m0,x)(m1,y)(Environment,z)(G(F(d0 and u1)) and G(F(d1
       and u0)));
8  end Formulae
```

Figure 2.16: Initial states and a formula to be verified

Example 4. This example is taken from [Toumi et al., 2015]. Consider a peer-to-peer network with two agents. At each time step, each agent can only either try to download or to upload. In order to download successfully, an agent must download while the other uploads at the same time. Both agents want to download infinitely often.

Figure 2.14 shows an agent for Example 4 described using ISPL and Figure 2.17[10] shows the structure of the system. Agent `m0` has two local variables: `u0` and `d0`, which correspond to *download* and *upload* respectively. There are 4 possible actions for `m0` to manipulate the values of its local variables according to the evolution function: action `val00` sets `u0` and `d0` to false, action `val01` sets `u0` to false and `d0` to true, action `val10` sets `u0` to true and `d0` to false, and action `val01` sets `u0` and `d0` to true. It is stated that at each time step, each agent can only either download or upload. This is reflected in the protocol function, where in every possible local state, there are only two actions available: `val01` and `val10`. The goal of agent `m0` (resp. `m1`) is to download infinitely often, and may be expressed with the `LTL` formula $\mathbf{GF}(\texttt{d0} \wedge \texttt{u1})$ (resp. $\mathbf{GF}(\texttt{d1} \wedge \texttt{u0})$).

An ISPL model also contains an `Evaluation` section following the agents' declarations (see Figure 2.15). In this section, the *atomic variables* that are used in the formulae to be verified in the model are declared. Figure 2.15 shows the definition of 4 atomic variables: `u0, d0, u1, d1`. The formulae to be verified (see `Formulae` section in Figure 2.16) are built over the atomic propositions defined here.

The description of the system is completed by declaring a set of initial states and the set of formulae to be verified. As shown in Figure 2.16, the set of initial states is defined in the section `InitStates` by means of a Boolean function determining values of local variables. There is also a `Fairness` section which corresponds to

[10]For the sake of presentation, the actions are ommitted from the figure. However, it should be clear which action leads to which state.

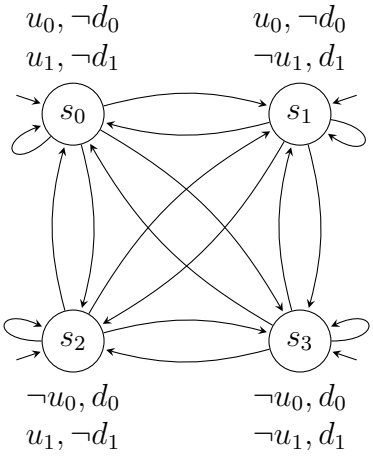

Figure 2.17: The structure of Example 4.

Büchi fairness constraints, a `Groups` section, and semantics of *evolution function*[11]. The formula in Figure 2.16 is in the SL[1G] fragment of SL [Mogavero et al., 2012] which is implemented in MCMAS with perfect recall strategies. The formula says that there exist strategies for agent `m0` and `m1` that satisfy their goals. Indeed, the formula is true since there exist strategies for both players that satisfy their goals. The strategies are those that end with oscillating move between s_1 and s_2, or in other words, strategies that end with $(s_1 s_2)^\omega$.

2.6.4 PRALINE

PRALINE [Brenguier, 2013] is a tool to compute Nash equilibria in concurrent games played over graphs. PRALINE finds pure Nash equilibria in games where the preferences of the players are given by (payoff) functions that map the vertices of the graph to integers. The goal of a player is to maximise the limit superior of her payoff. Although the goals in PRALINE can be seen as a generalisation of Büchi goals, PRALINE does not support full LTL goals. Thus, since games in PRALINE are deterministic, the goals are strictly less expressive than LTL goals.

Example 5. This example is taken from [Brenguier, 2013]. A set of users share access to a wireless channel. In each timeslot, users can either transmit or wait for the next timeslot. Too many users choosing to emit in the same timeslot will result in them failing to send data. Moreover, transmitting costs energy to the players. Players seek to maximise the number of succesful data transmitions using the available energy.

[11]Detailed documentation is available from [MCMAS Web, 2019].

```
move {
legal p1 0;
legal p2 0;
if (energy1 > 0) legal p1 1;
if (energy2 > 0) legal p2 1;
}

update {
if (action p1 == 1)
energy1 = energy1 -1;
if (action p2 == 1)
energy2 = energy2 -1;

if (action p1 == 1 && action p2 == 0)
trans1 = trans1 + 1;
if (action p1 == 0 && action p2 == 1)
trans2 = trans2 + 1;
}
```

Figure 2.18: Part of the code to model Example 5

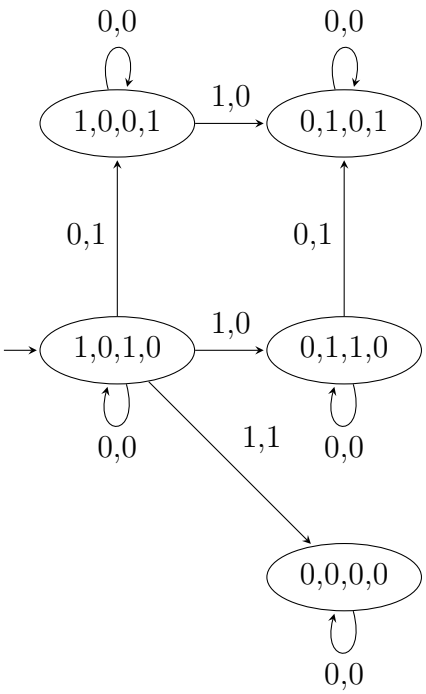

Figure 2.19: The structure of Example 5.

Figure 2.18 is one possible way to model the game in PRALINE. Players are `p1` and `p2`, energy levels are encoded in `energy1` and `energy2`, and number of successful attempts in `trans1` and `trans2`. Players can always wait (action 0), and if their energy is not equal to zero, then they can transmit (action 1). The arena representing an initial energy level allowing only one attempt for each player is shown in Figure 2.19. The labels of the nodes correspond to the valuation of the variables `energy1`, `trans1`, `energy2`, and `trans2`.

2.7 Rational Verification

Many solution concepts have been proposed in the game theory literature [Maschler et al., 2013] and among those concepts, Nash equilibrium is the most important and widely used analytical tool in the game theory study [Osborne and Rubinstein, 1994]. A Nash equilibrium is an outcome where no player can benefit by changing strategies unilaterally, that is, changing strategies while other players keep theirs unchanged. Since we are assuming that players are acting rationally in pursuit of their goals, Nash equilibrium is an ideal tool for analysing systems. The following sections present the *Simple Reactive Modules Language* (SRML) [van der Hoek et al., 2005], *Reactive Modules Games* (RMGs) [Gutierrez et al., 2013], the concept of *equilibrium checking* which gives rise to a general paradigm of *rational verification* [Wooldridge et al., 2016, Gutierrez et al., 2017b], and a prototype tool for equilibrium checking.

2.7.1 SRML

SRML is a strict subset of the Reactive Modules Language (RML) [Alur and Henzinger, 1999], a modelling language that is used by MOCHA, a model checking tool [Alur et al., 1998b, Alur et al., 2001], as well as influenced formalisms used by numerous other tools (including some of the tools presented in Section 2.6.) SRML defines agents/players as *modules*. A module in SRML consists of:

- an *interface*, which defines the name of the module and lists the Boolean variables controlled by the module.
- a number of *guarded commands*, which defines the choices available to the module at each state.

There are two kinds of guarded commands: **init** and **update**. **init** guarded commands are used for *initialising* the Boolean variables controlled by the module. **update**

guarded commands are used for *updating* the value of those Boolean variables during the run of the system. A guarded command has two parts: a "condition" part (the "guard") and an "action" part. The "guard" determines whether a guarded command can be executed or not given the current state, while the "action" part defines how to update the value of (some of) the variables controlled by a corresponding module. Intuitively, $\varphi \rightsquigarrow \alpha$ can be read as "if condition φ is satisfied, then *one* of the choices available to the module is to execute α". Note that the value of φ being true does not guarantee the execution of α, but it is *enabled* for execution thus *may be chosen*. If no guarded commands of a module are enabled in some state, then that module has no choice and the values of the variables controlled by it are assumed to remain unchanged in the next state.

Formally, a guarded command g over a set of Boolean variables Φ is an expression

$$g: \quad \varphi \rightsquigarrow x_1' := \psi_1; \ldots; x_k' := \psi_k$$

where the guard φ is a propositional logic formula over Φ, each x_i is a member of Φ and ψ_i is a propositional logic formula over Φ. Let $guard(g)$ denote the guard of g, thus, in the above rule, we have $guard(g) = \varphi$. It is required that no variable x_i appears on the left hand side of more than one assignment statements in the same guarded command, hence no issue on the (potentially) conflicting updates arises. The variables $x_1, \ldots, x_k$ are said to be *controlled variables* of g, and the set of controlled variables are denoted by $ctr(g)$. If no guarded command of a module is enabled, then the values of all variables in $ctr(g)$ remain unchanged. A set of guarded commands is said to be *disjoint* if their controlled variables are mutually disjoint.

To make it clearer, the following is an example of a guarded command:

$$\underbrace{(p \wedge q)}_{\text{guard}} \rightsquigarrow \underbrace{p' := true; q' := false}_{\text{action}}$$

The guard is the propositional logic formula $(p \wedge q)$, so this guarded command will be enabled if both p and q are true. If the guarded command is chosen (to be executed), then in the next time-step, variable p will be assigned to *true* and q to *false*.

Formally, an SRML module m_i is defined as a triple $m_i = (\Phi_i, I_i, U_i)$, where:

- $\Phi_i \subseteq \Phi$ is the finite set of Boolean variables controlled by m_i;
- I_i is a finite set of **init** guarded commands, such that for all $g \in I_i$, we have $ctr(g) \subseteq \Phi_i$;

- U_i is a finite set of **update** guarded commands, such that for all $g \in U_i$, we have $ctr(g) \subseteq \Phi_i$.

Definition 17 (SRML arena)**.** An SRML *arena* A is formally defined to be an $(n+2)$-tuple $A = (\mathrm{N}, \Phi, m_1, \ldots, m_n)$, where $\mathrm{N} = \{1, \ldots, n\}$ is a set of agents, Φ is a set of Boolean variables, and for each $i \in \mathrm{N}, m_i = (\Phi_i, I_i, U_i)$ is an SRML module over Φ that defines the choices available to agent i. It is required that $\{\Phi_1, \ldots, \Phi_n\}$ forms a partition of Φ, hence every variable in Φ is controlled by some agent, and no variable is controlled by more than one agent.

For every module $m_i = (\Phi_i, I_i, U_i)$, an additional guarded command $g_i^{\mathbf{skip}}$ is introduced. This command is given by:

$$g_i^{\mathbf{skip}} = \bigwedge_{g \in U_i} \neg guard(g) \rightsquigarrow \mathbf{skip}$$

This asserts that by executing $g_i^{\mathbf{skip}}$ will make the values of all variables in Φ_i to remain unchanged. Define the set of update guarded commands that are available to be executed as $enabled_i(v)$, where v is a valuation of variables in Φ that is visible to m_i. If there is no update guarded commands available, then $g_i^{\mathbf{skip}}$ is enabled. Formally,

$$enabled_i(v) = \Big\{ g \in U_i \cup \{g_i^{\mathbf{skip}}\} : v \models guard(g) \Big\}.$$

This definition means that $enabled_i(v)$ is never empty.

Let $g : \varphi \rightsquigarrow x_i' := \psi_i, \ldots, x_k' := \psi_k$ be a guarded command in module m_i that controls the variables in Φ_i. Define $exec_i(g, v)$ as a propositional valuation for the variables Φ_i as the result of executing g on v. It only specifies a valuation for the variables in Φ_i and does not give the value of variables $\Phi \backslash \Phi_i$. Formally,

$$exec(g, v) = \Big(v_i \backslash ctr(g) \Big) \cup \Big\{ x_i \in \{x_1, \ldots, x_k\} : v \models \psi_i \Big\}.$$

The progression of an SRML arena is defined by the execution of enabled guarded commands, one for each module, in a synchronous and concurrent fashion. A *joint guarded command* $J = (g_1, \ldots, g_n)$ is a profile of guarded commands, one for each module. Write

$$enabled(v) = enabled_1(v) \times \cdots \times enabled_n(v)$$

as the extension of guarded commands to joint guarded commands. Also write

$$exec(J, v) = exec_1(g_1, v) \cup \cdots \cup exec_n(g_n, v)$$

to denote the execution of guarded command $J = (g_1, \ldots, g_n)$. Moreover, write $exec_i(g_i, v_\perp)$ to indicate the initialisation of m_i and extends it to $exec(J, v_\perp)$, where $J \in I_1 \times \cdots \times I_n$ to indicate joint initialisation.

2.7.2 LTL Reactive Modules Game

LTL Reactive Modules Games (RMGs) [Gutierrez et al., 2017b] have two components: an *arena* and *goals*. The goals specify the preferences of players: every player i is associated with a goal γ_i, where γ_i is an LTL formula. Formaly, a game is defined as:

$$\mathcal{G} = (A, \gamma_1, \ldots, \gamma_n)$$

where $A = (\mathrm{N}, \Phi, m_1, \ldots, m_n)$ is an arena with player set $\mathrm{N} = \{1, \ldots, n\}$, Boolean variable set Φ, and m_i an SRML module describing the choices available to each player $i \in \mathrm{N}$. Furthermore, for each player $i \in \mathrm{N}$, the LTL formula γ_i represents the goal that i wants to satisfy. Games are played by each player i selecting a *deterministic* strategy σ_i that defines the player choices throughout the game. We write Φ_{-i} for $\Phi \backslash \Phi_i$ and let V_i (resp. V_{-i}) denote the set of valuations to variables in Φ_i (resp. Φ_{-i}). Given an arena $A = (\mathrm{N}, \Phi, m_1, \ldots, m_n)$, a *deterministic strategy* for module $m_i = (\Phi_i, I_i, U_i)$ is $\sigma_i = (Q_i, q_i^0, \delta_i, \tau_i)$, where Q is a finite and non-empty set of states, $q_i^0 \in Q_i$ is the initial state, $\delta_i : Q_i \times V_{-i} \to Q_i \setminus \{\emptyset\}$ is a *transition function*, and $\tau_i : Q_i \to V_i$ is an *output function*.

This model of strategies is, technically, given by deterministic Moore machines. For a machine σ_i representing a strategy for player i, the input language corresponds to the actions of other players, while the outputs are player i's actions implementing the strategy. There are several advantages of representing strategies as finite state machines. Firstly, the representation scheme is finite. Whereas abstract representations using functions may have an infinitely large domains, which makes the study of some problems difficult. Secondly, finite state machine representations are enough for players whose goals are expressed in temporal logic formulae [Gutierrez et al., 2015b]. Finally, the use of finite state machine strategies is in fact standard practice in the literature on iterated games [Binmore, 1992]. For a more detailed discussion on this topic, see [Gutierrez et al., 2015b].

In this thesis, all strategies are assumed to be *consistent*, that is, all strategies comply with the module's specification[12]. Formally, a strategy σ_i is consistent with m_i if the following two conditions are satisfied:

1. for the initial state q_i^0, we have $\tau_i(q_i^0) = exec(g, v_\perp)$ for some $g \in I_1 \times \cdots \times I_n$,
2. for all $q, q' \in Q_i$ and $v = v_i \cup v_{-i} \in V$ such that $\delta(q, v_{-i}) = q'$ and $v_i = \tau_i(q)$, we have $\tau(q') = exec(g_i, v)$ for some $g_i \in enabled_i(v)$.

[12]This is related to *protocol-compliant* in MCMAS system.

That is, the valuation prescribed by the strategy σ_i of i at every state $q \in Q_i \cup \{q_i^0\}$ can be realised by m_i. Hereafter, let Σ_i be the set of consistent strategies for m_i and $\sigma_i \in \Sigma_i$.

Once every player i has selected a strategy σ_i, we have a *strategy profile* $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ and an *outcome* of the game $\boldsymbol{\rho}(\vec{\sigma})$. A strategy profile $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ is *consistent with* A if each σ_i is consistent with m_i[13]. The outcome determines whether each player's goal is satisfied or not. A goal γ_i is satisfied by an outcome $\boldsymbol{\rho}(\vec{\sigma})$ if and only if $\boldsymbol{\rho}(\vec{\sigma}) \models \gamma_i$. In order to simplify notations, we write $\vec{\sigma} \models \gamma_i$ for $\boldsymbol{\rho}(\vec{\sigma}) \models \gamma_i$.

A *preference relation* $\succeq_i$ is defined over outcomes for each player i with goal γ_i for strategy profiles $\vec{\sigma}$ and $\vec{\sigma}'$ by saying that

$$\vec{\sigma} \succeq_i \vec{\sigma}' \quad \text{if and only if} \quad \vec{\sigma}' \models \gamma_i \quad \text{implies} \quad \vec{\sigma} \models \gamma_i.$$

Now the solution concept of *Nash equilibrium* [Osborne and Rubinstein, 1994] for LTL RMGs can be defined as follows: given a game $\mathcal{G} = (A, \gamma_1, \ldots, \gamma_n)$, a strategy profile $\vec{\sigma}$ is said to be a Nash equilibrium of $\mathcal{G}$if for all players i and all strategies $\vec{\sigma}'$ in the game, we have

$$\vec{\sigma} \succeq_i (\vec{\sigma}_{-i}, \sigma_i')$$

where $(\vec{\sigma}_{-i}, \sigma_i')$ denotes strategy profile $(\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$. Let $\mathrm{NE}(\mathcal{G})$ be the set of (pure strategy) Nash equilibria of game $\mathcal{G}$.

2.7.3 CTL Reactive Modules Games

Similar to LTL RMGs, CTL RMGs have two components: an *arena* and *goals*. In CTL RMGs, the goal of each player is specified with a CTL formula and each player strategy is *non-deterministic*. The definition of non-deterministic strategies is a generalisation of that of determinisitic ones, thus the transition function is redefined as follows: $\delta_i : Q_i \times V_{-i} \to 2^{Q_i} \setminus \{\emptyset\}$. The outcome $[\![\vec{\sigma}]\!]$ of a game with arena $A = (\mathrm{N}, \Phi, m_1, \ldots, m_n)$ is defined as a Kripke structure $K_{A_{\vec{\sigma}}}|_\Phi$ associated with the SRML arena $A_{\vec{\sigma}} = (\mathrm{N}, \Phi \cup \bigcup_{i \in N} Q_i, m_{\sigma_1}, \ldots, m_{\sigma_n})$ restricted to valuations with respect to Φ. The concept of Nash equilibrium with respect to strategy profile $\vec{\sigma}$ of consistent non-deterministic strategies is a straightforward extension of that in LTL RMGs.

[13] In the rest of this thesis, we restrict our attention to consistent strategy profiles.

2.8 Concurrent Multi-Player Games

This section contains some definitions that are going to be used in some of the next chapters. As defined in Section 2.3.2, a *concurrent game structure* (CGS) is a tuple $\mathcal{M} = (\mathrm{N}, (\mathrm{Ac}_i)_{i\in\mathrm{N}}, \mathrm{St}, s_0, \mathsf{tr}, \lambda)$. A multi-player game can be defined on top of a structure $\mathcal{M}$ by associating each player with a goal. In the rest of this thesis, we consider games with LTL goals. The rationale behind the choice of using LTL as the specification language is as follows. Firstly, LTL is arguably more intuitive, since it uses fewer modal operators. Secondly, branching-time logics are usually used to capture some uncertainty in some system, *e.g.*, incomplete information or uncertainty about the history of the run—hence the branching. Thus, since our setting is in perfect recall and complete information, it is reasonable to assume that the strategies are deterministic and, as such, induce some unique run. With this setting, we argue that LTL is the most appropriate language to specify the properties of the system.

Definition 18. A *(concurrent multi-player)* LTL *game* is a tuple $\mathcal{G}_{\mathsf{LTL}} = (\mathcal{M}, (\gamma_i)_{i\in\mathrm{N}})$ where each γ_i is the goal of player i, given as an LTL formula over AP.

To define games with parity goals we will consider priority functions. Let $\alpha : \mathrm{St} \to \mathbb{N}$ be a priority function. We say that a path π satisfies $\alpha : \mathrm{St} \to \mathbb{N}$, and write $\pi \models \alpha$ in such a case, if the minimum number occurring infinitely often in the infinite sequence $\alpha(\pi_0), \alpha(\pi_1), \alpha(\pi_2), \ldots$ is even.

Definition 19. A *parity game* is a two-player zero-sum turn-based game given by a labelled finite graph $H = (V_0, V_1, E, \alpha)$ such that $\mathrm{St} = V_0 \cup V_1$ is a set of states partitioned into Player 0 (V_0) and Player 1 (V_1) states, respectively, $E \subseteq V \times V$ is a set of edges/transitions, and $\alpha : \mathrm{St} \to \mathbb{N}$ is a labelling priority function. Player 0 wins if the smallest priority that occurs infinitely often in the infinite play is even. Otherwise, player 1 wins.

It is known that solving a parity game (checking which player has a winning strategy) is an NP $\cap$ coNP problem [Jurdzinski, 1998], and recently it is shown that it can be solved in quasi-polynomial time [Calude et al., 2017]. Despite more than 30 years of research, and extremely promising practical performance, it is still unknown whether parity games can be solved in polynomial time.

Definition 20. A *concurrent multi-player parity game* is a tuple $\mathcal{G}_{\mathrm{PAR}} = (\mathcal{M}, (\alpha_i)_{i\in\mathrm{N}})$, where each $\alpha_i : \mathrm{St} \to \mathbb{N}$ is the goal of player i, given as a priority function over St.

Hereafter, for statements regarding either LTL or Parity games, we will simply denote the underlying structure as $\mathcal{G}$. Games are played by each player i selecting a *strategy* σ_i that will define how to make choices over time. Formally, for a given game $\mathcal{G}$, a strategy $\sigma_i = (S_i, s_i^0, \delta_i, \tau_i)$ for player i is a finite state machine with output (a transducer), where S_i is a finite and non-empty set of *internal states*, s_i^0 is the *initial state*, $\delta_i : S_i \times \vec{\text{Ac}} \to S_i$ is a deterministic *internal transition function*, and $\tau_i : S_i \to \text{Ac}_i$ an *action function*. Let Σ_i be the set of strategies for player i. A strategy is *memoryless* in $\mathcal{G}$ from s if $S_i = \text{St}$, $s_i^0 = s$, and $\delta_i = \mathsf{tr}$. Once every player i has selected a strategy σ_i, a *strategy profile* $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ results and the game has an *outcome*, a path in $\mathcal{M}$, which we will denote by $\pi(\vec{\sigma})$. Because strategies are deterministic, $\pi(\vec{\sigma})$ is the unique path induced by $\vec{\sigma}$, that is, the infinite sequence $s_0, s_1, s_2, \ldots$ such that

- $s_{k+1} = \mathsf{tr}(s_k, (\tau_1(s_1^k), \cdots, \tau_n(s_n^k)))$, and
- $s_i^{k+1} = \delta_i(s_i^k, (\tau_1(s_1^k), \cdots, \tau_n(s_n^k)))$, for all $k \geq 0$.

Note that the path induced by the strategy profile $\vec{\sigma}(\sigma_1, \ldots, \sigma_n)$ from state s_0 corresponds to the one generated by the finite transducer $\mathsf{T}_{\vec{\sigma}}$ obtained from the composition of the strategies σ_i's in $\vec{\sigma}$, with input set St and output set $\vec{\text{Ac}}$, where the initial input is s_0. Since such transducer is finite, the generated path π is *ultimately periodic*, that is, there exists $p, r \in \mathbb{N}$ such that $\pi_k = \pi_{k+r}$ for every $p \leq k$. This means that, after the prefix $\pi_{\leq p}$, the path loops indefinitely over the sequence $\pi_{p+1} \ldots \pi_{p+r}$.

Definition 21. A *bisimulation*, denoted by $\sim$, between states $s^* \in \text{St}$ and $t^* \in \text{St}'$ is a non-empty binary relation $R \subseteq \text{St} \times \text{St}'$, such that $s^* \; R \; t^*$ and for all $s, s' \in \text{St}$, $t, t' \in \text{St}'$, and $\vec{a} \in \vec{\text{Ac}}$:

- $s \; R \; t$ implies $\lambda(s) = \lambda'(t)$,
- $s \; R \; t$ and $\mathsf{tr}(s, \vec{a}) = s'$ implies $\mathsf{tr}(t, \vec{a}) = t''$ for some $t'' \in \text{St}'$ with $s' \; R \; t''$,
- $s \; R \; t$ and $\mathsf{tr}(t, \vec{a}) = t'$ implies $\mathsf{tr}(s, \vec{a}) = s''$ for some $s'' \in \text{St}$ with $s'' \; R \; t'$.

Then, if there is a bisimulation between two states s^* and t^*, we say that they are *bisimilar* and write $s^* \sim t^*$ in such a case. We also say that CGSs $\mathcal{M}$ and $\mathcal{M}'$ are *bisimilar* (in symbols $\mathcal{M} \sim \mathcal{M}'$) if $s_0 \sim s_0'$. Bisimilar structures satisfy the same set of temporal logic properties, a desirable property that will be relevant later.

For parity games, we can define a pereference relation $\succeq_i$ over outcomes for each player i analogously as previously defined. For two strategy profiles $\vec{\sigma}$ and $\vec{\sigma}'$ in $\mathcal{G}$, we have

$$\pi(\vec{\sigma}) \succeq_i \pi(\vec{\sigma}') \text{ if and only if } \pi(\vec{\sigma}') \models \alpha_i \text{ implies } \pi(\vec{\sigma}) \models \alpha_i.$$

On this basis, we can define the concept of Nash equilibrium [Osborne and Rubinstein, 1994] for a multi-player game with parity goals: given a game $\mathcal{G}$, a strategy profile $\vec{\sigma}$ is a *Nash equilibrium* of $\mathcal{G}$ if, for every player i and strategy $\sigma_i' \in \Sigma_i$, we have

$$\pi(\vec{\sigma}) \succeq_i \pi((\vec{\sigma}_{-i}, \sigma_i'))$$

where $(\vec{\sigma}_{-i}, \sigma_i')$ denotes $(\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$, the strategy profile where the strategy of player i in $\vec{\sigma}$ is replaced by σ_i'. Let $\mathrm{NE}(\mathcal{G})$ denote the set of Nash equilibria of $\mathcal{G}$. In [Gutierrez et al., 2017a] it is shown that, using the model of strategies defined above, the existence of Nash equilibria is preserved across bisimilar systems. This is in contrast to other models of strategies considered in the concurrent games literature, which do not preserve Nash equilibria. Because of this, hereafter, we say that $\{\Sigma_i\}_{i \in \mathrm{N}}$ is a set of *bisimulation-invariant strategies* and that $\mathrm{NE}(\mathcal{G})$ is the set of bisimulation-invariant Nash equilibrium profiles of $\mathcal{G}$.

Definition 22. A *deterministic automaton on infinite words* is a tuple

$$\mathcal{A} = (\mathrm{AP}, Q, q^0, \rho, \mathcal{F})$$

where Q is a finite set of states, $\rho : Q \times \mathrm{AP} \to Q$ is a transition function, q^0 is an initial state, and $\mathcal{F}$ is an acceptance condition. We mainly use *parity* and *Streett* acceptance conditions. A parity condition $\mathcal{F}$ is a partition $\{F_1, \ldots, F_n\}$ of Q, where n is the *index* of the parity condition and any $[1, n] \ni k$ is a *priority*. We use a *priority function* $\alpha : Q \to \mathbb{N}$ that maps states to priorities such that $\alpha(q) = k$ if and only if $q \in F_k$. For a path $\pi = q^0, q^1, q^2 \ldots$, let $\mathrm{Inf}(\pi)$ denote the set of states occurring infinitely often in the path:

$$\mathrm{Inf}(\pi) = \{q \in Q \mid q = q^i \text{ for infinitely many } i\text{'s}\}$$

A path π is accepted by a deterministic parity word (DPW) automaton with condition $\mathcal{F}$ if the minimum priority that occurs infinitely often is even, i.e., if the following condition is satisfied:

$$\left(\min_{k \in [1,n]} (\mathrm{Inf}(\pi) \cap F_k \neq \varnothing) \right) \bmod 2 = 0.$$

A Streett condition $\mathcal{F}$ is a set of pairs $\{(E_1, C_1), \ldots, (E_n, C_n)\}$ where $E_k \subseteq Q$ and $C_k \subseteq Q$ for all $k \in [1, n]$. A path π is accepted by a deterministic Streett word (DSW) automaton $\mathcal{S}$ with condition $\mathcal{F}$ if π either visits E_k finitely many times or visits C_k infinitely often, i.e., if for every k either $\mathrm{Inf}(\pi) \cap E_k = \varnothing$ or $\mathrm{Inf}(\pi) \cap C_k \neq \varnothing$.

2.8.1 Equilibrium Checking

Wooldridge et al. [Wooldridge et al., 2016] proposes the problem of *Equilibrium Checking*, which basically asks whether a given temporal formula φ is satisfied on some run corresponding to a Nash equilibrium of the system. In this setting, the agents are seen as *non-deterministic reactive* programs. Non-determinism means that agents can (freely) choose any available actions, while reactivity captures the idea that agents are non-terminating.

The notion of equilibrium checking can be regarded as a counterpart to model checking and classical verification with a more "restricted" condition where we are given a (multi-agent) system $\mathcal{G}$ and a temporal logic formula φ representing a desirable property, and the question is whether φ could be satisfied *on some run that would arise from a Nash equilibrium collection of choices by players within the system.* This idea can be captured in the following decision problem:

E-Nash
Given : Multi-agent system $\mathcal{G}$; temporal formula φ.
Question : Is there any $\vec{\sigma} \in \mathrm{NE}(\mathcal{G})$ such that $\boldsymbol{\rho}(\vec{\sigma}) \models \varphi$?

We can also ask the obvious counterpart of E-Nash:

A-Nash
Given : Multi-agent system $\mathcal{G}$; temporal formula φ.
Question : Is it the case that for all $\vec{\sigma} \in \mathrm{NE}(\mathcal{G})$, we have $\boldsymbol{\rho}(\vec{\sigma}) \models \varphi$?

Other decision problems are: "verifying whether a system has any Nash equilibria" (Non-Emptiness) and "checking whether a given strategy profile represents a Nash equilibrium" (Membership):

Non-Emptiness
Given : Multi-agent system $\mathcal{G}$.
Question : Is it the case that $\mathrm{NE}(\mathcal{G}) \neq \varnothing$?

Membership
Given : Multi-agent system $\mathcal{G}$; strategy profile $\vec{\sigma}$.
Question : Is it the case that $\vec{\sigma} \in \mathrm{NE}(\mathcal{G})$?

Gutierrez, Harrenstein, and Wooldridge [Gutierrez et al., 2013] use Nash equilibrium to analyse a game-like concurrent system called Iterated Boolean Games (iBG) and investigate which computations can be generated in equilibrium. However, [Gutierrez et al., 2013] does not provide a language to reason about (Nash) equilibria of the system, hence the reasoning cannot be done explicitly. Gutierrez et al. in [Gutierrez et al., 2014] provide a language called *Equilibrium Logic* (EL) to address this issue and introduce a more appropriate computational model to represent concurrent programs. EL extends branching time logic with a new path quantifier [NE]φ expressing that φ holds *on all Nash equilibrium computations of the system.* This allows us to reason about equilibria of the system directly in the object language without having to carry it out at the meta-level.

2.8.2 A Prototype Equilibrium Checking Tool

The list of tools mentioned in Section 2.6 is obviously not exhaustive. However, those tools are actively being developed and arguably some of the most popular ones. In respect of the proposed notion of equilibrium checking (Section 2.8.1), the mentioned tools do not have any explicit support for it except PRALINE which allows us to find pure strategy Nash equilibrium in some generalised Büchi objective games. The other most probable candidate is MCMAS since it uses SL, which is able to express the existence of Nash equilibria in a concurrent multi-agent game. Thus in principle, it is possible to analyse some equilibrium properties of MCMAS systems. Indeed, later in Chapter 3 a method to perform equilibrium checking/rational verification using MCMAS will be discussed.

Apart from those mentioned in Section 2.6, one of the more recent multi-agent systems verification tools is EAGLE (Equilibrium Analyser for Game Like Environments) [Toumi et al., 2015]. EAGLE is a prototype tool for equilibrium checking which, particularly, solves the MEMBERSHIP problem. Thus, it needs two given inputs: a multi-agent system represented as a CTL RMG $\mathcal{G}$ and a strategy profile $\vec{\sigma}$. $\mathcal{G}$ is represented as a set of agents, defined using SRML [van der Hoek et al., 2005], and agents goals are specified in CTL. A strategy profile $\vec{\sigma}$ is represented as a collection of non-deterministic strategies for the agents encoded using SRML.

The basic algorithm of EAGLE is shown in Algorithm 6, where $\mathcal{G} = (A, \gamma_1, \ldots, \gamma_n)$, $A = (\mathrm{N}, \Phi, m_1, \ldots, m_n)$, and γ_i is the goal of m_i. It uses a CTL variant of algorithms first introduced in [Gutierrez et al., 2013] which relies on the existence of two oracles, one for model checking and one for satisfiability of temporal logic formulae. EAGLE uses open source external libraries for CTL satisfiability (CTL SAT [Prezza,

Algorithm 6 EAGLE Basic Algorithm

1: **function** EQCHECK($\mathcal{G}, \vec{\sigma}$)
2: $K_{A_{\vec{\sigma}}}|_\Phi$ = ARENA2KRIPKE($A_{\vec{\sigma}} = (\mathrm{N}, \Phi \cup \bigcup_{i \in N} Q_i, m_{\sigma_1}, \ldots, m_{\sigma_n})$)
3: **for each** $i \in \mathrm{N}$ **do**
4: **if** $K_{A_{\vec{\sigma}}}|_\Phi \not\models \gamma_i$ **then** (*calling* MR.WAFFLES *model checker*)
5: **if** $Sat(\mathrm{TH}_{CTL}(A) \wedge \gamma_i)$ **then** (*calling* CTL SAT)
6: **return** "no"
7: **end if**
8: **end if**
9: **end for**
10: **return** "yes"
11: **end function**
12:
13: **function** ARENA2KRIPKE($A_{\vec{\sigma}} = (\mathrm{N}, \Phi \cup \bigcup_{i \in N} Q_i, m_{\sigma_1}, \ldots, m_{\sigma_n})$)
14: $S_A := \varnothing$, $S_A^0 := \varnothing$, $R_A := \varnothing$, $\pi_A := \varnothing$
15: $C_1 := I_1; \ldots; C_n := I_n$;
16: **for each** $J \in C_1 \times \cdots \times C_n$ **do**
17: $S_A := S_A \cup \{exec(J, v_\perp)\}$
18: $S_A^0 := S_A$
19: $\pi_A := \pi_A \cup \{(exec(J, v_\perp), (exec(J, v_\perp))\}$
20: **end for**
21: $X := \varnothing$
22: **while** $X \neq S_A$ **do**
23: $X := S_A$
24: **for each** $v \in S_A$ **do**
25: $C_1 := enabled_i(\pi_A(v)); \ldots; C_n := enabled_n(\pi_A(v))$
26: **for each** $J \in C_1 \times \cdots \times C_n$ **do**
27: $S_A := S_A \cup \{exec(J, \pi_A(v))\}$
28: $\pi_A := \pi_A \cup \{(exec(J, \pi_A(v)), exec(J, \pi_A(v)))\}$
29: **end for**
30: **end for**
31: **end while**
32: **for each** $(v, v') \in S_A \times S_A$ **do**
33: $C_1 := enabled_i(\pi_A(v)); \ldots; C_n := enabled_n(\pi_A(v))$
34: **for each** $J \in C_1 \times \cdots \times C_n$ **do**
35: **if** $exec(J, \pi_A(v)) = \pi_A(v')$ **then**
36: $R_A := R_A \cup \{(v, v')\}$
37: **end if**
38: **end for**
39: **end for**
40: **return** $K_A = (S_A, S_A^0, R_A, \pi_A)$
41: **end function**

2016]) and CTL model checking (Mr.Waffles [Reynaud, 2016]). Key to EAGLE's performance is the construction of a formula $\textsc{Th}_{CTL}(A)$ using the construction in the proof for Lemma 1 (adapted from [Gutierrez et al., 2017b]).

Lemma 1. For every arena A of size $|A|$, there is a CTL formula $\textsc{Th}_{CTL}(A)$ of size polynomial in $|A|$ such that for all $\boldsymbol{\rho} : \mathbb{N} \to 2^{\Phi}$, $\boldsymbol{\rho} \in \mathbf{runs}(A)$ iff $\boldsymbol{\rho} \models \textsc{Th}_{CTL}(A)$.

Proof. Given an arena $A = (\mathrm{N}, \Phi, m_1, \ldots, m_n)$, we define the formula $\textsc{Th}_{CTL}(A) = \textsc{Init}(A) \wedge \textsc{Update}(A)$. We then define $\textsc{Unch}(\Psi)$ as variables that take the same value in the next state, formally:

$$\textsc{Unch}(\Psi) = \bigwedge_{x \in \Psi} (x \leftrightarrow \mathsf{AX}x).$$

We define the effect of a single initialisation guarded command as follows:

$$\textsc{Init}_i(g = \top \rightsquigarrow x_1' := b_1; \ldots; x_k' := b_k) = \big(\bigwedge_{l=1}^{k} x_l \leftrightarrow b_l\big) \vee \big(\bigwedge_{x \in \Phi_i \setminus ctr(g)} x \leftrightarrow \bot\big).$$

The expression $\textsc{Init}_i$ then captures the semantics of initialisation commands (we write $\oplus$ to denote "exactly one" operator):

$$\textsc{Init}_i = \bigoplus_{g \in I_i} \textsc{Init}_i(g) = \bigvee_{g \in I_i} \big(\textsc{Init}_i(g) \wedge \bigwedge_{g' \in I_i \setminus \{g\}} \neg \textsc{Init}_i(g')\big)$$

then we have:

$$\textsc{Init}(A) = \bigwedge_{i \in N} \textsc{Init}_i.$$

Next we define the semantics of update rules. Define the effect of a single update guarded command as follows:

$$\textsc{Update}_i(g = \varphi \rightsquigarrow x_1' := \psi_1; \ldots; x_k' := \psi_k) = \varphi \wedge \Big(\bigwedge_{l=1}^{k} \psi_l \leftrightarrow \mathsf{AX}x_l\Big) \wedge \textsc{Unch}(\Phi_i \setminus ctr(g)).$$

Then we define the overall effect of i's update commands:

$$\textsc{Update}_i = \Big(\bigwedge_{g \in U_i} \neg guard(g) \wedge \textsc{Unch}(\Phi_i)\Big) \vee \bigoplus_{g \in U_i} \textsc{Update}_i(g).$$

Finally:

$$\textsc{Update}(A) = \mathsf{AG} \bigwedge_{i \in N} \textsc{Update}_i.$$

□

Current implementation of EAGLE is roughly a prototype and, it is still limited in scope and not optimised. Currently, it only supports CTL and perfect information. It also does not have any graphical user interface.

Chapter 3

Rational Verification with MCMAS

This chapter presents an approach to rational verification with MCMAS [MCMAS Web, 2019, Cermák et al., 2014]. MCMAS allows automatic verification of specifications that use epistemic, temporal, and cooperation modalities. The modalities can be used to define properties of various systems such as communication protocols, security protocols, games, and other properties that may be difficult to represent using temporal operators only. MCMAS adopts *interpreted systems* [Fagin et al., 1995] as the formal semantics to represent systems comprised of multiple entities, and uses a dedicated programming language called *Interpreted Systems Programming Language* (ISPL) to describe the systems. Later in Section 3.3, a method to perform rational verification on LTL RMGs, as well as a tool that implements the method, will be presented.

3.1 Interpreted Systems

Interpreted systems [Halpern and Moses, 1990, Fagin et al., 1995] is a type of computationally grounded semantics that allows one to model temporal and *epistemic properties* of multi-agent systems and distributed computer systems. Intuitively, it describes a system by defining the states in which each agent and the environment can find itself. There are two kinds of state: *local* and *global* states. Local states model the configuration of the agent in the system at each time-step. Given an agent $i \in \mathrm{N}$, L_i is a set of local states for agent i. Refer E as the environment agent and denote L_E as the set of local states for E. The set $\mathscr{G}$ of *global states* of the system is given by

$$\mathscr{G} = L_1 \times \cdots \times L_n \times L_E.$$

Definition 23 (Global state)**.** A *global state* $_{\mathscr{g}} \in \boldsymbol{\mathscr{G}}$ is defined as tuple $_{\mathscr{g}} = (l_1, \ldots, l_n, l_E)$ that defines the configuration of the whole system in an instant of time. In that particular global state, agent 1 is in local state $l_1 \in L_1$, agent 2 is in local state $l_2 \in L_2$, ..., agent n is in local state $l_n \in L_n$, and the environment is in $l_E \in L_E$.

Definition 24 (Agent)**.** An agent $i \in \mathrm{N}$ is a tuple $i = (L_i, Act_i, Pr_i, t_i)$ where:

- L_i is the set of *local states*;
- Act_i is the set of *individual actions*;
- $Pr_i : L_i \to (2^{Act} \setminus \{\emptyset\})$ is the *protocol function*;
- $t_i : L_i \times ACT \to L_i$ is the *local transition function*, where $ACT = Act_1 \times \cdots \times Act_n \times Act_E$ is the set of *joint actions*, such that for $l_i \in L_i$ and $a_i \in Act_i$, $t_i(l_i, a_i)$ is defined iff $a_i \in Pr_i(l_i)$.

The intuitive explanation is an agent i is in local state $l_i \in L_i$, which represents the information it knows about the system, and it can perform action $a_i \in Act_i$ based on protocol function Pr_i. A joint action makes an overall change in the state of the system according to transition function t_i.

Definition 25 (Interpreted system)**.** An *interpreted system* is a tuple $M = (\mathrm{N}, \mathcal{I}, \Pi)$ where:

- every $i \in \mathrm{N}$ is an agent;
- $\mathcal{I} \subseteq \boldsymbol{\mathscr{G}}$ is the set of *global initial states*;
- $\Pi : \boldsymbol{\mathscr{G}} \to 2^{AP}$ is the *labelling function*.

Naturally, interpreted systems induce Kripke structures which can be used to interpret our specification language. The *induced structures* of IS $M = (\mathrm{N}, \mathcal{I}, \Pi)$ are defined as a tuple $K_M = (\boldsymbol{\mathscr{G}}, \mathcal{I}, T, \Pi)$ where:

- $\boldsymbol{\mathscr{G}} = L_1 \times \cdots \times L_n \times L_E$ is the set of global states reachable from initial states in $\mathcal{I}$ via T;
- $\mathcal{I}$ is the set of global initial states;
- $T = \boldsymbol{\mathscr{G}} \times ACT \times \boldsymbol{\mathscr{G}}$ is a transition relation representing the temporal evolution of the system, where $ACT = Act_1 \times \cdots \times Act_n \times Act_E$ is the set of *joint actions*, such that for $l_i \in L_i$ and $a_i \in Act_i$, $t_i(l_i, a_i)$ is defined if, and only if, $a_i \in Pr_i(l_i)$;
- $\Pi : \boldsymbol{\mathscr{G}} \to 2^{AP}$ is the *labelling function*. We write $\Pi|_l$ to denote the restriction of Π to $l \in_{\mathscr{g}}$, where $_{\mathscr{g}} \in \boldsymbol{\mathscr{G}}$.

3.2 Interpreted System Programming Language

The *Interpreted Systems Programming Language* (ISPL)[1] specifies a multi-agent system as an *Environment* agent and a set of *normal* agents. The Environment and normal agents have two sets of variables: *local* and *observable*. Local variables are private, while observable variables are visible to all agents in the system. Every agent also has a set of *actions*, a *protocol function*, and an *evolution function*.

ISPL also specifies the definition of *initial states*, *propositions*, *groups*, *fairness formulae*, and *formulae to be checked*. The following is the general structure of ISPL:

1. *Agents' declarations*. This section are used to define an agent using a sequence of declarations. The syntax is as follows:

```
1    Agent <agentID>
2    <agent_body>
3    end Agent
```

   where `<agentID>` is a valid agent name (not a reserved keyword for normal agent and `Environment` for the Environment agent), and `<agent_body>` contains the declaration of variables, actions, protocol, and evolution function for each agent.

   Variables. There are two types of variables: local and observable. Local variables can be defined with the following syntax:

```
1    Vars:
2    <var_name> : <var_type>
3    end Vars
```

   Observable variables are of the two types: local and global. Local observable variables in normal agents can be defined as follows:

```
1    Lobvars = {var_0,...,var_n}
```

[1]Documentation and user manual is available from `http://vas.doc.ic.ac.uk/software/mcmas/`

while (global) observable variables in the Environment agent can be seen by all agents in the interpreted system and defined as follows:

```
1    Obsvars:
2    <var_name> : <var_type>
3    end Obsvars
```

Actions. All actions of an agent can be defined as follows:

```
1    Actions = {action_0,...,action_n}
```

Protocol. A line in a protocol function is composed of a condition, which is a Boolean formula over local variables, and a list of actions. If the condition is satisfied, the actions in the list are allowed to be performed. The syntax is as follows:

```
1    Protocol:
2    <condition> : <list_of_actions>
3    end Protocol
```

To make it clearer, consider the following example:

```
1    Protocol:
2    (p=true and q=false) : {ac0,ac1};
3    end Protocol
```

action `ac0` and `ac1` only available (to be chosen and executed) at states in which p is true and q is false.

Evolution function. A line in an evolution function consists of a condition and a set of assignments of local variables. The syntax is as follows:

```
1    Evolution:
2    <assignment> if <condition>
3    end Evolution
```

To make it clearer, consider the following example:

```
Evolution:
(p=true) if Action=ac0;
(p=false) if Action=ac1;
end Evolution
```

if the agent executes `ac0`, then it will set the value of p to true, meanwhile if `ac1` is executed, then it will set the value of p to false.

2. *Evaluation function.* An evaluation function consists of a group of atomic propositions, which are defined over global states. Each atomic proposition is associated with a Boolean formula over local variables of all agents and observable variables in the Environment agent. The syntax is as follows:

```
Evaluation
<proposition_declaration>
end Evaluation
```

where `<proposition_declaration>` is a sequence of lines of the form `<proposition> if <condition_on_states>`, where `<proposition>` is a valid ISPL identifier (not a reserved keyword) and `<condition_on_states>` is a Boolean formula.

3. *Initial states.* Initial states are defined by a Boolean formula over variables. It goes by the following syntax:

```
InitStates
<condition_on_states>
end InitStates
```

where, similar to the evaluation function, `<condition_on_states>` is a Boolean formula over all variables of the agents and the Environment.

4. *Groups declaration.* Groups are used in formulae involving group modalities. A group consists of one or more agents, including the Environment. The syntax is as follows:

```
Groups
<groups_declaration>
end Groups
```

where `<groups_declaration>` takes the following form

```
name_of_group = {agent_0,...,agent_n,Environment}
```

5. *List of formulae to be verified.* A formula to be checked is defined over atomic proposition. The syntax is as follows:

```
Formulae
<formulae_list>
end Formulae
```

where `<formulae_list>` is a sequence of *Strategy Logic* formulae, with the following syntax.

SL Formula	ISPL Syntax
$\neg\varphi$	! *formula*
$\varphi \wedge \varphi$	*formula* and *formula*
$\varphi \vee \varphi$	*formula* or *formula*
$\mathbf{X}\varphi$	X *formula*
$\mathbf{F}\varphi$	F *formula*
$\mathbf{G}\varphi$	G *formula*
$\varphi\mathcal{U}\varphi$	*formula* U *formula*
$\langle\langle x\rangle\rangle\varphi$	<<*variable*>> *formula*
$[\![x]\!]\varphi$	[[*variable*]] *formula*
$(i,x)\varphi$	(*agent*,*variable*) *formula*

“Agent”	
“Lstate”	(local state)
“Lgreen”	(green local state)
“Action”	
“Protocol”	
“Ev”	(evolution function)
“Evaluation”	
“InitStates”	
“Groups”	
“Formulae”	
“end”	
“if”	
“and”	
“or”	
“->”	(implication)
“AG”	(temporal operator AG)
“EG”	(temporal operator EG)
“AX”	(temporal operator AX)
“EX”	(temporal operator EX)
“X”	(temporal operator X)
“F”	(temporal operator F)
“G”	(temporal operator G)
“AF”	(temporal operator AF)
“EF”	(temporal operator EF)
“A”	(universal path quantifier)
“E”	(existential path quantifier)
“U”	(temporal operator U)
“K”	(epistemic operator)
“GK”	(epistemic operator for everyone knows)
“GCK”	(epistemic operator for common knowledge)
“O”	(operator for correct behaviour)
“KH”	(operator for epistemic+deontic modality)
“DK”	(epistemic operator for distributed knowledge)

Figure 3.1: ISPL reserved keywords

Figure 3.1 shows reserved keywords in ISPL, and Figure 3.2 shows the general structure of an interpreted system represented in ISPL code.

```
Semantics = MultiAssignment | SingleAssignment
Agent Environment
Obsvars:
...
end Obsvars
Vars:
...
end Vars
Actions = {...};
Protocol:
...
end Protocol
Evolution:
...
end Evolution
end Agent

Agent m0
Vars:
...
end Vars
Actions = {...};
Protocol:
...
end Protocol
Evolution:
...
end Evolution
end Agent

Evaluation
...
end Evaluation

InitStates
...
end InitStates

Groups
...
end Groups

Fairness
...
end Fairness

Formulae
...
end Formulae
```

Figure 3.2: General structure of ISPL code

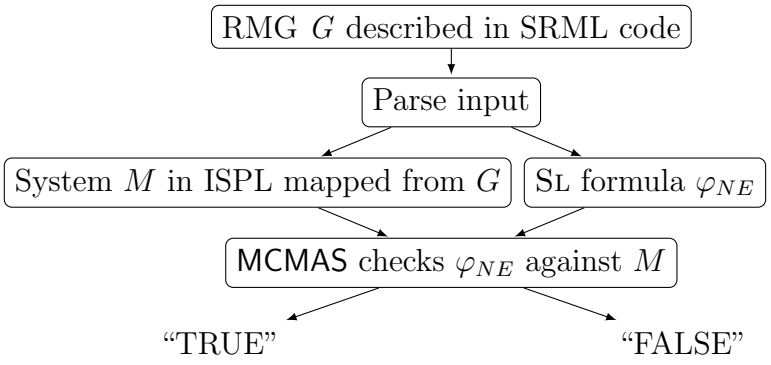

Figure 3.3: The general flow of the approach.

3.3 Rational Verification with MCMAS

In this section, we describe our approach to performing rational verification of multi-agent systems represented as LTL RMGs using MCMAS. To this end, we need to translate SRML code, used in RMGs, into ISPL code and perform rational verification using MCMAS engine. In particular, we want to use MCMAS to solve the NE-Emptiness problem of our LTL RMGs. The general flow of this approach is shown in Figure 3.3.

3.4 Translating SRML to ISPL

The high-level nature of SRML allows it to represent the specifications of open systems in a concise manner. The challenge of translating SRML to ISPL is finding a way to fill the gap of SRML's abstraction. In this section, we present a technique to translate SRML into ISPL and in doing so to be able to do equilibrium analysis of multi-agent systems modelled as SRML games.

3.4.1 States, Actions, and Variables in ISPL

Let A be an SRML arena $A = (\mathrm{N}, \Phi, m_1, \ldots, m_n)$, and m_i an SRML module $m_i = (\Phi_i, I_i, U_i)$ that we want to map into an agent in IS $i = (L_i, Act_i, Pr_i, t_i)$. The set of local states L_i can be built from all possible valuations of Boolean variables that are *visible* to m_i.

Since each module m_i, with $i \in \mathrm{N}$, controls a set of variables Φ_i and for each $x \in \Phi_i$, it can be set to either true or false, then there will be $2^{|\Phi_i|}$ possible dis-

tinct configurations of variables valuations in Φ_i which corresponds to the number of possible actions in Act_i of agent i.

The list of variables controlled by module m_i can be put directly into agent i's `Vars` section in ISPL. However, note that variables in the `Vars` list are private variables. These variables are only visible to agent i. To simulate the notion of public variable, which is a default assumption in the perfect information setting, we need to apply a "mirroring" technique. This technique is presented in the next section.

3.4.2 Simulating Public Variables in ISPL

Only variables placed inside the `Obsvars` tag in Agent Environment are visible to all agents in the system. We exploit this feature to make private variables in normal agents be public. This technique in principle employs a mirroring method, that is, making copies of private variables and placing them inside the `Agent Environment`'s `Obsvars` tag. The variable values are also updated according to the behaviour of corresponding normal agents. The following example shows how it works:

```
1   Agent Environment
2   Obsvars:
3   copy_of_p: boolean;
4   end Obsvars
5   ...
6   Evolution:
7   copy_of_p = false if i.Action = ac0;
8   copy_of_p = true if i.Action = ac1;
9   end Evolution
10  ...
11
12  Agent i
13  Vars:
14  p: boolean;
15  end Vars
16  ...
17  Evolution:
18  p = false if Action = ac0;
19  p = true if Action = ac1;
20  end Evolution
21  ...
22  InitStates
23  ...
24  Environment.copy_of_p=i.p;
25  ...
26  end InitStates
27  ...
```

Using this approach, the value of `copy-of-p` is the same as the value of `p` in initial states and each time-step.

3.4.3 Initial States in ISPL

Since SRML allows a module to have more than one **init** guarded command (hence, be non-deterministic), we could capture this by executing all possible combinations of "choices" (enabled **init** guarded commands) in the initialisation of the variables in Φ.

Let A be an SRML arena $A = (\mathrm{N}, \Phi, m_1, \ldots, m_n)$ and M be an interpreted system $M = (\mathrm{N}, \mathcal{I}, \Pi)$. We can build the set of global initial states $\mathcal{I} \subseteq \mathscr{G}$ from the product of the set of **init** guarded commands of each module m_i in A,that is, $enabled(v_\perp) = enabled_1(v_\perp) \times \cdots \times enabled_n(v_\perp)$. Now, let $J = enabled(v_\perp)$. Then we have $J \ni j = \{g_1^k, \ldots, g_n^k\}$, where $g_i^k \in I_i$. Assume without lost of generality that each module m_i has t number of **init** guarded commands and J has u number of elements, hence, $\mathcal{I} = exec((g_1^1, \ldots, g_n^1), v_\perp) \cup \cdots \cup exec((g_1^t, \ldots, g_n^t), v_\perp) = exec(j_1, v_\perp) \cup \cdots \cup exec(j_u, v_\perp) = \bigcup_{j_i \in J} exec(j_i, v_\perp)$.

Thus, **Init** sections that contain only one command can be mapped directly to `InitStates` of ISPL. Otherwise, the non-deterministic initialisation of variables can be simulated using the `or` connective. To make it clearer, consider the following SRML modules[2]:

module m_a **controls** p, q	**module** m_b **controls** r, s
init	**init**
$:: true \rightsquigarrow p' := true, q' := false;$	$:: true \rightsquigarrow r' := true, s' := false;$
	$:: true \rightsquigarrow r' := false, s' := true;$

These initialisations can be translated into ISPL as follows:

```
1   InitStates
2   (ma.p=true and ma.q=false)
3   and
4   (mb.r=true and mb.s=false) or (mb.r=false and mb.s=true);
5   end InitStates
```

[2]The **update** sections are ommitted since they are not relevant for this particular example.

3.4.4 Protocols in ISPL

Given an SRML module $m_i = (\Phi_i, I_i, U_i)$ that has a set of **update** guarded commands $U_i = \{g_1, g_2, \ldots, g_j\}$, such that for each $g_l \in U_i$ we have $ctr(g_l) = \{x_1, x_2, \ldots, x_k\}$, where $x_m \in \Phi_i$, the **update** commands in SRML are as follows:

$$\begin{array}{l}\textbf{update}\\ :: \varphi_1 \rightsquigarrow x'_1 := \psi_{1,1}; \ldots; x'_k := \psi_{1,k};\\ \vdots\\ :: \varphi_j \rightsquigarrow x'_1 := \psi_{j,1}; \ldots; x'_k := \psi_{j,k};\end{array}$$

then we can build the protocol function Pr_i of agent i in ISPL using Algorithm 7, which runs in exponential time with respect to the number of variables visible to module m_i, that is, variables that appear in $guard(g)$, with $g \in U_i$. We write $Pr_i(l_i)$ to denote the protocol for the local state $l_i \in L_i$ and $t_i(l_i, a_i)$ to denote the set of local states that can be reached in accordance with agent i's local transition function t_i from l_i by executing action a_i.

Algorithm 7 Build Protocol

```
1:  function BUILDPRO(N, Φ, m_1, ..., m_n)
2:      Pr_i := ∅;
3:      L_i := V;
4:      for each v ∈ V, l_i ∈ L_i do
5:          for each g_i ∈ U_i do
6:              x'_1 := ∅; ...; x'_k := ∅;
7:              if g_i ∈ enabled(v) then
8:                  for each x_i ∈ ctr(g_i) do
9:                      if x_i ∈ exec(g_i, v) then
10:                         x'_i := ⊤;
11:                     else
12:                         x'_i := ⊥;
13:                     end if
14:                 end for
15:                 find a_i ∈ Act_i such that l'_i ∈ t_i(l_i, a_i) and
                    Π|_l(l'_i) = {x'_1, ..., x'_k};
16:                 if a_i ∉ Pr_i(l_i) then
17:                     Pr_i(l_i) := Pr_i(l_i) ∪ a_i;
18:                 end if
19:             end if
20:         end for
21:     end for
22:     return Pr_i;
23: end function
```

Observe that for every SRML module $m_i = (\Phi_i, I_i, U_i)$ that we want to map into an agent $i = (L_i, Act_i, Pr_i, t_i)$ in ISPL, we will have a set of protocol Pr_i of size $|L_i|$. This directly follows from Algorithm 7, line 4. Also, observe that for each $l_i \in L_i$, the size of $Pr_i(l_i)$ is at most $2^{|\Phi_i|}$, which follows from Algorithm 7, line 5. Suppose the guard of each $g_l \in U_i$ is satisfied/true in state l_i and the action part of g_l is *complete* and *unique*. That is, there is no such g_v and g_w, $v \neq w$, and for all $y \in ctr(g_v)$ if and only if $y \in ctr(g_w)$ such that they give the same configuration in the next time-step for each variable y. Then, there will be at most 2^n possible distinct configurations, where n is the total number of variables controlled by m_i, hence $|j| = |Act_i| = 2^{|\Phi_i|}$.

To reason about the correctness of Algorithm 7 in building the protocol function of agent i, we illustrate it via a toy model in Example 6. Observe that each valuation v represents the local state l_i of agent i, therefore the set $enabled_i(v)$ corresponds to Algorithm 7, line 7. It then proceeds to determine which action $a_i \in Act_i$ can be included in $Pr_i(l_i)$. This is done by finding the corresponding action of agent i with respect to $exec(g, v)$, where $g \in enabled(v)$. Let $v_a^1 = \{p, q\} = \Pi|_l(l_a^1)$, then we have $enabled_a(v_a^1) = \{g_a^2, g_a^3\}$ and agent m_a has the set of actions in ISPL shown in Table 3.1. Thus executing g_a^2 on v_a^1 will give p the valuation true, whereas executing g_a^3 on v_a^1 will give p the valuation false, hence $exec(g_a^2, v_a^1) = \{p, q\}$ and $exec(g_a^3, v_a^1) = \{q\}$. Notice that $enabled_a(v_a^1)$ corresponds to a_a^1 and $exec(g_a^3, v_a^1)$ to a_a^2 (Algorithm 7, line 15). Therefore we put both a_a^1 and a_a^2 into $Pr_a(l_a^1)$ (see Algorithm 7, line 17) and have $Pr_a(l_a^1) = \{a_a^1, a_a^2\}$. Applying the technique above for each valuation v_a (which corresponds to local state l_a), agent m_a protocol function is complete as shown in Table 3.2.

Example 6. Let $\Phi = \{p, q\}$ be a set of Boolean variables and consider the SRML arena $A = (\{a, b\}, \{p, q\}, m_a, m_b)$. The specification of m_a and m_b is as follows:

module m_a **controls** p		**module** m_b **controls** q	
init		**init**	
$:: \top \rightsquigarrow p' := \top;$	(g_a^1)	$:: \top \rightsquigarrow q' := \bot;$	(g_b^1)
update		**update**	
$:: p \vee q \rightsquigarrow p' := \top;$	(g_a^2)	$:: p \wedge \neg q \rightsquigarrow q' := p;$	(g_b^2)
$:: q \rightsquigarrow p' := \neg p;$	(g_a^3)	$:: p \leftrightarrow q \rightsquigarrow q' := \neg q;$	(g_b^3)

Thus, we have

$$
\begin{array}{ll}
enabled_a(\{p,q\}) = \{g_a^2, g_a^3\} & enabled_b(\{p,q\}) = \{g_b^3\} \\
enabled_a(\{p\}) = \{g_a^2\} & enabled_b(\{p\}) = \{g_b^2\} \\
enabled_a(\{q\}) = \{g_a^2, g_a^3\} & enabled_b(\{q\}) = \{g_b^{\mathbf{skip}}\} \\
enabled_a(\{\}) = \{g_a^{\mathbf{skip}}\} & enabled_b(\{\}) = \{g_b^3\}
\end{array}
$$

name	assignment
a_a^1	$p := \top$
a_a^2	$p := \bot$

Table 3.1: The set of actions of agent m_a in ISPL.

local state	valuation	protocol
l_a^1	$\{p,q\}$	$Pr_a(l_a^1) = \{a_a^1, a_a^2\}$
l_a^2	$\{p\}$	$Pr_a(l_a^2) = \{a_a^1\}$
l_a^3	$\{q\}$	$Pr_a(l_a^3) = \{a_a^1\}$
l_a^4	$\{\}$	$Pr_a(l_a^4) = \{\}$

Table 3.2: The protocol function of agent m_a.

To implement $g_i^{\mathbf{skip}}$ we observe that such a guarded command can be implemented by setting the variables in Φ_i to have the same valuations as in the previous step. Thus, implicitly we have the following update guarded command

$$g_i^{\mathbf{skip}}: \quad \varphi \rightsquigarrow x_1' := x_1, \ldots, x_k' := x_k;$$

where $\Phi_i = \{x_1, \ldots, x_k\}$. That is, for every SRML module $m_i = (\Phi_i, I_i, U_i)$ that we want to map into an agent $i = (L_i, Act_i, Pr_i, t_i)$ in ISPL, for each $v \in V$, if $enabled(v)$ is empty, then it is equivalent that agent i executes $a_i \in Act_i$ such that $t_i(l_i, a_i) = \{l_i\}$. Based on this observation, we then designed Algorithm 8 to implement this feature.

It is straightforward that by using the technique presented in Section 3.4.1, the private variables inside the `Vars` tag can capture all possible valuations of the variables controlled by $m_i = (\Phi_i, I_i, U_i)$. The perfect information setting can be simulated using the technique presented in Section 3.4.2. The non-deterministic initialisation of the variables in Φ can be captured using the technique in Section 3.4.3. We can use the last statement as our basis for inductively build an IS M from the SRML game $\mathcal{G}$. Let K_A be the Kripke structure induced by $A \in G$, K_M the Kripke structure induced by M, $\rho_A \in Paths(K_A)$, and $\rho_M \in Paths(K_M)$. Assume that at a given time-step $j = k$, $\pi(\rho_A[j]) = \Pi(\rho_M[j])$. Since at any given time-step j each of agent $i \in \mathrm{N}, \mathrm{N} \in M$ acts

Algorithm 8 Implement $g^{\mathbf{skip}}$

```
1: function GSKIP(Pr_i)
2:     for each Pr_i(l_i) ∈ Pr_i do
3:         if Pr_i(l_i) = ∅ then
4:             find a_i ∈ Act_i such that t_i(l_i, a_i) = {l_i};
5:             Pr_i(l_i) := a_i;
6:         end if
7:     end for
8:     return Pr_i;
9: end function
```

according to the behaviour of each of module $m_i \in \mathrm{N}, \mathrm{N} \in \mathcal{G}$, then for $j = k + 1$, we also have $\pi(\rho_A[j]) = \Pi(\rho_M[j])$. Thus, given an SRML arena $A = (N, \Phi, m_1, \ldots, m_n)$, we can construct an IS $M = (\mathrm{N}, \mathcal{I}, \Pi)$ such that for every LTL formula φ, we have $K_A \models \varphi$ if and only if $K_M \models \varphi$, where K_A is the Kripke structure induced by A and K_M is the one induced by M.

Let σ_i^A be a strategy of module $m_i = (\Phi_i, I_i, U_i)$ in arena $A = (\mathrm{N}, \Phi, m_1, \ldots, m_n)$, and σ_i^M a strategy of mapped agent i in IS $M = (\mathrm{N}, \mathcal{I}, \Pi)$. We now define a congruence relation $\cong$ over strategies σ_i^A and σ_i^M by $\sigma_i^A \cong \sigma_i^M$ if and only if for every $v \in V$ and for every $l_i \in \not{\mathcal{I}}$ such that $v = \Pi|_l(l_i)$, we have $\sigma_i(v) = \Pi|_l(t_i(l_i, a_i))$, $a_i \in \sigma_i^M(l_i)$. We then extend the congruence relation to strategy profiles. We say that two strategy profiles $\vec{\sigma}^A$ and $\vec{\sigma}^M$ are congruent by $\vec{\sigma}^A \cong \vec{\sigma}^M$ if and only if for every $m_i \in \mathrm{N},\ \mathrm{N} \in A$ and for every $i \in \mathrm{N},\ \mathrm{N} \in M$, we have $\sigma_i^A \cong \sigma_i^M$.

Let $\mathcal{G} = (A, \gamma_1, \ldots, \gamma_n)$ be a LTL RMG, IS $M = (\mathrm{N}, \mathcal{I}, \Pi)$ a system constructed from A with γ_i be the goal of agent $i \in \mathrm{N}$, $\vec{\sigma}_A$ the strategy profile of the game $\mathcal{G}$ with the arena A, and $\vec{\sigma}_M$ the strategy profile of the IS M. It follows that, if $\vec{\sigma}^A \cong \vec{\sigma}^M$ then $\vec{\sigma}^A \models \gamma_i$ if and only if $\vec{\sigma}^M \models \gamma_i$.

3.5 Solving Rational Verification Problems with MCMAS

One of the decision problems in rational verification is checking the emptiness of the set of Nash equilibria. Formally, such a problem can be stated as follows:

NON-EMPTINESS
Given : Given a game $\mathcal{G}$.
Question : Is it the case that $\mathrm{NE}(\mathcal{G}) \neq \varnothing$?

Let $\mathrm{N} = \{1, \ldots, n\}$ be the set of players in $\mathcal{G}$, Var be the set of strategy variables, and Γ be the set of goals of players in $\mathcal{G}$. Using SL, we can express the existence of Nash equilibria with the formula φ_{NE}:

$$\varphi_{\mathrm{NE}} = \langle\langle x_1 \rangle\rangle (1, x_1) \ldots \langle\langle x_n \rangle\rangle (n, x_n) \bigwedge_{i \in \mathrm{N}} \Big(\neg\gamma_i \rightarrow [\![y_i]\!] (i, y_i) \neg\gamma_i \Big)$$

where $i \in N$, $x_i, y_i \in Var$, $\gamma_i \in \Gamma$[3].

Intuitively, formula φ_{NE} can be explained as follows: for each player i with its chosen strategy x_i in a game $\mathcal{G}$, if the goal of i cannot be achieved using strategy x_i then for every "alternative" strategy y_i, the goal of player i cannot be achieved. This means that, players who do not get their goals achieved cannot benefit from unilaterally changing their strategies. Thus, if φ_{NE} is true in $\mathcal{G}$, then there exists a Nash equilibrium in the game.

The other problems of rational verification, namely E-Nash and A-Nash, can be reduced to Non-emptiness [Gao et al., 2017]. The reduction is straightforward and involves an addition of two extra players. The construction from E-Nash to Non-emptiness is as follows. Given a game G and a property φ, build a new game $\mathcal{H}$ by adding two more players, say $n+1$ and $n+2$, with goals $\gamma_{n+1} = \varphi \vee (p \leftrightarrow q)$ and $\gamma_{n+2} = \varphi \vee \neg(p \leftrightarrow q)$, where $\Phi_{n+1} = \{p\}$ and $\Phi_{n+2} = \{q\}$ for two fresh Boolean variables p and q. With such construction, it is the case that the answer to the E-Nash problem of game $\mathcal{G}$ and property φ is a "yes" if, and only if, the answer to the Non-emptiness problem of $\mathcal{H}$ is a "yes" [Gao et al., 2017]. For A-Nash, it is straightforward, since it is (logically) the dual of E-Nash.

3.6 Summary

In this chapter, a technique for translating RMGs modelled in SRML code into MCMAS's dedicated language ISPL in order to perform equilibrium analysis on the games is presented. The technique has been implemented in SEVIA (SRML Games Equilibrium Verification via ISPL Analysis)[4]. At this point, it is important to point out that this approach only solves the Non-emptiness problem under imperfect recall strategies setting, since (full) SL with perfect recall strategies is currently not supported by

[3]This characterisation of the existence of Nash equilibria using SL has been done in the literature, e.g., in [Gao et al., 2017].

[4]It can be used online from: `http://eve.cs.ox.ac.uk/sevia.html`. The source code is also available from: `https://github.com/eve-mas/sevia`.

MCMAS[5]. Moreover, although MCMAS supports the analysis of games with imperfect information and non-uniform strategies, in this thesis we only look at games with perfect information and uniform strategies. Indeed, in this thesis we only consider an extension of MCMAS called MCMAS-SLK, whose (SL) semantics defined on uniform memoryless strategies. Later in Section 7.4.2, we will see a concrete example of how imperfect recall setting affects the result of an analysis of some system.

[5]Full SL is implemented in an extension of MCMAS called MCMAS-SLK[Cermák et al., 2014, Cermák et al., 2018]. However, MCMAS-SLK only supports imperfect recall strategies, since games with more than two players, perfect recall strategies, and imperfect information, give rise to undecidable problems [Gutierrez et al., 2018b].

Chapter 4

Parity Games for Rational Verification and Synthesis

This chapter presents a novel and simpler technique for checking the existence of Nash equilibria in games where players have goals given by LTL formulae. In particular, the technique described in this chapter does not rely on the solution of an alternating parity tree automaton. Instead, it reduces the problem to the solution of a parity game [Emerson and Jutla, 1991]. In the next section some preliminary definitions are given. In Section 4.1 the overall algorithm is outlined. In Section 4.2 the translation from a game with LTL goals to a parity game is presented. Section 4.3 provides a characterisation of the set of Nash equilibria within the parity game representation previously obtained, while Section 4.4 shows how to check for nonemptiness of such a set using Streett automata. Section 4.5 provides a way to solve synthesis and verification problems via Non-emptiness. Section 4.6 discusses the need and aspiration for bisimulation-invariant setups.

4.1 Reasoning with Parity Games

We now state the problem that is studied in the rest of this chapter. The problem, as already mentioned in Section 2.8.1, is called Non-Emptiness and formally defined as follows:

Given: An LTL Game $\mathcal{G}_{\mathsf{LTL}}$.
Question: Is it the case that $\mathrm{NE}(\mathcal{G}_{\mathsf{LTL}}) \neq \emptyset$?

As indicated before, we solve both verification and synthesis through a reduction to the above problem. The technique we develop consists of three steps. First, we

build a (deterministic) Parity game $\mathcal{G}_{\text{PAR}}$ from an input LTL game $\mathcal{G}_{\mathsf{LTL}}$[1]. Then—using a characterisation of Nash equilibrium (presented later) that separates players in the game into those that achieve their goals in a Nash equilibrium (the "winners", W) and those that do not achieve their goals (the "losers", L)—for each set of players in the game, we eliminate nodes and paths in $\mathcal{G}_{\text{PAR}}$ which cannot be a part of a Nash equilibrium, thus producing a modified Parity game, $\mathcal{G}_{\text{PAR}}^{-L}$. Finally, in the third step, we use Streett automata on infinite words to check if the obtained Parity game witnesses the existence of a Nash equilibrium. The overall algorithm is presented in Algorithm 9 which also includes some comments pointing to the relevant Sections/Theorems. The first step is contained in line 3, while the third step is in lines 12–14. The rest of the algorithm is concerned with the second step. In the sections that follow, we will describe each step of the algorithm and, in particular, what are and how to compute $\text{Pun}_j(\mathcal{G}_{\text{PAR}})$ and $\mathcal{G}_{\text{PAR}}^{-L}$, two key constructions used in our decision procedure. At this point, it is important to note that in Algorithm 9, it is implicitly assumed that the set W is not an empty set, since in that case, checking the existence of Nash equilibrium is trivial—we need only to check whether $\mathcal{G}_{\text{PAR}}^{-L}$ is empty of not, since all infinite runs in it are Nash equilibrium runs. We choose not to incorporate this trivial case in order to avoid clutter.

Algorithm 9 Nash equilibrium via Parity games

1: **input:** An LTL game $\mathcal{G}_{\mathsf{LTL}} = (\text{N}, (\text{Ac}_i)_{i\in\text{N}}, \text{St}, s_0, \mathsf{tr}, \lambda, (\gamma_i)_{i\in\text{N}})$.
2: **output:** "Yes" if $\text{NE}(\mathcal{G}_{\mathsf{LTL}}) \neq \emptyset$; "No" otherwise.
3: $\mathcal{G}_{\text{PAR}} \Longleftarrow \mathcal{G}_{\mathsf{LTL}}$; ▷ from Section 4.2 (Theorem 8)
4: **for each** $W \subseteq \text{N}$ **do**
5: **for each** $j \in L = \text{N} \setminus W$ **do**
6: **Compute** $\text{Pun}_j(\mathcal{G}_{\text{PAR}})$; ▷ from Section 4.3 (Theorem 9)
7: **end for**
8: **Compute** $\mathcal{G}_{\text{PAR}}^{-L}$;
9: **for each** $i \in W$ **do**
10: **Compute** $\mathcal{A}_i$ and $\mathcal{S}_i$ from $\mathcal{G}_{\text{PAR}}^{-L}$;
11: **end for**
12: **if** $\mathcal{L}(\bigtimes_{i\in W}(\mathcal{S}_i)) \neq \emptyset$ **then** ▷ from Section 4.3 (Theorem 11)
13: **return** "Yes"
14: **end if**
15: **end for**
16: **return** "No"

[1]Readers might ask the usage of parity objective in our approach—as opposed to some simpler ones, such as Büchi. The reason is that, in general, it is not always possible to translate an LTL formula into a deterministic Büchi automaton.

It should be noted that even though we use automata in our algorithm (specifically, Streett automata to check for the existence of a Nash equilibrium in line 12 of the algorithm, and parity automata to build a parity game in line 3 of the algorithm), most reasoning is done at the level of parity games, *i.e.*, in the second step of the decision procedure.

Complexity. The procedure presented above runs in doubly exponential time, matching the optimal upper bound of the problem. In the first step we obtain a doubly exponential blowup. The underlying structure $\mathcal{M}$ of the obtained Parity game $\mathcal{G}_{\text{PAR}}$ is doubly exponential in the size of the goals of the input LTL game $\mathcal{G}_{\mathsf{LTL}}$, but the priority functions set $(\alpha_i)_{i\in\text{N}}$ is only (singly) exponential. Then, in the second step, reasoning takes only polynomial time in the size of the underlying concurrent game structure of $\mathcal{G}_{\text{PAR}}$, but exponential time in both the number of players and the size of the priority functions set. Finally, the third step takes only polynomial time, leading to an overall $\mathsf{2EXPTIME}$ complexity.

4.2 LTL Games to Parity Games

We now describe how to realise line 3 of Algorithm 9, and in doing so we prove a strong correspondence between the set of Nash equilibria of the input LTL game $\mathcal{G}_{\mathsf{LTL}}$ and the set of Nash equilibria of its associated Parity game $\mathcal{G}_{\text{PAR}}$. This result will allow us to shift reasoning on the set of Nash equilibria of $\mathcal{G}_{\mathsf{LTL}}$ into reasoning on the set of Nash equilibria of $\mathcal{G}_{\text{PAR}}$. The basic idea behind this step of the decision procedure is to transform all LTL goals $(\gamma_i)_{i\in\text{N}}$ in $\mathcal{G}_{\mathsf{LTL}}$ into a collection of DPWs, denoted by $(\mathcal{A}_{\gamma_i})_{i\in\text{N}}$, that will be used to build the underlying CGS of $\mathcal{G}_{\text{PAR}}$. We construct $\mathcal{G}_{\text{PAR}}$ as follows.

In general, using the results in [Sistla et al., 1987, Piterman, 2007], from any LTL formula φ over AP one can build a DPW $\mathcal{A}_\varphi = (2^{\text{AP}}, Q, q^0, \rho, \alpha)$ such that, $\mathcal{L}(\mathcal{A}_\varphi) = \{\pi \in (2^{\text{AP}})^\omega : \pi \models \varphi\}$, that is, the language accepted by $\mathcal{A}_\varphi$ is exactly the set of words over 2^{AP} that are models of φ. The size of Q is doubly exponential in $|\text{AP}|$ and the size of the range of α is singly exponential in $|\text{AP}|$. Using this construction we can define, for each LTL goal γ_i, a DPW $\mathcal{A}_{\gamma_i}$.

Definition 26. Let $\mathcal{G}_{\mathsf{LTL}} = (\mathcal{M}, (\gamma_i)_{i\in\text{N}})$ be an LTL game whose underlying CGS is $\mathcal{M} = (\text{N}, (\text{Ac}_i)_{i\in\text{N}}, \text{St}, s_0, \mathsf{tr}, \lambda)$. Moreover, let $\mathcal{A}_{\gamma_i} = (2^{\text{AP}}, Q_i, q_i^0, \rho_i, \alpha_i)$ be the DPW corresponding to player i's goal γ_i in $\mathcal{G}_{\mathsf{LTL}}$. The *Parity game* $\mathcal{G}_{\text{PAR}}$ *associated to* $\mathcal{G}_{\mathsf{LTL}}$ is

the game $\mathcal{G}_{\text{PAR}} = (\mathcal{M}', (\alpha'_i)_{i\in\text{N}})$, where $\mathcal{M}' = (\text{N}, (\text{Ac}_i)_{i\in\text{N}}, \text{St}', s'_0, \mathsf{tr}', \lambda)$ and $(\alpha'_i)_{i\in\text{N}}$ are as follows:

- $\text{St}' = \text{St} \times \bigtimes_{i\in\text{N}} Q_i$ and $s'_0 = (s_0, q_1^0, \ldots, q_n^0)$;
- for each state $(s, q_1, \ldots, q_n) \in \text{St}'$ and action profile $\vec{a}$, $\mathsf{tr}'((s, q_1, \ldots, q_n), \vec{a}) = (\mathsf{tr}(s, \vec{a}), \rho_1(q_1, \lambda(s)), \ldots, \rho_n(q_n, \lambda(s))$;
- $\alpha'_i(s, q_1, \ldots q_n) = \alpha_i(q_i)$.

Intuitively, the game $\mathcal{G}_{\text{PAR}}$ is the product of the LTL game $\mathcal{G}_{\mathsf{LTL}}$ and the collection of parity word automata $\mathcal{A}_{\gamma_i}$ that recognise the models of each player's goal. Informally, the game executes in parallel the original LTL game together with the automata built on top of the LTL goals. At every step of the game, the first component of the product state follows the transition function of the original game $\mathcal{G}_{\mathsf{LTL}}$, while the "automata" components are updated according to the labelling of the current state of $\mathcal{G}_{\mathsf{LTL}}$. As a result, the execution in $\mathcal{G}_{\text{PAR}}$ is made, component by component, by the original execution, say π, in the LTL game $\mathcal{G}_{\mathsf{LTL}}$, paired with the unique runs of the DPWs $\mathcal{A}_{\gamma_i}$ generated when reading the word $\lambda(\pi)$.

Observe that in the translation from $\mathcal{G}_{\mathsf{LTL}}$ to its associated $\mathcal{G}_{\text{PAR}}$ the set of actions for each player is unchanged. This, in turn, means that the set of strategies in both $\mathcal{G}_{\mathsf{LTL}}$ and $\mathcal{G}_{\text{PAR}}$ is the same, since for every state $s \in \text{St}$ and action profile $\vec{a}$, it follows that $\vec{a}$ is available in s if and only if it is available in $(s, q_1, \ldots, q_n) \in \text{St}'$, for all $(q_1, \ldots, q_n) \in \bigtimes_{i\in\text{N}} Q_i$. Using this correspondence between strategies in $\mathcal{G}_{\mathsf{LTL}}$ and strategies in $\mathcal{G}_{\text{PAR}}$, we can prove the following Lemma, which states an invariance result between $\mathcal{G}_{\mathsf{LTL}}$ and $\mathcal{G}_{\text{PAR}}$ with respect to the satisfaction of players' goals.

Lemma 2 (Goals satisfaction invariance)**.** Let $\mathcal{G}_{\mathsf{LTL}}$ be an LTL game and $\mathcal{G}_{\text{PAR}}$ its associated parity game. Then, for every strategy profile $\vec{\sigma}$ and player i, it is the case that $\pi(\vec{\sigma}) \models \gamma_i$ in $\mathcal{G}_{\mathsf{LTL}}$ if and only if $\pi(\vec{\sigma}) \models \alpha_i$ in $\mathcal{G}_{\text{PAR}}$.

Proof. We prove the statement by double implication. To show the left to right implication, assume that $\pi(\vec{\sigma}) \models \gamma_i$ in $\mathcal{G}_{\mathsf{LTL}}$, for any player $i \in \text{N}$, and let π denote the infinite path generated by $\vec{\sigma}$ in $\mathcal{G}_{\mathsf{LTL}}$; thus, we have that $\lambda(\pi) \models \gamma_i$. On the other hand, let π' denote the infinite path generated in $\mathcal{G}_{\text{PAR}}$ by the same strategy profile $\vec{\sigma}$. Observe that the first component of π' is exactly π. Moreover, consider the $(i+1)$-th component ρ_i of π'. By the definition of $\mathcal{G}_{\text{PAR}}$, it holds that ρ_i is the run executed by the automaton $\mathcal{A}_{\gamma_i}$ when the word $\lambda(\pi)$ is read. By the definition of the labelling function of $\mathcal{G}_{\text{PAR}}$, it holds that the parity of π' according to α'_i corresponds to the one

recognised by $\mathcal{A}_{\gamma_i}$ in ρ_i. Thus, since we know that $\lambda(\pi) \models \gamma_i$, it follows that ρ_i is accepting in $\mathcal{A}_{\gamma_i}$ and therefore $\pi' \models \alpha_i$, which implies that $\pi(\vec{\sigma}) \models \alpha_i$ in $\mathcal{G}_{\text{PAR}}$.

For the right to left direction, observe that all implications used above are equivalences. Thus, using such equivalences we can reason backwards to prove the statement. □

Using Lemma 2 we can then show that the set of Nash Equilibria for any LTL game exactly corresponds to the set of Nash equilibria of its associated Parity game. Formally, we have the following invariance result between games.

Theorem 8. Let $\mathcal{G}_{\mathsf{LTL}}$ be an LTL game and $\mathcal{G}_{\text{PAR}}$ its associated Parity game. Then, it is the case that $\text{NE}(\mathcal{G}_{\mathsf{LTL}}) = \text{NE}(\mathcal{G}_{\text{PAR}})$.

Proof. The proof proceeds by double inclusion. First, assume that a strategy profile $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\mathsf{LTL}})$ is a Nash Equilibrium in $\mathcal{G}_{\mathsf{LTL}}$ and, by contradiction, it is not a Nash Equilibrium in $\mathcal{G}_{\text{PAR}}$. Observe that, due to Lemma 2, we know that the set of players that get their goals satisfied by $\pi(\vec{\sigma})$ in $\mathcal{G}_{\mathsf{LTL}}$ (the "winners", W) is the same set of players that get their goals satisfied by $\pi(\vec{\sigma})$ in $\mathcal{G}_{\text{PAR}}$. Then, there is player $j \in L = \text{N} \setminus W$ and a strategy σ'_j such that $\pi((\vec{\sigma}_{-j}, \sigma'_j)) \models \alpha_j$ in $\mathcal{G}_{\text{PAR}}$. Then, due to Lemma 2, we have that $\pi((\vec{\sigma}_{-j}, \sigma'_j)) \models \gamma_j$ in $\mathcal{G}_{\mathsf{LTL}}$ and so σ'_j would be a beneficial deviation for player j in $\mathcal{G}_{\mathsf{LTL}}$ too—a contradiction. On the other hand, for every $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\text{PAR}})$, we can reason in a symmetric way and conclude that $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\mathsf{LTL}})$. □

4.3 Nash Equilibria Characterisation

Thanks to Theorem 8, we can focus our attention on Parity games, since a technique for solving such games will also provide a technique for solving their associated LTL games. To do this we will characterise the set of Nash equilibria in the Parity game construction $\mathcal{G}_{\text{PAR}}$ in our algorithm.

The existence of Nash Equilibria in LTL games can be characterised in terms of punishment strategies and memoryful reasoning [Gutierrez et al., 2015a]. We will show that a similar characterisation can be shown here in a parity games framework, where only memoryless reasoning is required. To do this, we first introduce the notion of punishment strategies and regions formally, as well as some useful definitions and notations.

In what follows, given a memoryless strategy profile $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ defined on a state $s \in \text{St}$ of a Parity game $\mathcal{G}_{\text{PAR}}$, that is, such that $s_i^0 = s$ for every $i \in \text{N}$, we

write $\mathcal{G}_{\text{PAR}}, \vec{\sigma}, s \models \alpha_i$ if $\pi(\vec{\sigma}) \models \alpha_i$ in $\mathcal{G}_{\text{PAR}}$. Moreover, if $s = s_0$ is the initial state of the game, we omit it and simply write $\mathcal{G}_{\text{PAR}}, \vec{\sigma} \models \alpha_i$ in such a case.

Definition 27 (Punishment strategies and regions)**.** For a Parity game $\mathcal{G}_{\text{PAR}}$ and a player $i \in \text{N}$, we say that $\vec{\sigma}_{-i}$ is a *punishment strategy profile* against i in a state s if, for all strategies $\sigma'_i \in \Sigma_i$, it is the case that $\mathcal{G}_{\text{PAR}}, (\vec{\sigma}_{-i}, \sigma'_i), s \not\models \alpha_i$. A state s is *punishing* for i if there exists a punishment strategy profile against i in s. By $\text{Pun}_i(\mathcal{G}_{\text{PAR}})$ we denote the set of punishing states, the *punishment region*, for i in $\mathcal{G}_{\text{PAR}}$.

To better understand the meaning of a punishment strategy profile, it is useful to think of a modification of the game $\mathcal{G}_{\text{PAR}}$, in which player i still has its goal α_i, while the rest of the players are collectively playing in an adversarial mode, *i.e.*, trying to make sure that player i does not achieve α_i. This scenario is formally represented by a two-player zero-sum game, in which the winning strategies of the (coalition) players, denoted by $-i$, correspond (one-to-one) to the punishment strategies in the original game $\mathcal{G}_{\text{PAR}}$. As described in [Gutierrez et al., 2015a], knowing the set of punishment strategy profiles in a given game is important to compute its set of Nash Equilibria. For this reason, it is useful to compute the set $\text{Pun}_i(\mathcal{G}_{\text{PAR}})$, that is, the set of states in the game from which a given player i can be punished (*e.g.*, to deter undesirable unilateral player deviations). To do this, we reduce the problem to computing a winning strategy in a turn-based two-player zero-sum parity game, whose definition is as follows.

Definition 28. For a concurrent multi-player Parity game

$$\mathcal{G}_{\text{PAR}} = (\text{N}, \text{St}, (\text{Ac}_i)_{i \in \text{N}}, s_0, \mathsf{tr}, (\alpha_i)_{i \in \text{N}})^2$$

and player $j \in \text{N}$, the *sequentialisation* of $\mathcal{G}_{\text{PAR}}$ with respect to player j is the (turn-based two-player) parity game $\mathcal{G}^j_{\text{PAR}} = (V_0, V_1, E, \alpha)$ where

- $V_0 = \text{St}$ and $V_1 = \text{St} \times \vec{\text{Ac}}_{-j}$;
- $E = \{(s, (s, \vec{a}_{-j})) \in \text{St} \times (\text{St} \times \vec{\text{Ac}}_{-j})\} \cup \{((s, \vec{a}_{-j}), s') \in (\text{St} \times \vec{\text{Ac}}_{-j}) \times \text{St} \ : \ \exists a'_j \in \text{Ac}_j.\ s' = \mathsf{tr}(s, (\vec{a}_{-j}, a'_j))\}$;
- $\alpha : V_0 \cup V_1 \to \mathbb{N}$ is such that $\alpha(s) = \alpha_j(s) + 1$ and $\alpha(s, \vec{a}_{-j}) = \alpha_j(s) + 1$.

Figure 4.1: Sequentialisation of a game. On the left hand side, a representation of a transition from s_1 to s_2 using action profile $(\vec{a}_{-j}, a_j)$. On the right hand side, the two states s_1 and s_2 have been assigned to Player 0 in the parity game, which have been interleaved with a state of Player 1 corresponding to the choice of $\vec{a}_{-j}$ by coalition $-j$ in the original game.

The connection between the notion of punishment in $\mathcal{G}_{\text{PAR}}$ and the set of winning strategies in $\mathcal{G}^i_{\text{PAR}}$ is established in the following theorem, where by $\text{Win}_0(\mathcal{G}^j_{\text{PAR}})$ we denote the winning region of Player 0 in $\mathcal{G}^j_{\text{PAR}}$, that is, the states from which Player 0, representing the set of players $-j = \text{N} \setminus \{j\}$, has a (memoryless) winning strategy in the parity game $\mathcal{G}^j_{\text{PAR}}$.

Theorem 9. For all states $s \in \text{St}$, it is the case that $s \in \text{Pun}_j(\mathcal{G}_{\text{PAR}})$ if and only if $s \in \text{Win}_0(\mathcal{G}^j_{\text{PAR}})$. In other words, it holds that $\text{Pun}_j(\mathcal{G}_{\text{PAR}}) = \text{Win}_0(\mathcal{G}^j_{\text{PAR}}) \cap \text{St}$.

Proof. The proof goes by double inclusion. From left to right, assume $s \in \text{Pun}_j(\mathcal{G}_{\text{PAR}})$ and let $\vec{\sigma}_{-j}$ be a punishment strategy profile against player j in s, *i.e.*, such that $\mathcal{G}_{\text{PAR}}, (\vec{\sigma}_{-j}, \sigma'_j), s \not\models \alpha_j$, for every strategy $\sigma'_j \in \Sigma_j$ of player j. We now define a strategy σ_0 for player 0 in $\mathcal{G}^j_{\text{PAR}}$ that is winning in s. In order to do this, first observe that, for every finite path $\pi'_{\leq k} \in V^* \cdot V_0$ in $\mathcal{G}^j_{\text{PAR}}$ starting from s, there is a unique finite sequence of action profiles $\vec{a}^0_{-j}, \ldots, \vec{a}^k_{-j}$ and a sequence $\pi_{\leq k} = s^0, \ldots, s^{k+1}$ of states in St^* such that

$$\pi'_{\leq k} = s^0, (s^0, \vec{a}^0_{-j}), \ldots, s^k, (s^k, \vec{a}^k_{-j}), \ldots, s^{k+1} \ .$$

Now, for every path $\pi'_{\leq k}$ of this form that is consistent with $\vec{\sigma}_{-j}$, *i.e.*, the sequence $\vec{a}^0_{-j}, \ldots, \vec{a}^{k-1}_{-j}$ is generated by $\vec{\sigma}_{-j}$, define $\sigma_0(\pi'_{\leq k}) = (s^{k+1}, \vec{a}^{k+1}_{-j})$, where $\vec{a}^{k+1}_{-j}$ is the action profile selected by $\vec{\sigma}_{-j}$. To prove that σ_0 is winning, consider a strategy σ_1 for Player 1 and the infinite path $\pi' = \pi((\sigma_0, \sigma_1))$ generated by (σ_0, σ_1). It is not hard to see that the sequence π'_{odd} of odd positions in π' belongs to a path π in $\mathcal{G}_{\text{PAR}}$ and it is compatible with $\vec{\sigma}_{-j}$. Thus, since $\vec{\sigma}_{-j}$ is a punishment strategy, π'_{odd} does not satisfy α_j. Moreover, observe that the parity of the sequence π'_{even} of even positions equals that of π'_{odd}. Thus, we have that $\text{Inf}(\alpha(\pi')) = \text{Inf}(\alpha(\pi'_{\mathsf{odd}})) \cup \text{Inf}(\alpha(\pi'_{\mathsf{even}})) = \text{Inf}(\alpha_j(\pi)) + 1$ and so π' is winning for player 0 in $\mathcal{G}^j_{\text{PAR}}$ and σ_0 is a winning strategy.

[2]We omit the labelling function λ to avoid clutter, since it is not used here.

From right to left, let $s \in \mathrm{St} \cap \mathrm{Win}_0(\mathcal{G}^j_{\mathrm{PAR}})$ and let σ_0 be a winning strategy for Player 0 in $\mathcal{G}^j_{\mathrm{PAR}}$, and assume σ_0 is memoryless. Now, for every player i, with $i \neq j$, define the memoryless strategy σ_i in $\mathcal{G}_{\mathrm{PAR}}$ such that, for every $s' \in \mathrm{St}$, if $\sigma_0(s') = (s', \vec{a}_{-j})$, then $\sigma_i(s') = (\vec{a}_{-j})_i$ [3], *i.e.*, the action that player i takes in σ_0 at s'. Now, consider the (memoryless) strategy profile $\vec{\sigma}_{-j}$ given by the composition of all strategies σ_i, and consider a play π in $\mathcal{G}_{\mathrm{PAR}}$, starting from s, that is compatible with $\vec{\sigma}_{-j}$. Thus, there exists a play π' in $\mathcal{G}^j_{\mathrm{PAR}}$, compatible with σ_0, such that $\pi = \pi'_{\mathsf{odd}}$. Moreover, since $\pi'_{\mathsf{odd}} = \pi'_{\mathsf{even}}$, we have that $\mathrm{Inf}(\alpha(\pi')) = \mathrm{Inf}(\alpha(\pi'_{\mathsf{odd}})) \cup \mathrm{Inf}(\alpha(\pi'_{\mathsf{even}})) = \mathrm{Inf}(\alpha_j(\pi)) + 1$. Since π' is winning for Player 0, we know that $\pi \not\models \alpha_j$ and so $\vec{\sigma}_{-j}$ is a punishment strategy against Player j in s. □

Definition 28 and Theorem 9 not only make a bridge from the notion of punishment strategy to the notion of winning strategy for two-player zero-sum games, but also provide a way to understand how to compute punishment regions as well as how to synthesise an actual punishment strategy in multi-player parity games. In this way, by computing winning regions and winning strategies in these games we can solve the *synthesis* problem for individual players in the original game with LTL goals, one of the problems we are interested in. Thus, from Definition 28 and Theorem 9, we have the following corollary.

Corollary 10. Computing $\mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$ can be done in polynomial time with respect to the size of the underlying graph of the game $\mathcal{G}_{\mathrm{PAR}}$ and exponential in the size of the priority function α_j, that is, to the size of the range of α_j. Moreover, there is a memoryless strategy $\vec{\sigma}_j$ that is a punishment against player j in every state $s \in \mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$.

As described in [Gutierrez et al., 2015a], in any (infinite) run *sustained* by a Nash equilibrium $\vec{\sigma}$ in deterministic and pure strategies, that is, in $\pi(\vec{\sigma})$, it is the case that all players that do not get their goals achieved in $\pi(\vec{\sigma})$ can deviate from such a (Nash equilibrium) run only to states where they can be punished by the coalition consisting of all other players in the game. To formalise this idea in the present setting, we need one more concept about punishments, defined next.

Definition 29. An action profile run $\eta = \vec{a}_0, \vec{a}_1, \ldots \in \vec{\mathrm{Ac}}^{\omega}$ is *punishing-secure* in s for player j if, for all $k \in \mathbb{N}$ and a'_j, it holds that $\mathsf{tr}(\pi_j, ((\vec{a}_k)_{-j}, a'_j)) \in \mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$, where π is the only play in $\mathcal{G}_{\mathrm{PAR}}$ starting from s and generated by η.

[3] By an abuse of notation, we let $\sigma_i(s')$ be the value of $\tau_i(s')$.

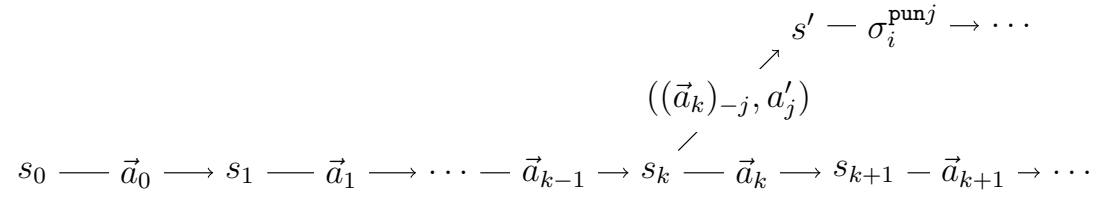

Figure 4.2: Representation of the strategy σ_i. At the beginning of the execution, player i starts following the transducer T_η that generates the action profile run η. The strategy adheres to it until a unilateral deviation from player j occurs, here represented at the k-th step of the play. Once the deviation has occurred, and the game entered a state s', player i starts executing the strategy $\sigma_i^{\text{pun}j}$, to employ the punishment strategy against the deviating player j.

Using the above definition, we can characterise the set of Nash equilibria of a given game. Since strategies are formalised as transducers, *i.e.*, as finite state machines with output, such Nash equilibria strategy profiles produce a set of runs which contains *ultimately periodic* runs, that is, runs which are the concatenation of a finite prefix with an infinite suffix consisting of a finite sequence that repeats itself infinitely often[4]. Furthermore, since in every run π there are players who get their goals achieved in π—the players whose minimum priorities that occur infinitely often in π are even—(and therefore do not have an incentive to deviate from π) and players who do not get their goals achieved in π—the players whose minimum priorities that occur infinitely often in π are odd—(and therefore may have an incentive to deviate from π), we will also want to explicitly refer to such players. To do that, the following notation will be useful: Let $W(\mathcal{G}_{\text{PAR}}, \vec{\sigma}) = \{i \in \text{N} : \mathcal{G}_{\text{PAR}}, \vec{\sigma} \models \alpha_i\}$ denote the set of player that get their goals achieved in $\pi(\vec{\sigma})$. We also write $W(\mathcal{G}_{\text{PAR}}, \pi) = \{i \in \text{N} : \mathcal{G}_{\text{PAR}}, \pi \models \alpha_i\}$.

Theorem 11 (Nash Equilibrium Characterisation)**.** For a parity game $\mathcal{G}_{\text{PAR}}$, there is a Nash Equilibrium strategy profile $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\text{PAR}})$ if and only if there is an ultimately periodic action profile run η such that, for all players $j \in L = \text{N} \setminus W(\mathcal{G}_{\text{PAR}}, \pi)$, the run η is punishing-secure against j in state s_0, where π is the unique path generated by η from s_0.

Proof. The proof is by double implication. From left to right, for $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\text{PAR}})$, let η be the ultimately periodic sequence of action profiles generated by $\vec{\sigma}$. Moreover,

[4]There are Büchi-recognisable languages that contain words which are not ultimately periodic. However, every such a language contains an ultimately periodic word. Indeed, given the goals of players expressed in LTL, strategies that produce ultimately periodic runs are sufficient (see [Gutierrez et al., 2015b], Lemma 1).

assume for a contradiction that η is not punishing-secure for some $j \in L$. By the definition of punishment-secure, there is $k \in \mathbb{N}$ and action $a'_j \in \mathrm{Ac}_j$ for player j such that $s' = \mathsf{tr}(\pi_k, ((\vec{a}_k)_{-j}, a'_j) \notin \mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$. Now, consider the strategy σ'_j that follows η up to the $(k-1)$-th step, executes action a'_j on step k to get into state s', and applies a strategy that achieves α_j from that point onwards. Note that such a strategy is guaranteed to exist since $s' \notin \mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$. Therefore, $\mathcal{G}_{\mathrm{PAR}}, (\vec{\sigma}_{-j}, \sigma'_j) \models \alpha_j$ and so σ'_j is a beneficial deviation for player j, a contradiction to $\vec{\sigma}$ being a Nash equilibrium.

From right to left, we need to define a Nash equilibrium $\vec{\sigma}$ assuming only the existence of η. First, recall that η can be generated by a finite transducer $\mathsf{T}_\eta = (Q_\eta, q^0_\eta, \delta_\eta, \tau_\eta)$ where $\delta_\eta : Q_\eta \to Q_\eta$ and $\tau_\eta : Q_\eta \to \vec{\mathrm{Ac}}$. Moreover, for every player i and deviating player j, with $i \neq j$, there is a (memoryless) strategy $\sigma_i^{\mathtt{pun}j}$ to punish player j in every state in $\mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$. By suitably combining the transducer with the punishment strategies, we define the following strategy $\sigma_i = (Q_i, q^0_i, \delta_i, \tau_i)$ for player i where

- $Q_i = \mathrm{St} \times Q_\eta \times (L \cup \{\top\})$ and $q^0_i = (s^0, q^0_\eta, \top)$;
- $\delta_i = Q_i \times \vec{\mathrm{Ac}} \to Q_i$ is such that
  - $\delta_i((s, q, \top), \vec{a}) = (\mathsf{tr}(s, \vec{a}), \delta_\eta(q), \top)$, if $a = \tau_\eta(q)$, and
  - $\delta_i((s, q, \top), \vec{a}) = (\mathsf{tr}(s, \vec{a}), \delta_\eta(q), j)$, if both
    $$a_{-j} = (\tau_\eta(q))_{-j} \text{ and } \vec{a}_j \neq (\tau_\eta(q))_j;$$
- $\tau_i : Q_i \to \mathrm{Ac}_i$ is such that
  - $\tau_i(s, q, \top) = (\tau_\eta(q))_i$, and
  - $\tau_i(s, q, j) = \sigma_i^{\mathtt{pun}j}(s)$.

To understand how strategy σ_i works, observe that its set of internal states is given by the following triple. The first component is a state of the game, remembering the position of the execution. The second component is a state of the transducer T_η, which is used to employ the execution of the action profile run η. The third component is either the symbol $\top$, used to flag that no deviation has occurred, or the name of a losing player j, used to remember that such a player has deviated from η. At the beginning of the play, strategy σ_i starts executing the actions prescribed by the transducer T_η. It sticks to it until some losing player j performs a deviation. In such a case, the third component of the internal state of σ_i switches to remember the deviating player. Moreover, from that point on, it starts executing the punishment

strategy $\sigma_i^{\mathtt{pun}j}$. Now, define σ to be the collection of all σ_i. It remains to prove that $\vec{\sigma}$ is a Nash Equilibrium.

First, observe that since $\vec{\sigma}$ produces exactly η, we have $W(\mathcal{G}_{\text{PAR}}, \vec{\sigma}) = W(\mathcal{G}_{\text{PAR}}, \eta)$, that is, the players that get their goals achieved in $\pi(\vec{\sigma})$ and η are the same. Thus, only players in L could have a beneficial deviation. Now, consider a player $j \in L$ and a strategy σ'_j and let $k \in \mathbb{N}$ be the minimum (first) step where σ'_j produces an outcome that differs from σ_j when executed along with $\vec{\sigma}_{-j}$. We write π' for $\pi((\vec{\sigma}_{-j}, \sigma'_j))$. Thus, we have $\pi_h = \pi'_h$ for all $h \leq k$ and $\pi_{k+1} \neq \pi'_{k+1}$. Hence $\pi'_{k+1} = \mathsf{tr}(\pi'_k, (\eta_k)_{-j}, a'_j) = \mathsf{tr}(\pi_k, (\eta_k)_{-j}, a'_j) \in \text{Pun}_j(\mathcal{G}_{\text{PAR}})$ and $\mathcal{G}_{\text{PAR}}, (\vec{\sigma}_{-j}, \sigma'_j) \not\models \alpha_j$, since σ_{-j} is a punishment strategy from π'_{k+1}. Thus, there is no beneficial deviation for j and $\vec{\sigma}$ is a Nash equilibrium. □

4.4 Finding Nash Equilibria

Theorem 11 allows us to reduce the problem of finding a Nash equilibrium to finding a path in the game satisfying certain properties, which we will show how to check using DPW and DSW automata. To do this, let us fix a given set $W \subseteq \text{N}$ of players in a given game $\mathcal{G}_{\text{PAR}}$, which are assumed to get their goals achieved. Now, due to Theorem 11, we have that an action profile run η corresponds to a Nash equilibrium with W being the set of "winners" in the game if, and only if, the following two properties are satisfied:

- η is punishment-secure for j in s^0, for all $j \in L = \text{N} \setminus W$;
- $\mathcal{G}_{\text{PAR}}, \pi \models \alpha_i$, for every $i \in W$;

where π is, as usual, the path generated by η from s^0.

To check the existence of such η, we have to check these two properties. First, note that, for η to be punishment-secure for every losing player $j \in L$, the game has to remain in the punishment region of each j. This means that an acceptable action profile run needs to generate a path that is, at every step, contained in the intersection $\bigcap_{j \in L} \text{Pun}_j(\mathcal{G}_{\text{PAR}})$. Thus, to find a Nash equilibrium, we can remove all states not in such an intersection. We also need to remove some edges from the game. Indeed, consider a state s and a partial action profile $\vec{a}_{-j}$. It might be the case that $\mathsf{tr}(s, (\vec{a}_{-j}, a'_j)) \notin \text{Pun}_j(\mathcal{G}_{\text{PAR}})$, for some $a'_j \in \text{Ac}_j$. Therefore, an action profile run that executes the partial profile $\vec{a}_{-j}$ over s cannot be punishment-secure, and so all outgoing edges from $(s, \vec{a}_{-j})$, can also be removed. After doing this for every $j \in L$, we obtain $\mathcal{G}^{-L}_{\text{PAR}}$, the game resulting from $\mathcal{G}_{\text{PAR}}$ after the removal of the states and

edges just described. As a consequence, $\mathcal{G}_{\text{PAR}}^{-L}$ has all and only the paths that can be generated by an action profile run that is punishment-secure for every $j \in L$.

The only thing that remains to be done is to check whether there exists a path in $\mathcal{G}_{\text{PAR}}^{-L}$ that satisfies all players in W. To do this, we use DPW and DSW automata. Since players goals are parity conditions, a path satisfying player i is an accepting run of the (one-letter) DPW $\mathcal{A}^i$ where the set of states and transitions are exactly those of $\mathcal{G}_{\text{PAR}}^{-L}$ and the acceptance condition is given by α_i. Then, in order to find a path satisfying the goals of all players in W, we can solve the emptiness problem of the automaton intersection $\bigtimes_{i \in W} \mathcal{A}^i$. To do this, we can see each $\mathcal{A}^i$ as a DSW $\mathcal{S}_i$ in the usual way (parity conditions are a special case of Streett [Kupferman, 2018]). Since Streett automata are closed under conjunctions of Streett conditions, $\bigtimes_{i \in W} \mathcal{A}^i$ translates to a DSW automaton that can be solved in polynomial time [Kupferman, 2018]. Finally, as we fixed W at the *beginning*, all we need to do is to use the procedure just described for each $W \subseteq \text{N}$, if needed (see Algorithm 9), obtaining an *optimal* decision procedure that has only exponential time and polynomial space complexity in $|\text{N}|$, the number of agents in the system.[5]

4.5 Synthesis and Verification

We now show how to solve the synthesis and verification problems using Non-Emptiness. For *synthesis*, the solution is already contained in the proof of Theorem 11. Note that, in the computation of punishing regions, the algorithm builds, for every player i and potential deviator j, a (memoryless) strategy that player i can play in the collective strategy profile $\vec{\sigma}_{-}j$ in order to punish player j, should player j wishes to deviate. If a Nash equilibrium exists, the algorithm also computes an ultimately periodic witness of it, that is, a computation π in G, that, in particular, satisfies the goals of players in W. At this point, using this information, we are able to define a strategy σ_i for each player $i \in \text{N}$ in the game (*i.e.*, including those not in W), as follows: while no deviation occurs, play the action that contributes to generate π, and if a deviation of player j occurs, then play the (memoryless) strategy σ_i^{punj} that is defined in the game to punish player j in case j were to deviate. Notice, in addition, that because of Lemma 2 and Theorem 8, every strategy for player i in the game with parity goals is also a valid strategy for player i in the game with LTL goals, and that such a strategy, being bisimulation-invariant, is also a strategy for every

[5]Some previous techniques, *e.g.* [Bouyer et al., 2015a], to the computation of pure Nash equilibria are not optimal as they have exponential space complexity in the number of players $|\text{N}|$.

possible bisimilar representation of player i. In this way, our technique can also solve the synthesis problem for every player, that is, can compute individual bisimulation-invariant strategies for every player (system component) in the original multi-player game (concurrent system).

For *verification*, one can use a reduction of the following two problems, called E-Nash and A-Nash in [Gutierrez et al., 2015b, Wooldridge et al., 2016, Gutierrez et al., 2017b], to Non-Emptiness.

Given: Game $\mathcal{G}_{\mathsf{LTL}}$, LTL formula φ.
E-Nash: Is it the case that $\pi(\vec{\sigma}) \models \varphi$, for some $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\mathsf{LTL}})$?
A-Nash: Is it the case that $\pi(\vec{\sigma}) \models \varphi$, for all $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\mathsf{LTL}})$?

We write $(\mathcal{G}_{\mathsf{LTL}}, \varphi) \in$ E-Nash to denote that $(\mathcal{G}_{\mathsf{LTL}}, \varphi)$ is an instance of E-Nash, *i.e.*, given a game $\mathcal{G}_{\mathsf{LTL}}$ and a LTL formula φ, the answer to E-Nash problem is a "yes"; and, similarly for A-Nash.

Because we are working on a bisimulation-invariant setting, we can ensure something even stronger: that for any two games $\mathcal{G}_{\mathsf{LTL}}$ and $\mathcal{G}'_{\mathsf{LTL}}$, whose underlying CGSs are $\mathcal{M}$ and $\mathcal{M}'$, respectively, we know that if $\mathcal{M}$ is bisimilar to $\mathcal{M}'$, then $(\mathcal{G}_{\mathsf{LTL}}, \varphi) \in$ E-Nash if and only if $(\mathcal{G}'_{\mathsf{LTL}}, \varphi) \in$ E-Nash, for all LTL formulae φ; and, similarly for A-Nash, as desired.

In order to solve E-Nash and A-Nash via Non-Emptiness, one could use the following result, whose proof is a simple adaptation of the same result for iterated Boolean games [Gutierrez et al., 2015b] and for multi-player games with LTL goals modelled using SRML [Gutierrez et al., 2017b], which was first presented in [Gao et al., 2017].

Proposition 1 ([Gao et al., 2017])**.** Let G be a game and φ be an LTL formula. There is a game H of constant size in G, such that $\text{NE}(H) \neq \emptyset$ if and only if $\exists \vec{\sigma} \in \text{NE}(G)$. $\pi(\vec{\sigma}) \models \varphi$.

However, since we have Algorithm 9 at our disposal, an easier—and more direct—solution can be obtained. To solve E-Nash we can modify line 12 of Algorithm 9 to include the restriction that such an algorithm, which now receives φ as a parameter, returns "Yes" in line 13 if and only if φ is satisfied in some run in the set of Nash equilibrium witnesses. The new line 12 is "**if** $\mathcal{L}(\bigtimes_{i \in W}(\mathcal{S}_i) \times \mathcal{S}_\varphi) \neq \emptyset$", where $\mathcal{S}_\varphi$ is the DSW automaton representing φ. All complexities remain the same; the modified algorithm for E-Nash is denoted as Algorithm 9'. We can then use Algorithm 9' to solve A-Nash, also as described in [Gao et al., 2017]: essentially, we can check

whether Algorithm 9'($\mathcal{G}_{\mathsf{LTL}}, \neg\varphi$) returns "No" in line 16. If it does, then no Nash equilibrium of $\mathcal{G}_{\mathsf{LTL}}$ satisfies $\neg\varphi$, either because no Nash equilibrium exists at all (thus, A-NASH is vacuously true) or because all Nash equilibria of $\mathcal{G}_{\mathsf{LTL}}$ satisfy φ, then solving A-NASH positively. Note that in this case, since A-NASH is solved positively when the algorithm returns "No" in line 16, then no specific Nash equilibrium strategy profile is synthesised, as expected. However, if the algorithm returns "Yes", that is, the case when the answer to A-NASH problem with $(\mathcal{G}_{\mathsf{LTL}}, \varphi)$ instance is negative, then a strategy profile is synthesised from Algorithm 9' which corresponds to a counter-example for $(\mathcal{G}_{\mathsf{LTL}}, \varphi) \in$ A-NASH. It should be easy to see that implementing E-NASH and A-NASH is straightforward from Algorithm 9. Also, as already known, it is also easy to see that Algorithm 9' solves NON-EMPTINESS if and only if $(\mathcal{G}_{\mathsf{LTL}}, \top) \in$ E-NASH.

4.6 The Role of Bisimilarity

One crucial aspect of our approach to rational verification and synthesis is the role of *bisimilarity* [Milner, 1980, Hennessy and Milner, 1985, De Nicola and Vaandrager, 1995, van Glabbeek and Weijland, 1996]. Bisimulation is the most important type of behavioural equivalence relation considered in computer science, and in particular two bisimilar systems will satisfy the same temporal logic properties.

In our setting, it is highly desirable that properties which hold in equilibrium are sustained across all bisimilar systems to $P_1, \ldots, P_n$. That is, that for every (temporal logic) property φ and every system component P_i' modelled as an agent in a multi-player game, if P_i' is bisimilar to $P_i \in \{P_1, \ldots, P_n\}$, then φ is satisfied in equilibrium—that is, on a run induced by some Nash equilibrium of the game—by $P_1, \ldots, P_i, \ldots P_n$ if and only if is also satisfied in equilibrium by $P_1, \ldots, P_i', \ldots, P_n$, the system in which P_i is replaced by P_i', that is, across all bisimilar systems to $P_1, \ldots, P_n$. This property is called *invariance under bisimilarity*. Unfortunately, as shown in [Gutierrez et al., 2015a, Gutierrez et al., 2017a], the satisfaction of temporal logic properties in equilibrium is not invariant under bisimilarity, thus posing a challenge for the modular and compositional reasoning of concurrent systems, since individual system components in a concurrent system cannot be replaced by (behaviourally equivalent) bisimilar ones, while preserving the temporal logic properties that the overall multi-agent system satisfies in equilibrium.

This is also a problem from a synthesis point of view. Indeed, a strategy for a system component P_i may not be a valid strategy for a bisimilar system component P_i'.

As a consequence, the problem of building strategies for individual processes in the concurrent system $P_1, \ldots, P_i, \ldots P_n$ may not, in general, be the same as building strategies for a bisimilar system $P_1, \ldots, P'_i, \ldots P_n$, again, deterring any hope of being able to do modular reasoning on concurrent and multi-agent systems. These problems were first identified in [Gutierrez et al., 2015a] and further studied in [Gutierrez et al., 2017a]. However, no algorithmic solutions to these two problems were presented in either [Gutierrez et al., 2015a] or [Gutierrez et al., 2017a]. Specifically, in this chapter, bisimilarity was exploited in two ways. Firstly, our construction of punishment strategies (used in the characterisation of Nash equilibrium given by Theorem 11) assumes that players have access to the history of choices that other players in the game have made. As shown in [Gutierrez et al., 2017a, Gutierrez et al., 2019a], with a model of strategies where this is not the case, the preservation of Nash equilibria in the game, as well as of temporal logic properties in equilibrium, may not be guaranteed. Secondly, the implementation in EVE guarantees that any two games whose underlying CGSs are bisimilar, and therefore should be regarded as observationally equivalent from a concurrency point of view, will produce the same answers to the rational verification and automated synthesis problems.

It is also worth noting that even though bisimilarity is probably the most widely used behavioural equivalence in concurrency, in the context of multi-agent systems other relations may be preferred, for instance, equivalence relations that take a detailed account of the independent interactions and behaviour of individual components in a multi-agent system. In such a setting, "alternating" relations with natural ATL* characterisations have been studied [Alur et al., 1998a]. Alternating bisimulation is very similar to bisimilarity on labelled transition systems [Milner, 1980, Hennessy and Milner, 1985], only that when defined on CGSs, instead of action profiles (directions) taken as possible transitions, one allows individual player's actions, which must be matched in the bisimulation game. Because of this, it immediately follows that any alternating bisimulation as defined in [Alur et al., 1998a] is also a bisimilarity as defined here. Despite having a different formal definition, a simple observation can be made: Nash equilibria are not preserved by the alternating (bisimulation) equivalence relations in [Alur et al., 1998a] either, which discourages the use of these even stronger equivalence relations for multi-agent systems. In fact, as discussed in [van Benthem, 2002], the "right" notion of equivalence for games (which can be indirectly used as an observationally equivalence between multi-agent systems) and their game theoretic solution concepts is, undoubtedly, an important and interesting topic of debate, which deserves to be investigated further.

4.7 Summary

The technique developed in this chapter, and its associated implementation (Chapter 7), considers games with LTL goals, deterministic and pure strategies, and dichotomous preferences. In particular, strategies in these games are assumed to be able to see all players' actions, leading to a setting where Nash equilibrium is invariant under bisimilarity [Gutierrez et al., 2017a]. This invariance property, in turn, enables the use of standard verification techniques for temporal logics when reasoning about (Nash) equilibria. In addition, the games are concurrent and synchronous (at each round all players make their choices independently and at the same time), with perfect information, and represented using the Simple Reactive Modules Language (SRML [van der Hoek et al., 2005]). We do not consider mixed or nondeterministic strategies, or goals given by branching-time formulae. We also do not allow for quantitative or probabilistic systems, *e.g.*, such as stochastic games or similar game models. We note, however, that some of these aspects of our reasoning framework have been placed to avoid undesirable computational properties. For instance, it is known that checking for the existence of a Nash equilibrium in multi-player games like the ones we consider is an undecidable problem if either imperfect information or (various kinds of) quantitative/probabilistic information is allowed [Gutierrez et al., 2018b, Ummels and Wojtczak, 2011].

Chapter 5

Some Tractable Cases of Rational Verification

In this chapter we show that the complexity of rational verification can be greatly reduced by restricting specifications to GR(1) [Bloem et al., 2012]. While some simpler goals, like safety and Büchi objectives give rise to tractable problems—P-complete for determining the existence of a Nash equilibrium [McNaughton, 1993, Bouyer et al., 2015a]—GR(1) objective is a relatively more interesting specification language, since it is a fragment of LTL that can represent most response properties of reactive systems. In what follows, we also provide improved complexity results for rational verification when considering players' goals given by mean-payoff utility functions—arguably the most widely used quantitative objective for agents in concurrent and multiagent systems. In particular, we show that for a number of relevant settings, rational verification can be done in polynomial space or even in polynomial time. Part of this chapter appeared in the proceedings of IJCAI'19 [Gutierrez et al., 2019d].

5.1 Preliminaries

General Reactivity of rank 1. The language of *General Reactivity of rank 1*, denoted GR(1), is the fragment of LTL of formulae written in the following form [Bloem et al., 2012]:

$$(\mathbf{GF}\psi_1 \wedge \ldots \wedge \mathbf{GF}\psi_m) \rightarrow (\mathbf{GF}\varphi_1 \wedge \ldots \wedge \mathbf{GF}\varphi_n),$$

where each subformula ψ_i and φ_i is a Boolean combination of atomic propositions.

Mean-Payoff value. For an infinite sequence $r \in \mathbb{R}^\omega$ of real numbers, let $\mathsf{mp}(r)$ be the *mean-payoff* value of r, that is,

$$\mathsf{mp}(r) = \lim \inf_{n\to\infty} \mathsf{avg}_n(r)$$

where, for $n \in \mathbb{N}$, we define $\mathsf{avg}_n(r) = \frac{1}{n}\sum_{j=0}^{n-1} r_j$.

We consider multi-player games with $\mathsf{GR}(1)$ and mp goals. A multi-player $\mathsf{GR}(1)$ game is a tuple $\mathcal{G}_{\mathsf{GR}(1)} = (\mathcal{M}, (\gamma_i)_{i\in\mathrm{N}})$ where $\mathcal{M}$ is a CGS and γ_i is the $\mathsf{GR}(1)$ goal for player i. A multi-player mp game is a tuple $\mathcal{G}_{\mathsf{mp}} = (\mathcal{M}, (\mathsf{w}_i)_{i\in\mathrm{N}})$, where $\mathcal{M}$ is a CGS and $\mathsf{w}_i : \mathrm{St} \to \mathbb{Z}$ is a function mapping every state of the CGS into an integer number. When it is clear from the context, we refer to a multi-player $\mathsf{GR}(1)$ or mp game as a *game* and denote it by $\mathcal{G}$. In any game with CGS $\mathcal{M}$, a path π in A induces a sequence $\lambda(\pi) = \lambda(s_0)\lambda(s_1)\cdots$ of sets of atomic propositions; if, in addition, $\mathcal{M}$ is the CGS of an mp game, then, for each player i, the sequence $\mathsf{w}_i(\pi) = \mathsf{w}_i(s_0)\mathsf{w}_i(s_1)\cdots$ of weights is also induced.

For a $\mathsf{GR}(1)$ game and a path π in it, the payoff of a player i is $\mathsf{pay}_i(\pi) = 1$ if $\lambda(\pi) \models \gamma_i$ and $\mathsf{pay}_i(\pi) = 0$ otherwise. Regarding an mp game, the payoff of player i is $\mathsf{pay}_i(\pi) = \mathsf{mp}(\mathsf{w}_i(\pi))$. Moreover, for a $\mathsf{GR}(1)$ game and a path π, by $W(\pi) = \{i \in \mathrm{N} : \pi \models \gamma_i\}$ and $L(\pi) = \{j \in \mathrm{N} : \pi \not\models \gamma_j\}$ we denote the set of *winners* and *losers*, respectively, over π, that is, the set of players that get their goal satisfied and not satisfied, respectively, over π. With an abuse of notation, we sometime denote $W(\vec{\sigma}, s) = W(\pi(\vec{\sigma}, s))$ and $L(\vec{\sigma}, s) = L(\pi(\vec{\sigma}, s))$, respectively, the set of winners and losers over the path generated by strategy profile $\vec{\sigma}$ when starting the game from s. Furthermore, we simply write $\pi(\vec{\sigma})$ for $\pi(\vec{\sigma}, s_0)$.

Nash equilibrium. Using the payoff functions defined above, we define the concept of Nash equilibrium [Osborne and Rubinstein, 1994] as usual. For a game $\mathcal{G}$, a strategy profile $\vec{\sigma}$ is a *Nash equilibrium* of $\mathcal{G}$ if, for every player i and strategy $\sigma_i' \in \Sigma_i$, we have

$$\mathsf{pay}_i(\pi(\vec{\sigma})) \geq \mathsf{pay}_i(\pi((\vec{\sigma}_{-i}, \sigma_i'))) \ .$$

5.2 Decision Problems

In rational verification, a key question is E-Nash, which is concerned with the existence of a Nash equilibrium that fulfils a given temporal specification φ. This problem can be instantiated in many ways. For instance, in [Gutierrez et al., 2015b], E-Nash was investigated over *iterated Boolean Games* with specifications and players' goals

in LTL, and was proved to be 2EXPTIME-complete. Iterated Boolean games is a very natural framework, but it is computationally intractable.

Motivated by this computational limitation, in this chapter, we study E-Nash for a number of relevant instantiations of the problem, which we show to have better computational complexity. In particular, we study cases where

- Specifications φ are LTL and players' goals are GR(1);
- Specifications φ are LTL and players have mp goals;
- Both the specification φ and the goals are GR(1);
- Specifications φ are GR(1) and players have mp goals.

5.3 Games of General Reactivity of Rank 1

As indicated before, we solve GR(1) games in two cases: the first one is when the specification formula is expressed in LTL, while the goals are in GR(1); the second one when the specification formula as well as the goals belong to GR(1). First, we provide a general result about a characterisation of Nash Equilibrium for GR(1) given in terms of punishments. We first require some notation.

For a GR(1) game $\mathcal{G}$, player $j \in \mathrm{N}$, and state $s \in \mathrm{St}$, the strategy profile $\vec{\sigma}_{-j}$ is *punishing* for player j in s if $\pi((\vec{\sigma}_{-j}, \sigma'_j), s) \not\models \gamma_j$, for every possible strategy σ'_j of player j. We say that a state s is punishing for j if there exists a punishing strategy profile for j on s. Moreover, we denote by $\mathrm{Pun}_j(\mathcal{G})$ the set of punishing states in $\mathcal{G}$. A pair $(s, \vec{\mathsf{a}}) \in \mathrm{St} \times \vec{\mathrm{Ac}}$ is *punishing-secure* for player j, if $\mathsf{tr}(s, (\vec{\mathsf{a}}_{-j}, \mathsf{a}'_j)) \in \mathrm{Pun}_j(\mathcal{G})$ for every action a'_j.

Theorem 12. In a given GR(1) game $\mathcal{G}$, there exists a Nash Equilibrium if and only if there exists an ultimately periodic path π such that, for every $k \in \mathbb{N}$, the pair $(s_k, \vec{\mathsf{a}}^k)$ of the k-th iteration of π is punishing secure for every $j \in L(\pi)$.

Proof. From left to right, let $\vec{\sigma} \in \mathrm{NE}(\mathcal{G})$ and π be the ultimately periodic path generated by $\vec{\sigma}$. Assume by contradiction that π is not punishing secure for some $j \in \mathrm{N}$, that is, there is $k \in \mathbb{N}$ and action a'_j such that $\mathsf{tr}(s_k, (\vec{\mathsf{a}}_{-j}, \mathsf{a}'_j)^k) \notin \mathrm{Pun}_j(\mathcal{G})$. Thus, j can deviate at s_k and satisfy γ_j, which is a contradiction to $\vec{\sigma}$ being a Nash equilibrium. From right to left, recall that π can be generated by a finite transducer, say T. Moreover, for every losing player j, there is a punishing strategy profile for j in every $s \in \mathrm{Pun}_j(\mathcal{G})$. Combining T with such punishment strategies, we build a

profile $\vec{\sigma}$ that follows the actions prescribed by T, until a losing player j deviates. In such a case, $\vec{\sigma}$ would start punishing player j. Observe that $\mathsf{GR}(1)$ objectives are prefix-independent, which is not true for general LTL objectives. That means that the punishment from the k-th iteration takes effect no matter what prefix $\pi_{\leq k}$ has been played so far. Thus, there is no beneficial deviation for j and $\vec{\sigma}$ is a Nash equilibrium. □

At this point, solving E-Nash can be done as follows:

1. Guess a set $W \subseteq \mathrm{N}$ of winners;
2. For each player $j \in L = \mathrm{N} \setminus W$, a loser in the game, compute its punishment region $\mathrm{Pun}_j(\mathcal{G})$;
3. Remove from $\mathcal{G}$ the states that are not punishing for players $j \in L$ and the edges (s, s') that are labelled with an action profile $\vec{\mathsf{a}}$ such that $(s, \vec{\mathsf{a}})$ is not punishing-secure for some $j \in L$, thus obtaining a game $\mathcal{G}^{-L}$;
4. Check whether there exists an ultimately periodic path π in $\mathcal{G}^{-L}$ such that $\pi \models \varphi \wedge \bigwedge_{i \in W} \gamma_i$ holds.

The four steps described in the above procedure yield Algorithm 10, which solves the problem at hand.

Algorithm 10 E-Nash of $\mathsf{GR}(1)$ games.

```
1: input: A game G_GR(1) and a specification formula φ.
2: for i ∈ N do
3:     Compute Pun_i(G)
4: end for
5: for W ⊆ N do
6:     Compute G^{-L}
7:     if π ⊨ (φ ∧ ⋀_{i∈W} γ_i) for some π ∈ G^{-L} then
8:         return Accept
9:     end if
10: end for
11: return Reject
```

While line 7 requires solving the model checking problem for an LTL formula, which can be done in polynomial space, line 6 can be done in polynomial time. Line 5, on the other hand, makes the procedure run in exponential time in the number of players, but still in polynomial space. We then only need to check line 3: this step can be done in polynomial time, as we now show.

Theorem 13. For a given $\mathsf{GR}(1)$ game $\mathcal{G}$ over the CGS $\mathcal{M} = (\mathrm{N}, \mathrm{Ac}, \mathrm{St}, s_0, \mathsf{tr}, \lambda)$ and a player $j \in \mathrm{N}$, computing the winning region $\mathrm{Pun}_j(\mathcal{G})$ of player j can be done in polynomial time with respect to the size of both $\mathcal{G}$ and γ_j.

Proof. We reduce the problem to computing the winning region of a suitably defined Streett game of index $k = 1$, whose complexity is known to be $O(mn^{k+1}kk!)$ [Piterman and Pnueli, 2006]. Given that in our case we have $k = 1$, we obtain a polynomial time algorithm.

Recall that the goal of player j is of the form:

$$\gamma_j = \bigwedge_{l=1}^{m_j} \mathbf{GF}\psi_l^j \rightarrow \bigwedge_{r=1}^{n_j} \mathbf{GF}\theta_r^j,$$

where ψ_l^j's and θ_r^j's are boolean combinations of atomic propositions. Then, consider the CGS $\mathcal{M}' = (\mathrm{N}, \mathrm{Ac}, \mathrm{St}', s_0', \mathsf{tr}')$ [1] where

- $\mathrm{St}' = \mathrm{St} \times \{0, \ldots, m_j\} \times \{0, \ldots, n_j\}$;
- $s_0' = (s_0, 0, 0)$;
- $\mathsf{tr}'((s, \iota_1, \iota_2), \vec{\mathsf{a}}) = (\mathsf{tr}(s, \vec{\mathsf{a}}), \iota_1', \iota_2')$ where

$$\iota_1' = \begin{cases} 1, & \text{if } \iota_1 = 0 \\ \iota_1, & \text{if } \iota_1 \neq 0 \text{ and } s \not\models \psi_{\iota_1}^j, \text{ and} \\ (\iota_1 \oplus_{(m_j+1)} 1), & \text{otherwise} \end{cases}$$

$$\iota_2' = \begin{cases} 1, & \text{if } \iota_2 = 0 \\ \iota_2, & \text{if } \iota_2 \neq 0 \text{ and } s \not\models \theta_{\iota_2}^j \text{ [2]} \\ (\iota_2 \oplus_{(n_j+1)} 1), & \text{otherwise} \end{cases}$$

Intuitively, CGS $\mathcal{M}'$ mimics the behaviour of $\mathcal{M}$ and carries two indexes, ι_1 and ι_2. Index ι_1 is increased by one every time the path visits a state that satisfies $\psi_{\iota_1}^j$ and resets to 0 every time the path visits a state that satisfies $\psi_{m_j}^j$. Clearly, ι_1 is reset infinitely many times if and only if the path satisfies every ψ_l^j infinitely many times, and so if and only if it satisfies the temporal specification $\bigwedge_{l=1}^{m_j} \mathbf{GF}\psi_l^j$. The same argument applies to index ι_2, but with respect to the boolean combinations θ_r^j's.

Now, consider the sets $C_j = \mathrm{St} \times \{0\} \times \{0, \ldots, n_j\}$ and $E_j = \mathrm{St} \times \{0, \ldots, m_j\} \times \{0\}$. Clearly, the Streett pair (C_j, E_j) is satisfied by all and only the paths in $\mathcal{M}'$ that satisfy γ_j. Therefore, the winning region of γ_j can be computed as the winning set of

[1] We omit the definition of labelling function, as not needed here.

[2] By $\oplus_k$ we denote the addition modulo k.

the Streett game of index 1 with (C_j, E_j) being the only Streett pair. As this can be done in polynomial time, we proved the statement. □

Based on Theorem 13, we have the following result.

Corollary 14. The E-NASH problem for GR(1) games with an LTL specification is PSPACE-complete.

Proof. The upper-bound follows from the procedure described above. Regarding the lower-bound, note that model-checking an LTL formula φ against a Kripke structure $\mathcal{KS}$ can be easily encoded as an instance of E-NASH where $\mathcal{G}$ is played over a Kripke structure $\mathcal{KS}$, taken to be its CGS, players' goals being tautologies, and the specification being $\neg\varphi$. In such a case, we have that $\mathcal{KS} \models \varphi$ if and only if E-NASH for the pair $(\mathcal{G}, \varphi)$ has a negative answer. □

Corollary 14 sharply contrasts with the same result in case the goals of the players are general LTL formulae. In this more general case, E-NASH is 2EXPTIME-complete.

The special case of GR(1) specifications. One of hardest parts of Algorithm 10 is line 7, where an LTL model checking problem has to be solved, making the running time of the whole procedure exponential in the size of the specification and goals of the players. As we show in this section, a way to drastically reduce the complexity of our decision procedure is to let the specification be in GR(1) too. In such a case, the LTL model checking procedure in line 7 of Algorithm 10 can be avoided, leading to a much simpler construction, which runs in polynomial time for every fixed number of players. In this section, we provide precisely such a simpler construction.

Recall that every GR(1) specification φ can be regarded as a Streett condition of index 1 over an CGS $\mathcal{M}'$ suitably constructed from the original CGS $\mathcal{M}$. Thus, by denoting (C_φ, E_φ) and (C_i, E_i) the Streett pairs corresponding to the GR(1) conditions φ and γ_i, respectively, the problem of finding a path in $\mathcal{M}'$ satisfying the formula $\varphi \wedge \bigwedge_{i \in W} \gamma_i$ amounts to deciding the emptiness of the Streett automaton $\mathcal{S} = (\vec{\mathrm{Ac}}, \mathrm{St}', s'_0, \mathsf{tr}, \mathcal{F})$ where $\mathcal{F} = \{(C_\varphi, E_\varphi), (C_{\gamma_i}, E_{\gamma_i})_{i \in W}\}$.

Note that the size of $\mathcal{M}'$ is polynomial in the size of the GR(1) formulae involved, polynomial in the number of states and actions in the original CGS $\mathcal{M}$, and exponential in the number of players. More specifically, we have that $|\mathrm{St}'| = |\mathrm{St}| \cdot |\gamma|^{|\mathrm{N}|}$ and so the number of edges is at most $|\mathrm{St}'|^2$. Moreover, the emptiness problem of a deterministic Streett word automaton can be solved in time that is polynomial in the

automaton's index and its number of states and transitions [Rauch Henzinger and Telle, 1996, Kupferman, 2018].

The complexity of the E-NASH problem takes $2^{|\mathrm{N}|}$ times a procedure for computing at most $|N|$ punishing regions (that is polynomial in the size of both $\mathcal{G}$ and $\varphi, \gamma_1, \ldots, \gamma_N$) plus the complexity of the emptiness problem for a Streett automaton whose size is polynomial in $\mathcal{G}$ $\varphi, \gamma_1, \ldots, \gamma_N$, and exponential in the number of players.

Based on the constructions described above, we can show the following (fixed-parameter tractable) complexity result.

Theorem 15. For a given $\mathsf{GR}(1)$ game $\mathcal{G}$ and a $\mathsf{GR}(1)$ formula φ, the E-NASH problem can be solved in time that is polynomial in $|\mathrm{St}|$, $|\mathrm{Ac}|$, and $|\varphi|$, $|\gamma_1|, \ldots, |\gamma_N|$ and exponential in the number of players $|\mathrm{N}|$. Therefore, the problem is fixed-parameter tractable, parametrized in the number of players.

5.4 Mean-Payoff Games

We now focus on multi-player mean-payoff (mp) games. As in the previous case, we first characterise the Nash Equilibria of a game in terms of punishments and then reduce E-NASH to a suitable path-finding problem in the underlying CGS. To do this, we first need to recall the notion of secure values for mean-payoff games [Ummels and Wojtczak, 2011].

For a player i and a state $s \in \mathrm{St}$, by $\mathtt{pun}_i(s)$ we denote the punishment value of i over s, that is, the maximum payoff that i can achieve from s, when all other players behave adversarially. Such a value can be computed by considering the corresponding two-player zero-sum mean-payoff game [Zwick and Paterson, 1996]. Thus, it is in $\mathsf{NP} \cap \mathsf{coNP}$, and note that both player i and coalition $\mathrm{N} \setminus \{i\}$ can achieve the optimal value of the game using memoryless strategies.

For a player i and a value $z \in \mathbb{R}$, a pair $(s, \vec{\mathsf{a}})$ is z-secure for i if $\mathtt{pun}_i(\mathsf{tr}(s, (\vec{\mathsf{a}}_{-i}, \mathsf{a}'_i))) \leq z$ for every $\mathsf{a}'_i \in \mathrm{Ac}$.

Theorem 16. For every mp game $\mathcal{G}$ and ultimately periodic path $\pi = (s_0, \vec{\mathsf{a}}_0), (s_1, \vec{\mathsf{a}}^1), \ldots$, the following are equivalent

1. There is $\vec{\sigma} \in \mathrm{NE}(\mathcal{G})$ such that $\pi = \pi(\vec{\sigma}, s_0)$;
2. There exists $z \in \mathbb{R}^{\mathrm{N}}$, where $z_i \in \{\mathtt{pun}_i(s) : s \in \mathrm{St}\}$ such that, for every $i \in \mathrm{N}$
   (a) for all $k \in \mathbb{N}$, the pair $(s_k, \vec{\mathsf{a}}^k)$ is z_i-secure for i, and

(b) $z_i \leq \mathsf{pay}_i(\pi)$.

Proof. For (1) implies (2): Let z_i be the largest value player i can get by deviating from π. Let $k \in \mathbb{N}$ be such that $z_i = \mathtt{pun}_i(\mathsf{tr}(s_k, (\vec{\mathsf{a}}_{-i}, \mathsf{a}'_i)))$. Suppose further that $\mathsf{pay}_i(\pi) < z_i$. Thus, player i would deviate at s_k, which is a contradiction to π being a path induced by a Nash equilibrium.

For (2) imples (1): Define strategy profile $\vec{\sigma}$ that follows π as long as no-one has deviated from π. In such a case where player i deviates on the k-th iteration, the strategy profile $\vec{\sigma}_{-i}$ starts playing the z_i-secure strategy for player i that guarantees the payoff of player i to be less than z_i. Therefore, we have $\mathsf{pay}_i(\pi(\vec{\sigma}_{-i}, \sigma'_i)) \leq z_i \leq \mathsf{pay}_i(\pi)$, for every possible strategy σ'_i of player i (the second inequality is due to condition 2(b)). Thus, there is no beneficial deviation for player i and π is a path induced by a Nash equilibrium. □

The characterization of Nash Equilibria provided in Theorem 16 allows us to turn the E-Nash problem for mp games into a path finding problem over $\mathcal{G}$. Similarly to the case of GR(1) games, we have the following procedure.

1. For every $i \in \mathrm{N}$ and $s \in \mathrm{St}$, compute the value $\mathtt{pun}_i(s)$;
2. Guess a vector $z \in \mathbb{R}^{\mathrm{N}}$ of values, each of them being a punishment value for a player i;
3. Compute the game $\mathcal{G}[z]$ by removing the states s such that $\mathtt{pun}_i(s) \leq z_i$ for some player i and the transitions $(s, \vec{\mathsf{a}})$ that are not z_i secure for some player i;
4. Find an ultimately periodic path π in game $\mathcal{G}[z]$ such that $\pi \models \varphi$ and $z_i \leq \mathsf{pay}_i(\pi)$ for every player $i \in \mathrm{N}$.

Step 1 can be done in NP for every pair (i, s), step 2 can be done in exponential time and polynomial space in the number of z-secure values, and step 3 can be done in polynomial time, similar to the case of GR(1) games. Regarding the last step, its complexity depends on the specification language. For the case of φ being an LTL formula, consider the formula

$$\varphi_{\text{E-Nash}} := \varphi \wedge \bigwedge_{i \in \mathrm{N}} (\mathsf{mp}(i) \geq z_i),$$

written in the language $\mathsf{LTL}^{\mathsf{Lim}}$, an extension of LTL where statements about mean-payoff values over a given weighted CGS can be made [Boker et al., 2014]. Observe

that formula $\varphi_{\text{E-NASH}}$ corresponds exactly to requirement 2(b) in Theorem 16. Moreover, since every path in $\mathcal{G}[z]$ satisfies condition 2(a) by construction, every path that satisfies $\varphi_{\text{E-NASH}}$ is a solution of the E-NASH problem and viceversa. We can solve the latter problem by model checking the formula against the CGS underlying $\mathcal{G}[z]$. Since this can be done in PSPACE [Boker et al., 2014], we have the following result.

Corollary 17. The E-NASH problem for mp games with an LTL specification formula φ is PSPACE-complete.

Proof. The upper-bound follows from Theorem 16. The lower-bound follows from the fact that LTL model checking is a special case of E-NASH, for instance, in which all weights for all players are set to the same value, say 0. □

As for the case of GR(1) games, we can summarize the procedure in the following algorithm.

Algorithm 11 E-NASH of mp games.

Input: A game $\mathcal{G}_{\mathsf{mp}}$ and a specification formula φ.
for $i \in$ N and $s \in$ St **do**
    Compute $\mathtt{pun}_i(\mathcal{G})$
end for
for $\vec{z} \in \{\mathtt{pun}_i(s) : s \in \text{St}\}^{\text{N}}$ **do**
    Compute $\mathcal{G}[z]$
    if $\pi \models \varphi_{\text{E-NASH}}$ for some $\pi \in \mathcal{G}[z]$ **then**
        return Accept
    end if
end for
return Reject

The special case of GR(1) specifications. As in the case of GR(1) games, here we show that restricting the specification language to GR(1) lowers the complexity also for mp games. The reason is that the path finding problem for GR(1) specifications can be done while avoiding model-checking of an $\mathsf{LTL}^{\mathsf{Lim}}$ formula. In order to do this, we follow a different approach. Using am mp game $\mathcal{G}$ and a GR(1) specification φ we define a linear program such that the linear program has a solution if and only if the pair $(\mathcal{G}, \varphi)$ is an instance of E-NASH. In particular, this approach is similar to the technique used in [Gutierrez et al., 2017c, Theorem 2], where Linear Programming is used to find the complexity of solving a variant of E-NASH. Formally, we have the following result.

Theorem 18. The E-NASH problem for mp games with a GR(1) specification φ is NP-complete.

Proof. We will define a linear program of size polynomial in $\mathcal{G}$ having a solution if and only if there exists an ultimately periodic path whose payoff for every player i is at least a minimum threshold z_i and satisfies the GR(1) specification.

In order to do that, first recall that φ has the following form

$$\varphi = \bigwedge_{l=1}^{m} \mathbf{GF}\psi_l \rightarrow \bigwedge_{r=1}^{n} \mathbf{GF}\theta_r,$$

and let $V(\psi_l)$ and $V(\theta_r)$ be the subset of states in $\mathcal{G}$ that satisfy the boolean combinations ψ_l and θ_r, respectively. Observe that property φ is satisfied over a path π if, and only if, either π visits every $V(\theta_r)$ infinitely many times or visits some of the $V(\psi_l)$ only a finite number of times.

For the game $\mathcal{G}[z]$, let (V, E) be the underlying graph, and for every edge $e \in E$ introduce a variable x_e. Informally, the value x_e is the number of times that the edge e is used on a cycle. Formally, let $\mathsf{src}(e) = \{v \in V : \exists w\, e = (v, w) \in E\}$; $\mathsf{trg}(e) = \{v \in V : \exists w\, e = (w, v) \in E\}$; $\mathsf{out}(v) = \{e \in E : \mathsf{src}(e) = v\}$; and $\mathsf{in}(v) = \{e \in E : \mathsf{trg}(e) = v\}$.

Consider ψ_l for some $1 \leq l \leq m$, and define the linear program $\mathsf{LP}(\psi_l)$ with the following inequalities and equations:

Eq1: $x_e \geq 0$ for each edge e — a basic consistency criterion;

Eq2: $\Sigma_{e \in E} x_e \geq 1$ — ensures that at least one edge is chosen;

Eq3: for each $a \in A$, $\Sigma_{e \in E} \mathsf{w}_a(\mathsf{src}(e)) x_e \geq 0$ — this enforces that the total sum of any solution is positive;

Eq4: $\Sigma_{\mathsf{src}(e) \cap V(\psi_l) \neq \emptyset} x_e = 0$ — this ensures that no state in $V(\psi_l)$ is in the cycle associated with the solution;

Eq5: for each $v \in V$, $\Sigma_{e \in \mathsf{out}(v)} x_e = \Sigma_{e \in \mathsf{in}(v)} x_e$ — this condition says that the number of times one enters a vertex is equal to the number of times one leaves that vertex.

By construction, it follows that $\mathsf{LP}(\psi_l)$ admits a solution if and only if there exists a path π in $\mathcal{G}$ such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i and visits $V(\psi_l)$ only *finitely many times*. In addition, consider the linear program $\mathsf{LP}(\theta_1, \ldots, \theta_n)$ defined with the following inequalities and equations:

Eq1: $x_e \geq 0$ for each edge e — a basic consistency criterion;

Eq2: $\Sigma_{e \in E} x_e \geq 1$ — ensures that at least one edge is chosen;

Eq3: for each $a \in A$, $\Sigma_{e \in E} \mathsf{w}_a(\mathsf{src}(e)) x_e \geq 0$ — this enforces that the total sum of any solution is positive;

Eq4: for all $1 \leq r \leq n$, $\Sigma_{\mathsf{src}(e) \cap V(\theta_r) \neq \emptyset} x_e \geq 1$ — this ensures that for every $V(\theta_r)$ at least one state is in the cycle;

Eq5: for each $v \in V$, $\Sigma_{e \in \mathsf{out}(v)} x_e = \Sigma_{e \in \mathsf{in}(v)} x_e$ — this condition says that the number of times one enters a vertex is equal to the number of times one leaves that vertex.

In this case, $\mathsf{LP}(\theta_1, \ldots, \theta_n)$ admits a solution if and only if there exists a path π such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i and visits every $V(\theta_r)$ *infinitely many times*.

Since the constructions above are polynomial in the size of both $\mathcal{G}$ and φ, we can conclude it is possible to check in NP the statement that there is a path π satisfying φ such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i in the game if and only if one of the two linear programs defined above has a solution. For the lower bound, we use [Ummels and Wojtczak, 2011] and observe that if φ is true, then the problem is equivalent to checking whether the mp game has a Nash equilibrium. □

5.5 Summary

E-Nash is, arguably, the most fundamental problem in the rational verification framework, but it is not the only one. The two other key problems are A-Nash and Non-emptiness.

We can conclude from (the proofs of) the results presented so far, which are summarised in Table 5.1, that while A-Nash for GR(1) games is also PSPACE and FPT, respectively, in case of LTL and GR(1) specifications, for mp games the problem is, respectively, PSPACE and coNP, in each case. In addition, we can also conclude that whereas Non-emptiness for GR(1) games is FPT, for mp games is NP-complete. These results contrast with those when players' goals are general LTL formulae, where all problems are 2EXPTIME-complete since LTL synthesis, which is 2EXPTIME-hard [Pnueli and Rosner, 1989], can be encoded. These results also contrast with those presented in [Gao et al., 2017], where it is shown that, in succinct

Players' goals	Specification	E-Nash	
LTL	LTL	2EXPTIME-complete	
GR(1)	LTL	PSPACE-complete	(Corollary 14)
GR(1)	GR(1)	FPT	(Theorem 15)
mp	LTL	PSPACE-complete	(Corollary 17)
mp	GR(1)	NP-complete	(Theorem 18)

Table 5.1: Summary of main complexity results.

model representations given by iterated Boolean games or reactive modules, all problems in the rational verification framework can be reduced to Non-emptiness, which clearly cannot be the case here, unless the whole polynomial hierarchy collapses.

Chapter 6

Equilibrium Design

In game theory, *mechanism design* is concerned with the design of incentives so that a desired outcome of the game can be achieved. In this chapter, we study the design of incentives so that a desirable equilibrium is obtained, for instance, an equilibrium satisfying a given temporal logic property—a problem that we call *equilibrium design*. We base our study on a framework where system specifications are represented as temporal logic formulae, games as quantitative concurrent game structures, and players' goals as mean-payoff objectives. In particular, we consider system specifications given by LTL and GR(1) formulae, and show that implementing a mechanism to ensure that a given temporal logic property is satisfied on some/every Nash equilibrium of the game, whenever such a mechanism exists, can be done in PSPACE for LTL properties and in NP/Σ_2^{P} for GR(1) specifications. We also study the complexity of various related decision and optimisation problems, such as optimality and uniqueness of solutions, and show that the complexities of all such problems lie within the polynomial hierarchy. As an application, equilibrium design can be used as an alternative solution to the rational synthesis and verification problems for concurrent games with mean-payoff objectives whenever no solution exists, or as a technique to repair, whenever possible, concurrent games with undesirable rational outcomes (Nash equilibria) in an optimal way. Part of this chapter appeared in the proceedings of CONCUR'19 [Gutierrez et al., 2019c].

6.1 From Mechanism Design to Equilibrium Design

We now describe the two main problems that are our focus of study. As discussed in the introduction, such problems are closely related to the well-known problem of

mechanism design in game theory. Consider a system populated by agents N, where each agent $i \in \mathrm{N}$ wants to maximise its payoff $\mathsf{pay}_i(\cdot)$. As in a mechanism design problem, we assume there is an external *principal* who has a goal φ that it wants the system to satisfy, and to this end, wants to incentivise the agents to act collectively and rationally so as to bring about φ. In our model, incentives are given by *subsidy schemes* and goals by temporal logic formulae.

Subsidy Schemes. A subsidy scheme defines additional imposed rewards over those given by the weight function w. While the weight function w is fixed for any given game, the principal is assumed to be at liberty to define a subsidy scheme as they see fit. Since agents will seek to maximise their overall rewards, the principal can incentivise agents away from performing visiting some states and towards visiting others; if the principal designs the subsidy scheme correctly, the agents are incentivised to choose a strategy profile $\vec{\sigma}$ such that $\pi(\vec{\sigma}) \models \varphi$. Formally, we model a subsidy scheme as a function $\kappa : \mathrm{N} \to \mathrm{St} \to \mathbb{N}$, where the intended interpretation is that $\kappa(i)(s)$ is the subsidy in the form of a natural number $k \in \mathbb{N}$ that would be imposed on player i if such a player visits state $s \in \mathrm{St}$. For instance, if we have $\mathsf{w}_i(s) = 1$ and $\kappa(i)(s) = 2$, then player i gets $1 + 2 = 3$ for visiting such a state. For simplicity, hereafter we write $\kappa_i(s)$ instead of $\kappa(i)(s)$ for the subsidy for player i.

Notice that having an unlimited fund for a subsidy scheme would make some problems trivial, as the principal can always incentivise players to satisfy φ (provided that there is a path in A satisfying φ). A natural and more interesting setting is that the principal is given a constraint in the form of *budget* $\beta \in \mathbb{N}$. The principal then can only spend within the budget limit. To make this clearer, we first define the *cost* of a subsidy scheme κ as follows.

Definition 30. Given a game $\mathcal{G}$ and subsidy scheme κ, we let $\mathsf{cost}(\kappa) = \sum_{i \in \mathrm{N}} \sum_{s \in \mathrm{St}} \kappa_i(s)$.

We say that a subsidy scheme κ is *admissible* if it does not exceed the budget β, that is, if $\mathsf{cost}(\kappa) \leq \beta$. Let $\mathcal{K}(\mathcal{G}, \beta)$ denote the set of admissible subsidy scheme over $\mathcal{G}$ given budget $\beta \in \mathbb{N}$. Thus we know that for each $\kappa \in \mathcal{K}(\mathcal{G}, \beta)$ we have $\mathsf{cost}(\kappa) \leq \beta$. We write $(\mathcal{G}, \kappa)$ to denote the resulting game after the application of subsidy scheme κ on game $\mathcal{G}$. Formally, we define the application of some subsidy scheme on a game as follows.

Definition 31. Given a game $\mathcal{G} = (A, (\mathsf{w}_i)_{i \in \mathrm{N}})$ and an admissible subsidy scheme κ, we define $(\mathcal{G}, \kappa) = (A, (\mathsf{w}'_i)_{i \in \mathrm{N}})$, where $\mathsf{w}'_i(s) = \mathsf{w}_i(s) + \kappa_i(s)$, for each $i \in \mathrm{N}$ and $s \in \mathrm{St}$.

We now come to the main question(s) that we consider in the remainder of the chapter. We ask whether the principal can find a subsidy scheme that will incentivise players to collectively choose a rational outcome (a Nash equilibrium) that satisfies its temporal logic goal φ. We call this problem *equilibrium design*. Following [Wooldridge et al., 2013], we define two variants of this problem, a *weak* and a *strong* implementation of the equilibrium design problem.

- In the Weak Implementation, we are given a game $\mathcal{G}$, a formula φ, and a budget β, and we are asked whether there exists any subsidy scheme κ such that $(\mathcal{G}, \kappa)$ has at least one Nash equilibrium that satisfies φ;
- In the Strong Implementation, we are given a game $\mathcal{G}$, a formula φ, and a budget β, and we are asked whether there exists any subsidy scheme κ such that
  1. $(\mathcal{G}, \kappa)$ has at least one Nash equilibrium, and
  2. all Nash equilibria of $(\mathcal{G}, \kappa)$ satisfy φ.

6.2 Equilibrium Design: Weak Implementation

In this section, we study the weak implementation of the equilibrium design problem, a logic-based computational variant of the principal's mechanism design problem in game theory. We assume that the principal has full knowledge of the game $\mathcal{G}$ under consideration, that is, the principal uses all the information available of $\mathcal{G}$ to find the appropriate subsidy scheme, if such a scheme exists. We now formally define the weak variant of the implementation problem, and study its respective computational complexity, first with respect to goals (specifications) given by LTL formulae and then with respect to $\mathsf{GR}(1)$ formulae.

Let $\mathrm{WI}(\mathcal{G}, \varphi, \beta)$ denote the set of subsidy schemes over $\mathcal{G}$ given budget β that satisfy a formula φ in at least one path π generated by $\vec{\sigma} \in \mathrm{NE}(\mathcal{G})$. Formally

$$\mathrm{WI}(\mathcal{G}, \varphi, \beta) = \{\kappa \in \mathcal{K}(\mathcal{G}, \beta) : \exists \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa) \text{ s.t. } \pi(\vec{\sigma}) \models \varphi\}.$$

Definition 32 (Weak Implementation)**.** Given a game $\mathcal{G}$, formula φ, and budget β:

$$\text{Is it the case that } \mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing?$$

In order to solve Weak Implementation, we first characterise the Nash equilibria of a multi-player concurrent game in terms of punishment strategies. To do this in our setting, we recall the notion of secure values for mean-payoff games [Ummels and Wojtczak, 2011].

For a player i and a state $s \in \mathrm{St}$, by $\mathtt{pun}_i(s)$ we denote the punishment value of i over s, that is, the maximum payoff that i can achieve from s, when all other players behave adversarially. Such a value can be computed by considering the corresponding two-player zero-sum mean-payoff game [Zwick and Paterson, 1996]. Thus, it is in $\mathsf{NP} \cap \mathsf{coNP}$, and note that both player i and coalition $\mathrm{N} \setminus \{i\}$ can achieve the optimal value of the game using *memoryless* strategies. Then, for a player i and a value $z \in \mathbb{R}$, a pair $(s, \vec{\mathsf{a}})$ is z-secure for player i if $\mathtt{pun}_i(\mathsf{tr}(s, (\vec{\mathsf{a}}_{-i}, \mathsf{a}'_i))) \leq z$ for every $\mathsf{a}'_i \in \mathrm{Ac}$. Write $\mathtt{pun}_i(\mathcal{G})$ for the set of punishment values for player i in $\mathcal{G}$. From Chapter 5 Theorem 16, it is the case that for every mp game $\mathcal{G}$ and ultimately periodic path $\pi = (s_0, \vec{\mathsf{a}}_0), (s_1, \vec{\mathsf{a}}^1), \ldots$, the following are equivalent:

1. There is $\vec{\sigma} \in \mathrm{NE}(\mathcal{G})$ such that $\pi = \pi(\vec{\sigma}, s_0)$;
2. There exists $z \in \mathbb{R}^{\mathrm{N}}$, where $z_i \in \{\mathtt{pun}_i(s) : s \in \mathrm{St}\}$ such that, for every $i \in \mathrm{N}$
   (a) for all $k \in \mathbb{N}$, the pair $(s_k, \vec{\mathsf{a}}^k)$ is z_i-secure for i, and
   (b) $z_i \leq \mathsf{pay}_i(\pi)$.

The characterisation of Nash Equilibria provided in Theorem 16 will allow us to turn the Weak Implementation problem into a *path finding* problem over $(\mathcal{G}, \kappa)$. On the other hand, with respect to the budget β that the principal has at its disposal, the definition of subsidy scheme function κ implies that the size of $\mathcal{K}(\mathcal{G}, \beta)$ is of bounded size, and particularly, it is bounded by β and the number of agents and states in the game $\mathcal{G}$, in the following way.

Proposition 2. Given a game $\mathcal{G}$ with $|N|$ players and $|\mathrm{St}|$ states and budget β, it holds that

$$|\mathcal{K}(\mathcal{G}, \beta)| = \frac{\beta + 1}{m} \binom{\beta + m}{\beta + 1},$$

with $m = |N \times \mathrm{St}|$ being the number of pairs of possible agents and states.

Proof. For a fixed budget b, the number of subsidy schemes of budget exactly b corresponds to the number of *weak compositions* of b in m parts, which is given by $\binom{b+m-1}{b}$ [S. Heubach and T. Mansour, 2009]. Therefore, the number of subsidy schemes of budget at most β is the sum

$$|\mathcal{K}(\mathcal{G},\beta)| = \sum_{b=0}^{\beta} \binom{b+m-1}{b}.$$

We now prove that

$$\sum_{b=0}^{\beta} \binom{b+m-1}{b} = \frac{\beta+1}{m}\binom{\beta+m}{\beta+1}.$$

By induction on β, as base case, for $\beta = 0$, we have that

$$\binom{\beta+m-1}{\beta} = 1 = \frac{\beta+1}{m}\binom{\beta+m}{\beta+1}.$$

For the inductive case, let us assume that the assertion hold for some β and let us prove for $\beta+1$. We have the following:

$$\sum_{b=0}^{\beta+1} \binom{b+m-1}{b} = \sum_{b=0}^{\beta} \binom{b+m-1}{b} + \binom{\beta+m-1+1}{\beta+1} = \frac{\beta+1}{m}\binom{\beta+m}{\beta+1} + \binom{\beta+m}{\beta+1}.$$

Therefore we have

$$\frac{\beta+1}{m}\binom{\beta+m}{\beta+1} + \binom{\beta+m}{\beta+1} = \binom{\beta+m}{\beta+1}\left(\frac{\beta+1}{m}+1\right) =$$
$$\binom{\beta+m}{\beta+1}\frac{\beta+1+m}{m} = \frac{\beta+1+m}{m}\cdot\frac{(\beta+m)!}{(\beta+1)!(\cancel{\beta}+m-\cancel{\beta}-1)!} =$$
$$\frac{(\beta+m+1)!}{(\beta+1)!m!} = \frac{(\beta+m+1)!}{(\beta+1)!m!}\cdot\frac{\beta+2}{\beta+2}\cdot\frac{m}{m} = \frac{\beta+2}{m}\cdot\frac{(\beta+m+1)!}{(\beta+2)!(m-1)!} =$$
$$\frac{\beta+2}{m}\cdot\frac{(\beta+m+1)!}{(\beta+2)!(\beta+m+1-\beta-2)!} = \frac{\beta+2}{m}\binom{\beta+m+1}{\beta+2}$$

that proves the assertion. □

From Proposition 2 we derive that the number of possible subsidy schemes is *polynomial* in the budget β and singly *exponential* in both the number of agents and states in the game. At this point, solving Weak Implementation can be done with the following procedure :

1. Guess:
   - a subsidy scheme $\kappa \in \mathcal{K}(\mathcal{G},\beta)$,
   - a state $s \in \mathrm{St}$ for every player $i \in \mathrm{N}$, and
   - punishment memoryless strategies $(\vec{\sigma}_{-1},\ldots,\vec{\sigma}_{-n})$ for all players $i \in \mathrm{N}$;

2. Compute $(\mathcal{G}, \kappa)$;

3. Compute $z \in \mathbb{R}^{\mathrm{N}}$;

4. Compute the game $(\mathcal{G}, \kappa)[z]$ by removing the states s such that $\mathtt{pun}_i(s) \leq z_i$ for some player i and the transitions $(s, \vec{\mathsf{a}}_{-i})$ that are not z_i secure for player i;

5. Check whether there exists an ultimately periodic path π in $(\mathcal{G}, \kappa)[z]$ such that $\pi \models \varphi$ and $z_i \leq \mathsf{pay}_i(\pi)$ for every player $i \in \mathrm{N}$.

The (deterministic) algorithm implementing the procedure above can be summarised as follows.

Algorithm 12 WEAK IMPLEMENTATION.

Input: A game $\mathcal{G}$, a specification formula φ, and budget β.
for $\kappa \in \mathcal{K}(\mathcal{G}, \beta), s \in \mathrm{St}$, and $(\vec{\sigma}_{-1}, \ldots, \vec{\sigma}_{-n}) \in \bigtimes_{j \in \mathrm{N}} (\times_{i \in \mathrm{N} \setminus j} \sigma_i)$ **do**
    Compute $(\mathcal{G}, \kappa)$
    for $i \in \mathrm{N}$ **do**
        Compute $z_i = \mathtt{pun}_i(s)$ using $\vec{\sigma}_{-i}$
    end for
    Compute $(\mathcal{G}, \kappa)[z]$
    if there is $\vec{\sigma} \in \mathrm{NE}((\mathcal{G}, \kappa)[z])$ such that $\pi(\vec{\sigma}) \models \varphi$ **then**
        return Accept
    end if
end for
return Reject

Since the set $\mathcal{K}(\mathcal{G}, \beta)$ is finitely bounded (Proposition 2), and punishment strategies only need to be memoryless, thus also finitely bounded, clearly step 1 can be guessed nondeterministically. Moreover, each of the guessed elements is of polynomial size, thus this step can be done (deterministically) in polynomial space. Step 2 clearly can be done in polynomial time. Step 3 can also be done in polynomial time since, given $(\vec{\sigma}_{-1}, \ldots, \vec{\sigma}_{-n})$, we can compute z solving $|\mathrm{N}|$ one-player mean-payoff games, one for each player i [Zwick and Paterson, 1996, Thm. 6]. For step 5, we will use Theorem 16 and consider two cases, one for LTL specifications and one for GR(1) specifications. Firstly, for LTL specifications, consider the formula

$$\varphi_{\mathrm{WI}} := \varphi \wedge \bigwedge_{i \in \mathrm{N}} (\mathsf{mp}(i) \geq z_i)$$

written in $\mathsf{LTL}^{\mathsf{Lim}}$ [Boker et al., 2014], an extension of LTL where statements about mean-payoff values over a given weighted CGS can be made.[1] The semantics of the temporal operators of $\mathsf{LTL}^{\mathsf{Lim}}$ is just like the one for LTL over infinite computation paths $\pi = s_0, s_1, s_3 \ldots$. On the other hand, the meaning of $\mathsf{mp}(i) \geq z_i$ is simply that such an atomic formula is true if, and only if, the mean-payoff value of π with respect to player i is greater or equal to z_i, a constant real value; that is, $\mathsf{mp}(i) \geq z_i$ is true in π if and only if $\mathsf{pay}_i(\pi) = \mathsf{mp}(\mathsf{w}_i(\pi))$ is greater or equal than constant value z_i. Formula φ_{WI} corresponds exactly to $2(b)$ in Theorem 16. Furthermore, since every path in $(\mathcal{G}, \kappa)[z]$ satisfies condition $2(a)$ of Theorem 16, every computation path of $(\mathcal{G}, \kappa)[z]$ that satisfies φ_{WI} is a witness to the Weak Implementation problem.

Theorem 19. Weak Implementation with LTL specifications is PSPACE-complete.

Proof. Membership follows from the procedure above and the fact that model checking for $\mathsf{LTL}^{\mathsf{Lim}}$ is PSPACE-complete [Boker et al., 2014]. Hardness follows from the fact that LTL model checking is a special case of Weak Implementation. For instance, consider the case in which all weights for all players are set to the same value, say 0, and the principal has budget $\beta = 0$. □

Case with $\mathsf{GR}(1)$ specifications. One of the main bottlenecks of our procedure to solve Weak Implementation lies in step 5, where we solve an $\mathsf{LTL}^{\mathsf{Lim}}$ model checking problem. To reduce the complexity of our decision procedure, we consider Weak Implementation with the specification φ expressed in the $\mathsf{GR}(1)$ sublanguage of LTL. With this specification language, the path finding problem can be solved without model-checking the $\mathsf{LTL}^{\mathsf{Lim}}$ formula given before. In order to do this, we can define a linear program (LP) such that the LP has a solution if and only if $\mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$. From our previous procedure, observe that step 1 can be done nondeterministically in polynomial time, and steps 2–4 can be done (deterministically) in polynomial time. Furthermore, using LP, we also can check step 5 deterministically in polynomial time. For the lower-bound, we use [Ummels and Wojtczak, 2011] and note that if $\varphi = \top$ and $\beta = 0$, then the problem reduces to checking whether the underlying mp game has a Nash equilibrium. Based on the above observations, we have the following result.

Theorem 20. Weak Implementation with $\mathsf{GR}(1)$ specifications is NP-complete.

[1] The formal semantics of $\mathsf{LTL}^{\mathsf{Lim}}$ can be found in [Boker et al., 2014]. We prefer to give only an informal description here.

Proof. We will define a linear program of size polynomial in $(\mathcal{G}, \kappa)$ having a solution if and only if there exists an ultimately periodic path π such that $z_i \leq \mathsf{pay}_i(\pi)$ and satisfies the $\mathsf{GR}(1)$ specification.

Recall that φ has the following form

$$\varphi = \bigwedge_{l=1}^{m} \mathbf{GF}\psi_l \rightarrow \bigwedge_{r=1}^{n} \mathbf{GF}\theta_r,$$

and let $V(\psi_l)$ and $V(\theta_r)$ be the subset of states in $(\mathcal{G}, \kappa)$ that satisfy the boolean combinations ψ_l and θ_r, respectively. Observe that property φ is satisfied over a path π if, and only if, either π visits every $V(\theta_r)$ infinitely many times or visits some of the $V(\psi_l)$ only a finite number of times.

For the game $(\mathcal{G}, \kappa)[z]$, let $W = (V, E, (\mathsf{w}_a)_{a \in \mathrm{N}})$ be the underlying multi-weighted graph, and for every edge $e \in E$ introduce a variable x_e. Informally, the value x_e is the number of times that the edge e is used on a cycle. Formally, let $\mathsf{src}(e) = \{v \in V : \exists w\, e = (v, w) \in E\}$; $\mathsf{trg}(e) = \{v \in V : \exists w\, e = (w, v) \in E\}$; $\mathsf{out}(v) = \{e \in E : \mathsf{src}(e) = v\}$; and $\mathsf{in}(v) = \{e \in E : \mathsf{trg}(e) = v\}$.

Consider ψ_l for some $1 \leq l \leq m$, and define the linear program $\mathsf{LP}(\psi_l)$ with the following inequalities and equations:

Eq1: $x_e \geq 0$ for each edge e — a basic consistency criterion;

Eq2: $\Sigma_{e \in E} x_e \geq 1$ — ensures that at least one edge is chosen;

Eq3: for each $a \in \mathrm{N}$, $\Sigma_{e \in E} \mathsf{w}_a(\mathsf{src}(e)) x_e \geq 0$ — this enforces that the total sum of any solution is non-negative;

Eq4: $\Sigma_{\mathsf{src}(e) \cap V(\psi_l) \neq \emptyset} x_e = 0$ — this ensures that no state in $V(\psi_l)$ is in the cycle associated with the solution;

Eq5: for each $v \in V$, $\Sigma_{e \in \mathsf{out}(v)} x_e = \Sigma_{e \in \mathsf{in}(v)} x_e$ — this condition says that the number of times one enters a vertex is equal to the number of times one leaves that vertex.

By construction, it follows that $\mathsf{LP}(\psi_l)$ admits a solution if and only if there exists a path π in $\mathcal{G}$ such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i and visits $V(\psi_l)$ only *finitely many times*. In addition, consider the linear program $\mathsf{LP}(\theta_1, \ldots, \theta_n)$ defined with the following inequalities and equations:

Eq1: $x_e \geq 0$ for each edge e — a basic consistency criterion;

Eq2: $\Sigma_{e \in E} x_e \geq 1$ — ensures that at least one edge is chosen;

Eq3: for each $a \in \mathrm{N}$, $\Sigma_{e \in E} \mathsf{w}_a(\mathsf{src}(e)) x_e \geq 0$ — this enforces that the total sum of any solution is non-negative;

Eq4: for all $1 \leq r \leq n$, $\Sigma_{\mathsf{src}(e) \cap V(\theta_r) \neq \emptyset} x_e \geq 1$ — this ensures that for every $V(\theta_r)$ at least one state is in the cycle;

Eq5: for each $v \in V$, $\Sigma_{e \in \mathsf{out}(v)} x_e = \Sigma_{e \in \mathsf{in}(v)} x_e$ — this condition says that the number of times one enters a vertex is equal to the number of times one leaves that vertex.

In this case, $\mathsf{LP}(\theta_1, \ldots, \theta_n)$ admits a solution if and only if there exists a path π such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i and visits every $V(\theta_r)$ *infinitely many times*.

Since the constructions above are polynomial in the size of both $(\mathcal{G}, \kappa)$ and φ, we can conclude it is possible to check in NP the statement that there is a path π satisfying φ such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i in the game if and only if one of the two linear programs defined above has a solution.

For the lower-bound, we use [Ummels and Wojtczak, 2011] and observe that if φ is true and $\beta = 0$, then the problem is equivalent to checking whether the mp game has a Nash equilibrium. □

We now turn our attention to the strong implementation of the equilibrium design problem. As in this section, we first consider LTL specifications and then GR(1) specifications.

6.3 Equilibrium Design: Strong Implementation

Although the principal may find $\mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$ to be good news, it might not be good enough. It could be that even though there is a desirable Nash equilibrium, the others might be undesirable. This motivates us to consider the *strong implementation* variant of equilibrium design. Intuitively, in a strong implementation, we require that *every* Nash equilibrium outcome satisfies the specification φ, for a *non-empty* set of outcomes. Then, let $\mathrm{SI}(\mathcal{G}, \varphi, \beta)$ denote the set of subsidy schemes κ given budget β over $\mathcal{G}$ such that:

1. $(\mathcal{G}, \kappa)$ has at least one Nash equilibrium outcome,
2. every Nash equilibrium outcome of $(\mathcal{G}, \kappa)$ satisfies φ.

Formally we define it as follows:

$$\mathrm{SI}(\mathcal{G}, \varphi, \beta) = \{\kappa \in \mathcal{K}(\mathcal{G}, \beta) : \mathrm{NE}(\mathcal{G}, \kappa) \neq \varnothing \wedge \forall \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa) \text{ s.t. } \pi(\vec{\sigma}) \models \varphi\}.$$

This gives us the following decision problem:

Definition 33 (Strong Implementation)**.** Given a game $\mathcal{G}$, formula φ, and budget β:

$$\text{Is it the case that } \mathrm{SI}(\mathcal{G}, \varphi, \beta) \neq \varnothing?$$

Strong Implementation can be solved with a 5-step procedure where the first four steps are as in Weak Implementation, and the last step (step 5) is as follows:

1. Guess:
   - a subsidy scheme $\kappa \in \mathcal{K}(\mathcal{G}, \beta)$,
   - a state $s \in \mathrm{St}$ for every player $i \in \mathrm{N}$, and
   - punishment memoryless strategies $(\vec{\sigma}_{-1}, \ldots, \vec{\sigma}_{-n})$ for all players $i \in \mathrm{N}$;
2. Compute $(\mathcal{G}, \kappa)$;
3. Compute $z \in \mathbb{R}^{\mathrm{N}}$;
4. Compute the game $(\mathcal{G}, \kappa)[z]$ by removing the states s such that $\mathtt{pun}_i(s) \leq z_i$ for some player i and the transitions $(s, \vec{\mathsf{a}}_{-i})$ that are not z_i secure for player i;
5. Check whether:
   (a) there is no ultimately periodic path π in $(\mathcal{G}, \kappa)[z]$ such that $z_i \leq \mathsf{pay}_i(\pi)$ for each $i \in \mathrm{N}$;
   (b) there is an ultimately periodic path π in $(\mathcal{G}, \kappa)[z]$ such that $\pi \models \neg\varphi$ and $z_i \leq \mathsf{pay}_i(\pi)$, for each $i \in \mathrm{N}$.

For step 5, observe that a positive answer to 5(a) or 5(b) is a counterexample to $\kappa \in \mathrm{SI}(\mathcal{G}, \varphi, \beta)$. Then, to carry out this procedure for the Strong Implementation problem with LTL specifications, consider the following $\mathsf{LTL}^{\mathsf{Lim}}$ formulae:

$$\varphi_{\exists} = \bigwedge_{i \in \mathrm{N}} (\mathsf{mp}(i) \geq z_i);$$
$$\varphi_{\forall} = \varphi_{\exists} \rightarrow \varphi.$$

Notice that the expression $\mathrm{NE}(\mathcal{G}, \kappa) \neq \varnothing$ can be expressed as "there exists a path π in $\mathcal{G}$ that satisfies formula $\varphi_\exists$". On the other hand, the expression $\forall \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa)$ such that $\pi(\vec{\sigma}) \models \varphi$ can be expressed as "for every path π in $\mathcal{G}$, if π satisfies formula $\varphi_\exists$, then π also satisfies formula φ". Thus, using these two formulae, we obtain the following result.

Corollary 21. Strong Implementation with LTL specifications is PSPACE-complete.

Proof. Membership follows from the fact that step 5(a) can be solved by existential $\mathsf{LTL}^{\mathsf{Lim}}$ model checking, whereas step 5(b) by universal $\mathsf{LTL}^{\mathsf{Lim}}$ model checking—both clearly in PSPACE by Savitch's theorem [Savitch, 1970]. Hardness is similar to the construction in Theorem 19. □

The (deterministic) algorithm implementing the procedure for Strong Implementation is shown in Algorithm 13.

Algorithm 13 Strong Implementation.

Input: A game $\mathcal{G}$, a specification formula φ, and budget β.
for $\kappa \in \mathcal{K}(\mathcal{G}, \beta)$ **do**
    Compute $(\mathcal{G}, \kappa)$
    for $i \in \mathrm{N}$ and $s \in \mathrm{St}$ **do**
        Compute $\mathsf{pun}_i((\mathcal{G}, \kappa))$
    end for
    $f_\exists \leftarrow \bot; f_\forall \leftarrow \top$
    for $z \in \{\mathsf{pun}_i(s) : s \in \mathrm{St}\}^{\mathrm{N}}$ **do**
        Compute $(\mathcal{G}, \kappa)[z]$
        if there exists $\pi \in (\mathcal{G}, \kappa)[z]$ such that for each $i \in \mathrm{N}, \mathsf{pay}_i(\pi) \geq z_i$ **then**
            $f_\exists \leftarrow \top$
        end if
    end for
    for $z \in \{\mathsf{pun}_i(s) : s \in \mathrm{St}\}^{\mathrm{N}}$ **do**
        if there exists $\pi \in (\mathcal{G}, \kappa)[z]$ s.t. for each $i \in \mathrm{N}, \mathsf{pay}_i(\pi) \geq z_i \wedge \pi \models \neg\varphi$ **then**
            $f_\forall \leftarrow \bot$
        end if
    end for
    if $(f_\exists \wedge f_\forall)$ **then**
        return Accept
    end if
end for
return Reject

Case with GR(1) specifications. Notice that the first part, *i.e.*, $\mathrm{NE}(\mathcal{G}, \kappa) \neq \varnothing$ can be solved in NP [Ummels and Wojtczak, 2011]. For the second part, observe that

$$\forall \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa) \text{ such that } \pi(\vec{\sigma}) \models \varphi$$

is equivalent to

$$\neg\exists \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa) \text{ such that } \pi(\vec{\sigma}) \models \neg\varphi.$$

Thus we have

$$\neg\varphi = \bigwedge_{l=1}^{m} \mathbf{GF}\psi_l \wedge \neg\big(\bigwedge_{r=1}^{n} \mathbf{GF}\theta_r\big).$$

To check this, we modify the LP in Theorem 20. Specifically, we modify Eq4 in $\mathsf{LP}(\theta_1, \ldots, \theta_n)$ to encode the θ-part of $\neg\varphi$. Thus, we have the following equation in $\mathsf{LP}'(\theta_1, \ldots, \theta_n)$:

Eq4: there exists r, $1 \leq r \leq n$, $\Sigma_{\mathsf{src}(e) \cap V(\theta_r) \neq \emptyset} x_e = 0$ — this condition ensures that at least one set $V(\theta_r)$ does not have any state in the cycle associated with the solution.

In this case, $\mathsf{LP}'(\theta_1, \ldots, \theta_n)$ has a solution if and only if there is a path π such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i and, for at least one $V(\theta_r)$, its states are visited only *finitely many times*. Thus, we have a procedure that checks if there is a path π that satisfies $\neg\varphi$ such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i, if and only if both linear programs have a solution. Using this new construction, we can now prove the following result.

Theorem 22. Strong Implementation with GR(1) specifications is Σ_2^{P}-complete.

Proof. For membership, observe that by rearranging the problem statement, we have the following question:
Check whether the following expression is true

$$\exists \kappa \in \mathcal{K}(\mathcal{G}, \beta), \tag{1}$$

$$\exists \vec{\sigma} \in \sigma_1 \times \cdots \times \sigma_n, \text{ such that } \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa), \tag{2}$$

and

$$\forall \vec{\sigma}' \in \sigma_1 \times \cdots \times \sigma_n, \text{ if } \vec{\sigma}' \in \mathrm{NE}(\mathcal{G}, \kappa) \text{ then } \pi(\vec{\sigma}') \models \varphi. \tag{3}$$

Statement (2) can be checked in NP (Theorem 16). Whereas, verifying statement (3) is in coNP; to see this, notice that we can rephrase (3) as follows: $\nexists z \in \{\mathtt{pun}_i(s) :$

$s \in \mathrm{St}\}^{\mathrm{N}}$ such that both $\mathsf{LP}(\psi_l)$ and $\mathsf{LP}'(\theta_1, \ldots, \theta_n)$ have a solution in $(\mathcal{G}, \kappa)[z]$. Thus Σ_2^{P} membership follows.

We prove hardness by a reduction from QSAT_2 (satisfiability of quantified Boolean formula with 2 alternations) [Papadimitriou, 1994]. Let $\psi(\mathbf{x}, \mathbf{y})$ be an $n+m$ variable Boolean 3DNF formula, where $\mathbf{x} = \{x_1, \ldots, x_n\}$ and $\mathbf{y} = \{y_1, \ldots, y_n\}$, with $t_1, \ldots, t_k$ terms. Write $\mathbf{t}_j$ for the set of literals in j-th term and $\mathbf{t}_j^i$ for the i-th literal in $\mathbf{t}_j$. Moreover write x_i^j and y_i^j for variable $x_i \in \mathbf{x}$ and $y_i \in \mathbf{y}$ that appears in j-th term, respectively. For instance, if the fifth term is of the form of $(x_2 \wedge \neg x_3 \wedge y_4)$, then we have $\mathbf{t}_5 = \{x_2^5, x_3^5, y_4^5\}$ and $\mathbf{t}_5^1 = x_2^5$. Let $\mathbf{T} = \{\mathbf{t}_i \cap \mathbf{y} : 1 \leq i \leq k\}$, that is, the set of subset of $\mathbf{t}_i$ that contains only y-literals.

For a formula $\psi(\mathbf{x}, \mathbf{y})$ we construct an instance of Strong Implementation such that $\mathrm{SI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$ if and only if there is an $\vec{\mathbf{x}} \in \{0,1\}^n$ such that $\psi(\mathbf{x}, \mathbf{y})$ is true for every $\vec{\mathbf{y}} \in \{0,1\}^m$. Let $\mathcal{G}$ be such a game where

- $\mathrm{N} = \{1, 2\}$,
- $\mathrm{St} = \{\bigcup_{j \in [1,k]}(\mathbf{t}_j \times \{0,1\}^3)\} \cup \{\mathbf{T} \times \{0\}^3\} \cup \{(\mathtt{source}, \{0\}^3), (\mathtt{sink}, \{0\}^3)\}$,
- $s_0 = \mathtt{source}$,
- for each state $s \in \mathrm{St}$
  - $\mathrm{Ac}_1(s) = \{\{\mathbf{T} \cup \{\mathtt{sink}\}\} \times \{0\}^3\}$, $\mathrm{Ac}_2(s) = \{\varepsilon\}$, if $s = (\mathtt{source}, \{0\}^3)$,
  - $\mathrm{Ac}_1(s) = \{\mathbf{t}_i^1 : s[0] \subseteq \mathbf{t}_i \wedge i \in [1,k]\}$, $\mathrm{Ac}_2(s) = \{0,1\}^3$, if $s \in \{\mathbf{T} \times \{0\}^3\}$,
  - $\mathrm{Ac}_1(s) = \{\varepsilon\}$, $\mathrm{Ac}_2(s) = \{\varepsilon\}$, if $s \in \bigcup_{j \in [1,k]}(\mathbf{t}_j \times \{0,1\}^3)$,
- for an action profile $\vec{\mathsf{a}} = (\mathsf{a}_1, \mathsf{a}_2)$
  - $\mathsf{tr}(s, \vec{\mathsf{a}}) = \mathsf{a}_1$, if $s = (\mathtt{source}, \{0\}^3)$,
  - $\mathsf{tr}(s, \vec{\mathsf{a}}) = (\mathsf{a}_1, \mathsf{a}_2)$, if $s \in \{\mathbf{T} \times \{0\}^3\}$,
  - $\mathsf{tr}(s, \vec{\mathsf{a}}) = (\mathbf{t}_j^{(i \bmod 3)+1}, s[1])$, if $s = (\mathbf{t}_j^i, s[1]) \in \bigcup_{j \in [1,k]}(\mathbf{t}_j \times \{0,1\}^3)$;
  - $\mathsf{tr}(s, \vec{\mathsf{a}}) = s$, otherwise;
- for each state $s \in \mathrm{St}$, $\lambda(s) = s[0]$,
- for each state $s \in \mathrm{St}$
  - $\mathsf{w}_1(s) = \frac{2}{3}$, if $s[0] = \mathtt{sink}$[2],

[2]This can be implemented by a macrostate with three substates—2 substates with weight of 1, and 1 with weight of 0—forming a simple cycle.

– $\mathsf{w}_1(s) = 0$, otherwise;

- the payoff of player $i \in \mathrm{N}$ for an ultimately periodic path π in $\mathcal{G}$ is
  - $\mathsf{pay}_1(\pi) = \mathsf{mp}(\mathsf{w}_1(\pi))$,
  - $\mathsf{pay}_2(\pi) = -\mathsf{mp}(\mathsf{w}_1(\pi))$,

Furthermore, let $\beta = |\mathbf{x}|$ and the $\mathsf{GR}(1)$ property to be $\varphi := \mathbf{GF}\ \neg\mathtt{sink}$. Define a (partial) subsidy scheme $\kappa : \mathbf{x} \to \{0, 1\}$. The weights are updated with respect to κ as follows:
for each $s \in \mathrm{St}$ such that $s[0] \in \mathbf{t}_j \setminus \mathbf{y}$, that is, an x-literal that appears in term t_j

$$\mathsf{w}_1(s) = \begin{cases} 1, & \text{if } \kappa(s) = 1 \wedge s[0] \text{ is not negated in } t_j \\ 1, & \text{if } \kappa(s) = 0 \wedge s[0] \text{ is negated in } t_j \\ 0, & \text{if } \kappa(s) = 1 \wedge s[0] \text{ is negated in } t_j \\ 0, & \text{otherwise;} \end{cases}$$

for each $s \in \mathrm{St}$ such that $s[0] \in \mathbf{t}_j \setminus \mathbf{x}$, that is, a y-literal that appears in term t_j, $s[0] = \mathbf{t}^i_j$

$$\mathsf{w}_1(s) = \begin{cases} 1, & \text{if } s[1][i] = 1 \wedge s[0] \text{ is not negated in } t_j \\ 1, & \text{if } s[1][i] = 0 \wedge s[0] \text{ is negated in } t_j \\ 0, & \text{if } s[1][i] = 1 \wedge s[0] \text{ is negated in } t_j \\ 0, & \text{otherwise;} \end{cases}$$

the weights of other states remain unchanged.

The construction is now complete, and polynomial to the size of formula $\psi(\mathbf{x}, \mathbf{y})$. We claim that $\mathrm{SI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$ if and only if there is an $\vec{\mathbf{x}} \in \{0, 1\}^n$ such that $\psi(\mathbf{x}, \mathbf{y})$ is true for every $\vec{\mathbf{y}} \in \{0, 1\}^m$. From left to right, consider a subsidy scheme $\kappa \in \mathrm{SI}(\mathcal{G}, \varphi, \beta)$ which implies that there exists no Nash equilibrium run in $(\mathcal{G}, \kappa)$ that ends up in $\mathtt{sink}$. This means that for every action $\vec{\mathsf{a}}_2 \in \mathrm{Ac}_2(s)$, there exists $\vec{\mathsf{a}}_1 \in \mathrm{Ac}_1(s), s \in \{\mathbf{T} \times \{0\}^3\}$, such that $\mathsf{pay}_1(\pi) = 1$, where π is the resulting path of the joint action. Observe that this corresponds to the existence of (at least) a term t_i, which evaluates to true under assignment $\vec{\mathbf{x}}$, regardless the value of $\vec{\mathbf{y}}$. From right to left, consider an assigment $\vec{\mathbf{x}} \in \{0, 1\}^n$ such that for all $\vec{\mathbf{y}} \in \{0, 1\}^m$, the formula $\psi(\mathbf{x}, \mathbf{y})$ is true. This means that for every $\vec{\mathbf{y}}$, there exists (at least one) term t_i in $\psi(\mathbf{x}, \mathbf{y})$ that evaluates to true. By construction, specifically the weight updating rules, for every $\vec{\mathsf{a}}_2$ corresponding to assignment $\vec{\mathbf{y}}$, there exists $\mathbf{t}_j$ such that $\forall i \in [1, 3], \mathsf{w}_1(\mathbf{t}^i_j) = 1$. This means that player 1 can always get payoff equals to 1, therefore, any run that ends in $\mathtt{sink}$ is not sustained by Nash equilibrium. □

6.4 Optimality and Uniqueness of Solutions

Having asked the questions studied in the previous sections, the principal – the *designer* in the equilibrium design problem – may want to explore further information. Because the power of the principal is limited by its budget, and because from the point of view of the system, it may be associated with a reward (*e.g.*, money, savings, etc.) or with the inverse of the amount of a finite resource (*e.g.*, time, energy, etc.) an obvious question is asking about *optimal* solutions. This leads us to *optimisation* variations of the problems we have studied. Informally, in this case, we ask what is the least budget that the principal needs to ensure that the implementation problems have positive solutions. The principal may also want to know whether a given subsidy scheme is *unique*, so that there is no point in looking for any other solutions to the problem. In this section, we investigate this kind of problems, and classify our study into two parts, one corresponding to the Weak Implementation problem and another one corresponding to the Strong Implementation problem.

6.4.1 Optimality and Uniqueness in the Weak Domain

We can now define formally some of the problems that we will study in the rest of this section. To start, the optimisation variant for Weak Implementation is defined as follows.

Definition 34 (Opt-WI)**.** Given a game $\mathcal{G}$ and a specification formula φ:

$$\text{What is the optimum budget } \beta \text{ such that } \mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing?$$

Another natural problem, which is related to Opt-WI, is the "exact" variant – a membership question. In this case, in addition to $\mathcal{G}$ and φ, we are also given an integer b, and ask whether it is indeed the smallest amount of budget that the principal has to spend for some optimal weak implementation. This decision problem is formally defined as follows.

Definition 35 (Exact-WI)**.** Given a game $\mathcal{G}$, a specification formula φ, and an integer b:

$$\text{Is } b \text{ equal to the optimum budget for } \mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing?$$

To study these problems, it is useful to introduce some concepts first. More specifically, let us introduce the concept of *implementation efficiency*. We say that a Weak Implementation (resp. Strong Implementation) is *efficient* if $\beta =$

$\mathsf{cost}(\kappa)$ and there is no κ' such that $\mathsf{cost}(\kappa') < \mathsf{cost}(\kappa)$ and $\kappa' \in \mathrm{WI}(\mathcal{G}, \varphi, \beta)$ (resp. $\kappa' \in \mathrm{SI}(\mathcal{G}, \varphi, \beta)$). In addition to the concept of efficiency for an implementation problem, it is also useful to have the following result.

Proposition 3. Let z_i be the largest payoff that player i can get after deviating from a path π. The optimum budget is an integer between 0 and $\sum_{i \in \mathrm{N}} z_i \cdot (|\mathrm{St}| - 1)$.

Proof. The lower-bound is straightforward. The upper-bound follows from the fact that the maximum value the principal has to pay to player i is when the path π is a simple cycle and formed from all states in St, apart from 1 deviation state. □

Using Proposition 3, we can show that both Opt-WI and Exact-WI can be solved in PSPACE for LTL specifications. Intuitively, the reason is that we can use the upper bound given by Proposition 3 to go through all possible solutions in exponential time, but using only nondeterministic polynomial space. Formally, we have the following results.

Theorem 23. Opt-WI with LTL specifications is FPSPACE-complete.

Proof. Since the search space is bounded (Proposition 3), by using Weak Implementation an an oracle we can iterate through every instance and return the smallest β such that $\mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$. Moreover, each instance is of polynomial size in the size of the input. Thus membership in FPSPACE follows. Hardness is straightforward. □

Corollary 24. Exact-WI with LTL specifications is PSPACE-complete.

The fact that both Opt-WI and Exact-WI with LTL specifications can be answered in PSPACE does not come as a big surprise: checking an instance can be done using polynomial space and there are only exponentially many instances to be checked. However, for Opt-WI and Exact-WI with GR(1) specifications, these two problems are more interesting.

Theorem 25. Opt-WI with GR(1) specifications is $\mathsf{FP}^{\mathsf{NP}}$-complete.

Proof. Membership follows from the fact that the search space, which is bounded as in Proposition 3, can be fully explored using binary search and Weak Implementation as an oracle. More precisely, we can find the smallest budget β such that $\mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$ by checking every possible value for β, which lies between 0 and 2^n, where n is the length of the encoding of the instance. Since we need logarithmically many calls to the NP oracle (to Weak Implementation), in the end we have searching procedure that runs in polynomial time.

For hardness we reduce from TSP Cost (optimal travelling salesman problem) that is known to be $\mathsf{FP}^{\mathsf{NP}}$-complete [Papadimitriou, 1994]. Given a TSP Cost instance (G, c), $G = (V, E)$ is a graph, $c : E \to \mathbb{Z}$ is a cost function. We assume that $\mathrm{WI}(\mathcal{G}, \varphi, \beta)$ is efficient. To encode TSP Cost instance, we construct a game $\mathcal{G}$ and $\mathsf{GR}(1)$ formula φ, such that the optimum budget β corresponds to the value of optimum tour. Let $\mathcal{G}$ be such a game where

- $\mathrm{N} = \{1\}$,
- $\mathrm{St} = \{(v, e) : v \in V \wedge e \in \mathsf{in}(v)\} \cup \{(\mathtt{sink}, \varepsilon)\}$,
- s_0 can be chosen arbitrarily from $\mathrm{St} \setminus \{(\mathtt{sink}, \varepsilon)\}$,
- for each state $(v, e) \in \mathrm{St}$ and edge $e' \in E \cup \{\varepsilon\}$
  - $\mathsf{tr}((v, e), e') = (\mathsf{trg}(e'), e')$, if $v \neq \mathtt{sink}$ and $e' \neq \varepsilon$,
  - $\mathsf{tr}((v, e), e') = (\mathtt{sink}, \varepsilon)$, otherwise;
- for each state $(v, e) \in \mathrm{St}$
  - $\mathsf{w}_1((v, e)) = \max\{c(e') : e' \in E\} - c(e)$, if $v \neq \mathtt{sink}$,
  - $\mathsf{w}_1((v, e)) = \max\{c(e') : e' \in E\}$, otherwise;
- the payoff of player 1 for a path π in $\mathcal{G}$ is $\mathsf{pay}_1(\pi) = \mathsf{mp}(\mathsf{w}_1(\pi))$,
- for each state $(v, e) \in \mathrm{St}$, the set of actions available to player 1 is $\mathsf{out}(v) \cup \{\varepsilon\}$,
- for each state $(v, e) \in \mathrm{St}$, $\lambda((v, e)) = v$.

Furthermore, let $\varphi := \bigwedge_{v \in V} \mathbf{GF}\, v$. The construction is now complete, and is polynomial to the size of (G, c).

Now, consider the smallest $\mathsf{cost}(\kappa), \kappa \in \mathrm{WI}(\mathcal{G}, \varphi, \beta)$. We argue that $\mathsf{cost}(\kappa)$ is indeed the lowest value such that a tour in G is attainable. Suppose for contradiction, that there exists κ' such that $\mathsf{cost}(\kappa') < \mathsf{cost}(\kappa)$. Let π' be a path in $(\mathcal{G}, \kappa')$ and $z_1 = \mathsf{w}_1((\mathtt{sink}, \varepsilon))$ the largest value player 1 can get by deviating from π'. We have $\mathsf{pay}_1(\pi') < z_1$, and since for every $(v, e) \in \mathrm{St}$ there exists an edge to $(\mathtt{sink}, \varepsilon)$, thus player 1 would deviate to $(\mathtt{sink}, \varepsilon)$ and stay there forever. This deviation means that φ is not satisfied, which is a contradiction to $\kappa' \in \mathrm{WI}(\mathcal{G}, \varphi, \beta)$. The construction of φ also ensures that the path is a valid tour, i.e., the tour visits every city at least once. Notice that φ does not guarantee a Hamiltonian cycle. However, removing the condition of visiting each city *only once* does not remove the hardness, since *Euclidean*

TSP is NP-hard [Garey et al., 1976, Papadimitriou, 1977]. Therefore, in the planar case there is an optimal tour that visits each city only once, or otherwise, by the triangle inequality, skipping a repeated visit would not increase the cost. Finally, since $\mathrm{WI}(\mathcal{G}, \varphi, \beta)$ is efficient, we have β to be exactly the value of the optimum tour in the corresponding TSP Cost instance. □

Corollary 26. Exact-WI with GR(1) specifications is D^{P}-complete.

Proof. For membership, observe that an input is a "yes" instance of Exact-WI if and only if it is a "yes" instance of Weak Implementation *and* a "yes" instance of Weak Implementation Complement (the problem where one asks whether $\mathrm{WI}(\mathcal{G}, \varphi, \beta) = \varnothing$). Since the former problem is in NP and the latter problem is in coNP, membership in D^{P} follows. For the lower bound, we use the same reduction technique as in Theorem 25, and reduce from Exact TSP, a problem known to be D^{P}-hard [Papadimitriou, 1994, Papadimitriou and Yannakakis, 1984]. □

Following [Papadimitriou, 1984], we may naturally ask whether the optimal solution given by Opt-WI is unique. We call this problem UOpt-WI. For some fixed budget β, it may be the case that for two subsidy schemes $\kappa, \kappa' \in \mathrm{WI}(\mathcal{G}, \varphi, \beta)$ – we assume the implementation is efficient – we have $\kappa \neq \kappa'$ and $\mathsf{cost}(\kappa) = \mathsf{cost}(\kappa')$. With LTL specifications, it is not difficult to see that we can solve UOpt-WI in polynomial space. Therefore, we have the following result.

Corollary 27. UOpt-WI with LTL specifications is PSPACE-complete.

For GR(1) specifications, we reason about UOpt-WI using the following procedure:

1. Find the exact budget using binary search and Weak Implementation as an oracle;

2. Use an NP oracle once to guess two distinct subsidy schemes with precisely this budget; if no such subsidy schemes exist, return "yes"; otherwise, return "no".

The above decision procedure clearly is in Δ_2^{P} (for the upper bound). Furthermore, since Theorem 25 implies Δ_2^{P}-hardness [Krentel, 1988] (for the lower bound), we have the following corollary.

Corollary 28. UOpt-WI with GR(1) specifications is Δ_2^{P}-complete.

6.4.2 Optimality and Uniqueness in the Strong Domain

In this subsection, we study the same problems as in the previous subsection but with respect to the Strong Implementation variant of the equilibrium design problem. We first formally define the problems of interest and then present the two first results.

Definition 36 (Opt-SI)**.** Given a game $\mathcal{G}$ and a specification formula φ:

$$\text{What is the optimum budget } \beta \text{ such that } \mathrm{SI}(\mathcal{G}, \varphi, \beta) \neq \varnothing?$$

Definition 37 (Exact-SI)**.** Given a game $\mathcal{G}$, a specification formula φ, and an integer b:

$$\text{Is } b \text{ equal to the optimum budget for } \mathrm{SI}(\mathcal{G}, \varphi, \beta) \neq \varnothing?$$

For the same reasons discussed in the weak versions of these two problems, we can prove the following two results with respect to games with LTL specifications.

Theorem 29. Opt-SI with LTL specifications is FPSPACE-complete.

Proof. The proof is analogous to that of Theorem 23. □

Corollary 30. Exact-SI with LTL specifications is PSPACE-complete.

For GR(1) specifications, observe that using the same arguments for the upper-bound of Opt-WI with GR(1) specifications, we obtain the upper-bound for Opt-SI with GR(1) specifications. Then, it follows that Opt-SI is in $\mathsf{FP}^{\Sigma_2^{\mathsf{P}}}$. For hardness, we define an $\mathsf{FP}^{\Sigma_2^{\mathsf{P}}}$-complete problem, namely Weighted MinQSAT$_2$. Recall that in QSAT$_2$ we are given a Boolean 3DNF formula $\psi(\mathbf{x}, \mathbf{y})$ and sets $\mathbf{x} = \{x_1, \ldots, x_n\}, \mathbf{y} = \{y_1, \ldots, y_m\}$, with a set of terms $T = \{t_1, \ldots, t_k\}$. Define Weighted MinQSAT$_2$ as follows. Given $\psi(\mathbf{x}, \mathbf{y})$ and a weight function $\mathsf{c} : \mathbf{x} \to \mathbb{Z}^{\geq}$, Weighted MinQSAT$_2$ is the problem of finding an assignment $\vec{\mathbf{x}} \in \{0,1\}^n$ with the least total weight such that $\psi(\mathbf{x}, \mathbf{y})$ is true for every $\vec{\mathbf{y}} \in \{0,1\}^m$. Observe that Weighted MinQSAT$_2$ generalises MinQSAT$_2$, which is known to be $\mathsf{FP}^{\Sigma_2^{\mathsf{P}}[\log n]}$-hard [Chockler and Halpern, 2004], *i.e.*, MinQSAT$_2$ is an instance of Weighted MinQSAT$_2$, where all weights are 1.

Theorem 31. Weighted MinQSAT$_2$ is $\mathsf{FP}^{\Sigma_2^{\mathsf{P}}}$-complete.

Proof. Membership follows from the upper-bound of MinQSAT$_2$ [Chockler and Halpern, 2004]: since we have an exponentially large input with respect to that of MinQSAT$_2$, by using binary search we will need polynomially many calls to the Σ_2^{P} oracle. Hardness is immediate [Chockler and Halpern, 2004]. □

Now that we have an $\mathsf{FP}^{\Sigma_2^{\mathrm{P}}}$-hard problem in our hands, we can proceed to determine the complexity class of Opt-SI with $\mathsf{GR}(1)$ specifications. For the upper bound we one can use arguments analogous to those in Theorem 25. For the lower bound, one can reduce from Weighted MinQSAT$_2$. Formally, we have:

Theorem 32. Opt-SI with $\mathsf{GR}(1)$ specifications is $\mathsf{FP}^{\Sigma_2^{\mathrm{P}}}$-complete.

Proof. Membership uses arguments analogous to those in Theorem 25. For hardness, we reduce Weighted MinQSAT$_2$ to Opt-SI using the same techniques used in Theorem 22 with few modifications. Given a Weighted MinQSAT$_2$ instance $(\psi(\mathbf{x},\mathbf{y}),\mathsf{c})$, we construct a game $\mathcal{G}$ and $\mathsf{GR}(1)$ formula φ, such that the optimum budget β corresponds to the value of optimal solution to $(\psi(\mathbf{x},\mathbf{y}),\mathsf{c})$. To this end, we may assume that $\mathrm{SI}(\mathcal{G},\varphi,\beta)$ is efficient and construct $\mathcal{G}$ with exactly the same rules as in Theorem 22 except for the following:

- clearly the value of β is unknown,
- the initial weight for each state $s \in \mathrm{St}$
  - $\mathsf{w}_1(s) = \frac{2}{3}$, if $s[0] = \texttt{sink}$,
  - $$\mathsf{w}_1(s) = \begin{cases} -\mathsf{c}(s[0]) + 1, & \text{if } s[0] \in \mathbf{t}_j \setminus \mathbf{y} \wedge s[0] \text{ is not negated in } t_j \\ 1, & \text{if } s[0] \in \mathbf{t}_j \setminus \mathbf{y} \wedge s[0] \text{ is negated in } t_j; \end{cases}$$
  - $\mathsf{w}_1(s) = 0$, otherwise;
- given a subsidy scheme κ, we update the weight for each $s \in \mathrm{St}$ such that $s[0] \in \mathbf{t}_j \setminus \mathbf{y}$, that is, an x-literal that appears in term t_j
  $$\mathsf{w}_1(s) = \begin{cases} \mathsf{w}_1(s) + \kappa(s), & \text{if } s[0] \text{ is not negated in } t_j \\ \mathsf{w}_1(s), & \text{otherwise;} \end{cases}$$

the construction is complete and polynomial to the size of $(\psi(\mathbf{x},\mathbf{y}),\mathsf{c})$.

Let o be the optimal solution to Weighted MinQSAT$_2$ given the input $(\psi(\mathbf{x},\mathbf{y}),\mathsf{c})$. We claim that β is exactly o. To see this, consider the smallest $\mathsf{cost}(\kappa)$, $\kappa \in \mathrm{SI}(\mathcal{G},\varphi,\beta)$. We argue that this is indeed the least total weight of an assignment $\vec{\mathbf{x}}$ such that $\psi(\mathbf{x},\mathbf{y})$ is true for every $\vec{\mathbf{y}}$. Assume towards a contradiction that $\mathsf{cost}(\kappa) < o$. By the construction of $\mathsf{w}_1(\cdot)$, there exists no π such that $\mathsf{pay}_1(\pi) > \frac{2}{3}$. Therefore, any run π' that ends up in $\texttt{sink}$ is sustained by Nash equilibrium, which is a contradiction to $\kappa \in \mathrm{SI}(\mathcal{G},\varphi,\beta)$. Now, since $\mathrm{SI}(\mathcal{G},\varphi,\beta)$ is efficient, by definition, there exists

no $\kappa' \in \mathrm{SI}(\mathcal{G}, \varphi, \beta)$ such that $\mathsf{cost}(\kappa') < \mathsf{cost}(\kappa)$. Thus we have β equals to o as required. □

Corollary 33. Exact-SI with GR(1) specifications is $\mathsf{D}_2^{\mathsf{P}}$-complete.

Proof. Membership follows from the fact that an input is a "yes" instance of Exact-SI (with GR(1) specifications) if and only if it is a "yes" instance of Strong Implementation *and* a "yes" instance of Strong Implementation Complement, the decision problem where we ask $\mathrm{SI}(\mathcal{G}, \varphi, \beta) = \varnothing$ instead. The lower bound follows from the hardness of Strong Implementation and Strong Implementation Complement problems, which immediately implies $\mathsf{D}_2^{\mathsf{P}}$-hardness [Aleksandrowicz et al., 2017, Lemma 3.2]. □

Furthermore, analogous to UOpt-WI, we also have the following corollaries.

Corollary 34. UOpt-SI with LTL specifications is PSPACE-complete.

Corollary 35. UOpt-SI with GR(1) specifications is Δ_3^{P}-complete.

6.5 Summary

In this chapter, we model agents as synchronously executing concurrent processes, with each agent receiving an integer payoff for every state the overall system visits; the overall payoff an agent receives over an infinite computation path is then defined to be the mean payoff over this path. While agents (naturally) seek to maximise their individual mean payoff, the designer of the subsidy scheme wishes to see some temporal logic formula satisfied, either on some or on every Nash equilibrium of the game.

With this model, we assume that the designer—an external principal—has a finite budget that is available for making subsidies, and this budget can be allocated across agent/state pairs. By allocating this budget appropriately, the principal can incentivise players away from some states and towards others. Since the principal has some temporal logic goal formula, it desires to allocate subsidies so that players are rationally incentivised to choose strategies so that the principal's temporal logic goal formula is satisfied in the path that would result from executing the strategies. For this general problem, following [Wooldridge et al., 2013], we identify two variants of the principal's mechanism design problem, which we refer to as Weak Implementation and Strong Implementation. In the Weak variant, we ask whether the principal can allocate the budget so that the goal is achieved on *some* computation

	LTL Spec.	GR(1) Spec.
Weak Implementation	PSPACE-complete (Thm. 19)	NP-complete (Thm. 20)
Strong Implementation	PSPACE-complete (Cor. 21)	Σ_2^{P}-complete (Thm. 22)
Opt-WI	FPSPACE-complete (Thm. 23)	$\mathsf{FP}^{\mathsf{NP}}$-complete (Thm. 25)
Opt-SI	FPSPACE-complete (Thm. 29)	$\mathsf{FP}^{\Sigma_2^{\mathsf{P}}}$-complete (Thm. 32)
Exact-WI	PSPACE-complete (Cor. 24)	D^{P}-complete (Cor. 26)
Exact-SI	PSPACE-complete (Cor. 30)	$\mathsf{D}_2^{\mathsf{P}}$-complete (Cor. 33)
UOpt-WI	PSPACE-complete (Cor. 27)	Δ_2^{P}-complete (Cor. 28)
UOpt-SI	PSPACE-complete (Cor. 34)	Δ_3^{P}-complete (Cor. 35)

Table 6.1: Summary of main complexity results.

path that would be generated by Nash equilibrium strategies in the resulting system; in the Strong variation, we ask whether the principal can allocate the budget so that the resulting system has at least one Nash equilibrium, and moreover the temporal logic goal is satisfied on *all* paths that could be generated by Nash equilibrium strategies. For these two problems, we consider goals specified by LTL formulae or GR(1) formulae [Bloem et al., 2012], give algorithms for each case, and classify the complexity of the problem. While LTL is a natural language for the specification of properties of concurrent and multi-agent systems, GR(1) is an LTL fragment that can be used to easily express several prefix-independent properties of computation paths of reactive systems, such as ω-regular properties often used in automated formal verification. We then go on to study variations of these two problems, for example considering *optimality* and *uniqueness* of solutions, and show that the complexities of all such problems lie within the polynomial hierarchy, thus making them potentially amenable to efficient practical implementations. Table 6.1 summarises the main computational complexity results in this chapter.

Chapter 7

Implementation & Evaluation

In this chapter we present EVE (Equilibrium Verification Environment), a formal verification tool for the automated analysis of temporal equilibrium properties of concurrent and multi-agent systems. Systems are modelled using the Simple Reactive Module Language (SRML) as a collection of independent system components (players/agents in a game.) Players' goals are expressed using Linear Temporal Logic (LTL) formulae. EVE can be used to check the existence of pure strategy Nash equilibria in such systems and verify which temporal logic properties are satisfied in the equilibria. Part of this chapter appeared in the proceedings of ATVA'18 [Gutierrez et al., 2018a].

7.1 Description

Once a multi-agent system is modelled in SRML, it can be seen as a multi-player game in which players (the modules) use strategies to resolve the non-deterministic choices in the system. EVE uses Algorithm 9 to solve Non-Emptiness. The main idea behind this algorithm is illustrated in Figure 7.1. The general flow of the implementation is as follows. Let $\mathcal{G}_{\text{LTL}}$ be a game, modelled using SRML, with a set of players/modules $\mathrm{N} = \{1, \ldots, n\}$ and LTL goals $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$, one for each player. Using $\mathcal{G}_{\text{LTL}}$ we construct an associated concurrent game with parity goals $\mathcal{G}_{\text{PAR}}$ in order to shift reasoning on the set of Nash equilibria of $\mathcal{G}_{\text{LTL}}$ into the set of Nash equilibria of $\mathcal{G}_{\text{PAR}}$. The basic idea of this construction is, firstly, to transform all LTL goals in $\mathcal{G}_{\text{LTL}}$ into deterministic parity word (DPW) automata. To do this, we use LTL2BA tool [Gastin and Oddoux, 2001, Gastin and Oddoux, 2019] to transform the formulae into nondeterministic Büchi word (NBW) automata. From NBWs, we construct the associated deterministic parity word (DPW) automata via construction described in [Piterman, 2007]. Secondly, to perform a product construction of the Kripke structure

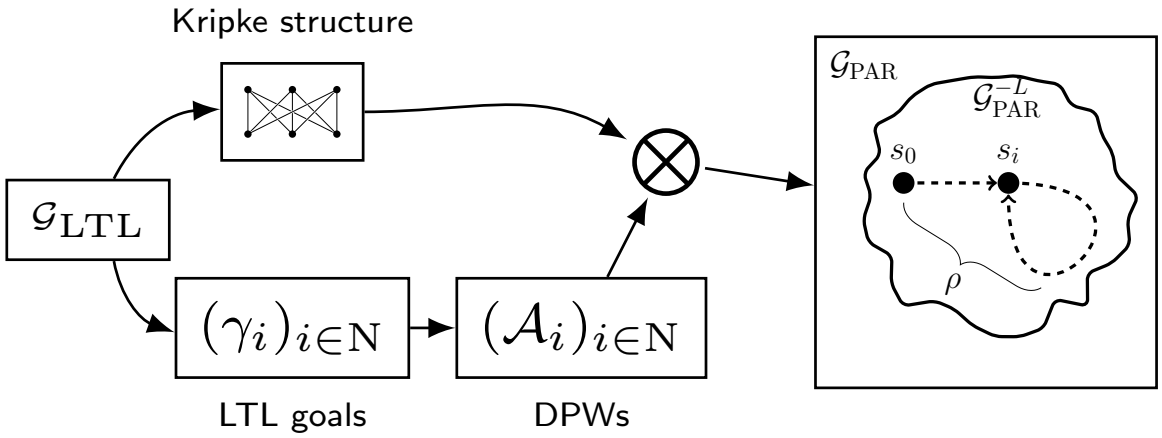

Figure 7.1: High-level workflow of EVE.

that represents $\mathcal{G}_{\text{LTL}}$ with the collection of DPWs in which the set of Nash equilibria of the input game is preserved. With $\mathcal{G}_{\text{PAR}}$ in our hands, we can then reason about Nash equilibria by solving a collection of parity games. To solve these parity games, we use PGSolver tool [Friedmann and Lange, 2010, PGSolver, 2019]. EVE then iterates through all possible set of "winners" $W \subseteq \text{N}$ (Algorithm 9 line 4) and computes a punishment region $\text{Pun}_j(\mathcal{G}_{\text{PAR}})$ for each $j \in L = \text{N}\backslash W$, with which a reduced parity game $\mathcal{G}_{\text{PAR}}^{-L} = \bigcap_{j\in L} \text{Pun}_j(\mathcal{G}_{\text{PAR}})$ is built. Notice that for each player j, $\text{Pun}_j(\mathcal{G}_{\text{PAR}})$ need only computed once and can be stored, thus resulting in a more efficient running time. Lastly, EVE checks whether there exists a path ρ in $\mathcal{G}_{\text{PAR}}^{-L}$ that satisfies the goals of each $i \in W$. To do this, we translate $\mathcal{G}_{\text{PAR}}^{-L}$ into a deterministic Streett automata, whose language is empty if and only if so is the set of Nash equilibria of $\mathcal{G}_{\text{PAR}}$. For E-Nash problem, we simply need to find a run in the witness returned when we check for Non-Emptiness; this can be done via automata intersection[1]. This algorithm runs in doubly exponential time, matching the optimal upper bound of the problem [Mogavero et al., 2014]. We obtain a doubly exponential blowup when converting LTL goals to DPWs. Computing punishment regions takes exponential time in the number of players and parity game priorities, while checking Streett automata emptyness can be done in polynomial time, thus resulting in an overall 2EXPTIME algorithm.

7.2 Features & Usage

EVE was developed in Python and can be used via webservice from `eve.cs.ox.ac.uk`. EVE takes as input a concurrent and multi-agent system described in SRML

[1]For A-Nash is straightforward, since it is the dual of E-Nash.

code, with player goals and a property φ to be checked specified in LTL. For Non-Emptiness, EVE returns "YES" (along with a set of winning players W) if the set of Nash equilibria in the system is not empty, and returns "NO" otherwise. For E-Nash (A-Nash), EVE returns "YES" if φ holds on *some* (*all*) Nash equilibria of the system, and "NO" otherwise. Moreover, EVE returns a witness for every "YES" instance, which also is the synthesised strategy profile. EVE is open-source and the code is available from `https://github.com/eve-mas/eve-parity`. To install EVE, we need the following modules:

- OPAM,
- OCaml version 4.03.x or later,
- Cairo,
- IGraph.

More detailed installation steps can be found in `https://github.com/eve-mas/eve-parity/blob/master/README.md`.

7.3 Case Studies

In this section, we present two case studies from the literature of concurrent and distributed systems to show the practical usage of EVE. Among other things, these two examples differ in the way they are modelled as a concurrent game. While the first one is played in an arena implicitly given by the specification of the players in the game (as done in [Gutierrez et al., 2017b]), the second and third ones are played on a graph, *e.g.*, as done in [Alur et al., 2002] with the use of concurrent game structures. Both of these modelling approaches can be used within our tool. We will also use these two examples to evaluate EVE's performance in practice and compare it against MCMAS and PRALINE in Section 7.4. It is important to note that, while MCMAS can be used to solve problems in rational verification, it is tailored to reason about systems with imperfect information setting. As such, the encoding of the benchmarks used in the experiments might inadvertently give a slight disadvantage for MCMAS.

7.3.1 Gossip Protocol

These are a class of networking and communication protocols that mimic the way social networks disseminate information. They have been used to solve problems

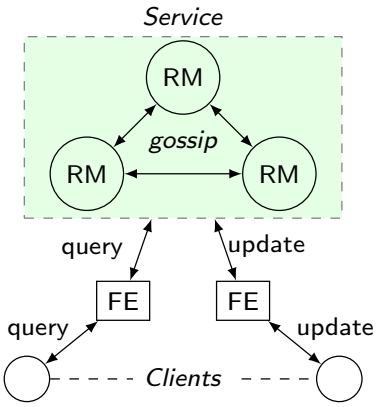

Figure 7.2: Gossip framework structure.

```
module RM1 controls s1
init
:: true ~> s1':=true;
update
:: s1 ~> s1':=false;
:: s1 ~> s1':=true;
:: !s1 and (!s2 or ... or !sn)
   ~> s1':=true;
goal
:: G F (!s1);
```

Figure 7.3: SRML code modelling RM_1.

in many large-scale distributed systems, such as *peer-to-peer* and *cloud* computing systems. Ladin *et al.* [Ladin et al., 1992] developed a framework to provide high availability services via replication which is based on the gossip approach first introduced in [Fischer and Michael, 1982, Wuu and Bernstein, 1984]. The main feature of this framework is the use of *replica managers* (RMs) which exchange "gossip" messages periodically in order to keep the data updated. The architecture of such an approach is shown in Figure 7.2.

We can model each RM as a module in SRML as follows: (1) When in *servicing mode*, an RM can choose either to keep in servicing mode or to switch to gossiping mode; (2) If it is in gossiping mode and there is at least another RM also in gossiping mode[2], since the information during gossip exchange is of (small) bounded size, it goes back to servicing mode in the subsequent step. We then set the goal of each RM to be able to gossip infinitely often. As shown in Figure 7.3, the module `RM1` controls a variable: `s1`. Its value being true signifies that `RM1` is in servicing mode; otherwise,

[2]The core of the protocol involves (at least) pairwise interactions periodically.

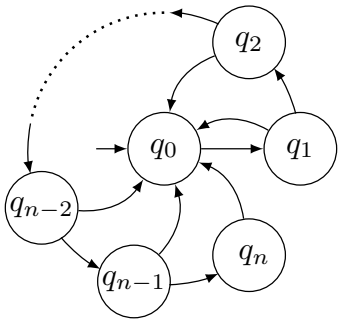

Figure 7.4: Gifford's protocol modelled as a game.

it is in gossiping mode. Behaviour (1) is reflected in the first and second update commands, while behaviour (2) is reflected in the third update command. The goal of `RM1` is specified with the `LTL` formula $\mathbf{GF} \neg$ `s1`, which expresses that `RM1`'s goal is to gossip infinitely often: "always" ($\mathbf{G}$) "eventually" ($\mathbf{F}$) gossip ($\neg$ `s1`).

Observe that with all RMs rationally pursuing their goals, they will adopt any strategy which induces a run where each RM can gossip (with at least one other RM) infinitely often. In fact, this kind of game-like modelling gives rise to a powerful characteristic: on *all* runs that are sustained by a Nash equilibrium, the distributed system is guaranteed to have two crucial *non-starvation/liveness* properties: RMs can gossip infinitely often and clients can be served infinitely often. Indeed, these properties are verified in the experiments; with E-Nash: no Nash equilibrium sustains "all RMs forever gossiping"; and with A-Nash: in all Nash equilibria at least one of the RM is in servicing mode infinitely often. We also notice that each RM is modelled as a non-deterministic open system: non-determinism is used in the first two updated commands, as they have the same guard `s1` and therefore will be both enabled at the same time; and the system is open since each module's state space and choices depend on the states of other modules, as reflected by the third updated command.

7.3.2 Replica Control

Consensus is a key issue in distributed computing and multi-agent systems. An important application domain is in maintaining data consistency. Gifford [Gifford, 1979] proposed a quorum-based voting protocol to ensure data consistency by not allowing more than one processes to read/write a data item concurrently. To do this, each copy of a replicated item is assigned a vote.

We can model a (modified version of) Gifford's protocol as a game as follows. The set of players N = $\{1, \ldots, n\}$ in the game is arranged in a request queue represented by the sequence of states $q_1, \ldots, q_n$, where q_i means that player i is requesting to

read/write the data item. At state q_i, other players in N\{i} then can vote whether to allow player i to read/write. If the majority of players in N vote "yes", then the transition goes to q_0, *i.e.*, player i is allowed to read/write, and otherwise it goes to q_{i+1}[3]. The voting process then restarts from q_1. The protocol's structure is shown in Figure 7.4. Notice that at the last state, q_n, there is only one outgoing arrow to q_0. As in the previous example, the goal of each player i is to visit q_0 right after q_i infinitely often, so that the desired behaviour of the system is sustained on all Nash equilibria of the system: a data item is not concurrently accessed by two different processes and the data is updated in *every* round. The associated temporal properties are automatically verified in the experiments in Section 7.4.1. Specifically, the temporal properties we check are as follows. With E-Nash: there is no Nash equlibrium in which the data is never updated; and, with A-Nash: on all Nash equilibria, for each player, its request will be granted infinitely often. Also, in this example, we define a module, called "Environment", which is used to represent the underlying concurrent game structure, shown in Figure 7.4, where the game is played.

7.4 Evaluation

7.4.1 Experiment I

In order to evaluate the practical preformance of our tool and approach (against MCMAS and PRALINE), we present results on the temporal equilibrium analysis for the examples in Section 7.3. We ran the tools on the two examples with different numbers of players ("P"), states ("S"), and edges ("E"). The experiments were obtained on a PC with Intel i5-4690S CPU 3.20 GHz machine with 8 GB of RAM running Linux kernel version 4.12.14-300.fc26.x86-64. We report the running time[4] for solving Non-Emptiness ("ν"), E-Nash ("ϵ"), and A-Nash ("α"). For the last two problems, since there is no direct support in PRALINE and MCMAS, we used the reduction of E/A-Nash to Non-Emptiness presented in [Gao et al., 2017]. Time-out ("TO") was fixed to be 7200 seconds.

From the experiment results shown in Table 7.1 and 7.2, we observe that, in general, EVE has the best performance, followed by PRALINE and MCMAS. Although PRALINE performed better than MCMAS, both struggled (timed-out) with inputs

[3]We assume arithmetic modulo ($|\mathrm{N}|+1$) in this example.

[4]To carry out a fairer comparison (since PRALINE does not accept LTL goals), we added to PRALINE's running time the time needed to convert LTL games into its input.

Table 7.1: Gossip Protocol experiment results.

P	S	E	**EVE**			**PRALINE**			**MCMAS**		
			ν (s)	ϵ (s)	α (s)	ν (s)	ϵ (s)	α (s)	ν (s)	ϵ (s)	α (s)
2	4	9	0.02	0.24	0.08	0.02	1.71	1.73	**0.01**	**0.01**	**0.01**
3	8	27	0.09	0.43	0.26	0.33	26.74	27.85	**0.02**	**0.06**	**0.06**
4	16	81	**0.42**	**3.51**	**1.41**	0.76	547.97	548.82	760.65	3257.56	3272.57
5	32	243	**2.30**	**35.80**	**25.77**	10.06	TO	TO	TO	TO	TO
6	64	729	**16.63**	**633.68**	**336.42**	255.02	TO	TO	TO	TO	TO
7	128	2187	**203.05**	TO	TO	5156.48	TO	TO	TO	TO	TO
8	256	6561	**4697.49**	TO	TO	TO	TO	TO	TO	TO	TO

Table 7.2: Replica control experiment results.

P	S	E	**EVE**			**PRALINE**			**MCMAS**		
			ν (s)	ϵ (s)	α (s)	ν (s)	ϵ (s)	α (s)	ν (s)	ϵ (s)	α (s)
2	3	8	0.04	0.11	0.10	0.05	0.64	0.74	**0.01**	**0.01**	**0.02**
3	4	20	0.11	1.53	0.22	0.12	4.96	5.46	**0.02**	**0.06**	**0.11**
4	5	48	**0.34**	**1.73**	**0.68**	0.56	65.50	67.45	1.99	4.15	11.28
5	6	112	**1.43**	**2.66**	**2.91**	6.86	1546.90	1554.80	1728.73	6590.53	TO
6	7	256	**5.87**	**13.69**	**16.03**	94.39	TO	TO	TO	TO	TO
7	8	576	**32.84**	**76.50**	**102.12**	2159.88	TO	TO	TO	TO	TO
8	9	1280	**166.60**	**485.99**	**746.55**	TO	TO	TO	TO	TO	TO

with more than 100 edges, while EVE could handle up to 6000 edges (for Non-Emptiness).

7.4.2 Experiment II

This experiment is taken from the motivating examples in [Gutierrez et al., 2017a]. In this experiment, unlike in previous ones, EVE manages to compute a Nash equilibrium in bisimulation-invariant strategies, while PRALINE and MCMAS do not. In this experiment, we extended the number of states by adding more layers to the game structures used there in order to test the practical performance of EVE, MCMAS, and PRALINE. The experiments were performed on a PC with Intel i7-4702MQ CPU 2.20GHz machine with 12GB of RAM running Linux kernel version 4.14.16-300.fc26.x86-64. We divided the test cases based on the number of Kripke states and

Table 7.3: Example with no Nash equilibrium.

states	edges	MCMAS		EVE		PRALINE	
		time (s)	NE	time (s)	NE	time (s)	NE
5	80	**0.04**	No	0.75	Yes	0.77	No
8	128	**0.24**	No	2.99	Yes	2.06	No
11	176	6.28	No	**3.86**	Yes	4.42	No
14	224	273.14	No	**7.46**	Yes	8.53	No
17	272	TO	–	**13.31**	Yes	15.33	No
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
50	800	TO	–	**655.80**	Yes	789.77	No

Table 7.4: Example with Nash equilibria

states	edges	MCMAS		EVE		PRALINE	
		time (s)	NE	time (s)	NE	time (s)	NE
6	96	**0.02**	Yes	1.09	Yes	1.19	Yes
9	144	**0.77**	Yes	3.36	Yes	3.76	Yes
12	192	65.31	No	**7.45**	Yes	8.89	Yes
15	240	TO	–	**15.52**	Yes	17.72	Yes
18	288	TO	–	**30.06**	Yes	30.53	Yes
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
51	816	TO	–	**1314.47**	Yes	1563.79	Yes

edges; then, for each case, we report (i) the total running time[5] ("time") and (ii) whether the tools find any Nash equilibria ("NE").

Table 7.3 shows the results of the experiments on the example in which the model of strategies that depends only on the run (sequence of states) of the game (called run-based strategies in [Gutierrez et al., 2017a]) cannot sustain any Nash equilibria, a model of strategies that is not invariant under bisimilarity. Indeed, since MCMAS and PRALINE use this model of strategies, both did not find any Nash equilibria in the game, as shown in Table 7.3. EVE, which uses a model of strategies that not only depends on the run of the game but also on the actions of players (a bisimulation-invariant model of strategies called computation-based in [Gutierrez et al., 2017a]), found a Nash equilibrium in the game. We can also see that EVE outperformed

[5]Similarly to Experiment I (Section 7.4.1), we added to PRALINE's running time the time needed to convert LTL games into its input to carry out a fairer comparison.

MCMAS on games with 14 or more states. In fact, MCMAS timed-out[6] on games with 17 states or more, while EVE kept working efficiently for games of bigger size. We can also observe that PRALINE performed almost as efficiently as EVE in this experiment, although EVE performed better in both small and large instances of these games.

In Table 7.4, we used the example in which Nash equilibria is sustained in run-based strategies. As shown in the table, MCMAS found Nash equilibria in games with 6 and 9 states. However, since MCMAS uses imperfect recall, when the third layer was added (case with 12 states in Table 7.4) to the game, it could not find any Nash equilibria. Regarding running times, EVE outperformed MCMAS from the game with 12 states and beyond, where MCMAS timed-out on games with 15 or more states. As for PRALINE, it performed comparably to EVE in this experiment, but again, EVE performed better in all instances.

7.4.3 Experiment III

In this experiment, we have two agents inhabiting a grid world with dimensions $n \times n$. Initially, the agents are located at opposing corners of the grid; specifically, agent 1 is located at the top-left corner (coordinate $(0, 0)$) and agent 2 at the bottom-right corner $(n-1, n-1)$. The agents are each able to move around the grid in directions *north*, *south*, *east*, and *west*. The goal of each agent is to reach the opposite corner, that is, agent 1's goal is to reach position $(n-1, n-1)$, and agent 2's goal is to reach position $(0, 0)$. A number of obstacles are also placed (uniformly) randomly on the grid. The agents are not allowed to move into a coordinate occupied by an obstacle, the other agent, or outside the grid world. We used a binary encoding to represent the spatial information of the grid world which includes the grid coordinates, as well as the obstacles and the agents locations.

To make it clearer, consider the example shown in Figure 7.5; a (grey) filled square depicts an obstacle. Agent 1, depicted by ■, can move north to $(2, 0)$, south to $(2, 2)$, east to $(3, 1)$, and west to $(1, 1)$. Whereas agent 2, depicted by ◯, can only move north to $(0, 1)$ and south to $(0, 3)$ (she cannot move west because it is outside the world, nor east because there is an obstacle.)

In this experiment we make the following assumptions: (1) at each timestep, each agent has to make a move, that is, she cannot stay at the same position for two consecutive timesteps, and she can only move at most one step; (2) the goal of each

[6]We fixed the time-out value to be 3600 seconds (1 hour).

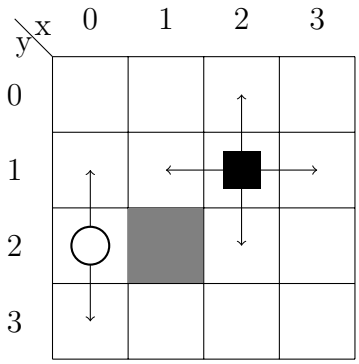

Figure 7.5: Example of a 4×4 grid world.

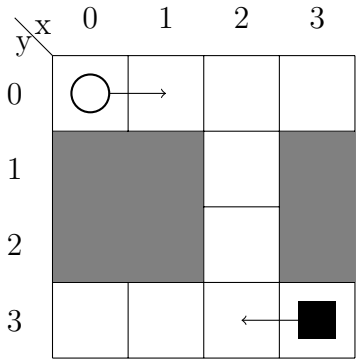

Figure 7.6: Example of a 4×4 grid world without "safe" Nash equilibria.

agent is, as stated previously, to eventually reach the opposite corner of her initial position. From system design point of view, the question that may be asked is: can we synthesise a strategy profile such that it induces a stable (Nash equilibrium) run and at the same time ensures that the agents never crash into each other? We can translate this question into an E-Nash instance with the property to be checked is "two agents never occupying the same coordinate at the same time", in other words, two agents never crash into each other.

Checking the existence of such a strategy profile is not trivial. For instance, the

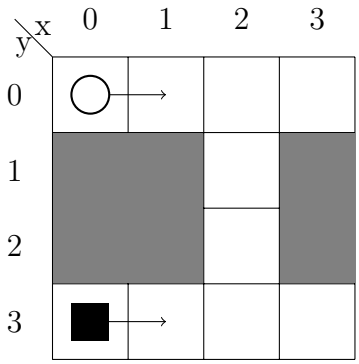

Figure 7.7: A 4×4 grid world with safe Nash equilibrium.

Table 7.5: Grid world experiment results.

Size	# Obs	KS	KE	GS
3	3	15(13, 18)	44(32, 72)	60(53, 73)
4	6	40(32, 52)	150(98, 200)	156(121, 209)
5	10	94(61, 125)	398(242, 512)	376(453, 741)
6	15	155(113, 185)	655(450, 800)	619(453, 741)
7	21	228(181, 290)	994(800, 1250)	909(725, 1161)
8	28	491(394, 666)	2297(1922, 2888)	1963(1577, 2665)
9	36	564(269, 765)	2687(1352, 3698)	2256(1077, 3061)
10	45	916(730, 1258)	4780(3528, 6498)	3657(2921, 5033)

Size	GE	ν (s)	ϵ (s)
3	173(129, 289)	0.44(0.19, 1.14)	1.21(0.5, 2.63)
4	595(379, 801)	0.98(0.63, 1.16)	1.57(1.01, 2.24)
5	1591(969, 2049)	4.73(2.62, 6.22)	22.51(18.22, 26.25)
6	2622(1801, 3201)	9.53(7.13, 11.49)	32.32(26.05, 37.35)
7	3969(3161, 5001)	17.69(13.81, 21.58)	48.90(39.70, 59.50)
8	9190(7689, 11553)	50.91(38.38, 72.49)	121.33(95.03, 167.25)
9	10748(5409, 14793)	100.94(45.81, 137.91)	6002.80(5477.63, 6374.26)
10	19102(14113, 25993)	211.30(152.74, 311.43)	6871.16(6340.64, 7650.87)

configuration in Figure 7.6 does not admit any safe Nash equilibrium runs, that is, where all agents get their goals achieved without crashing into each other. On the other hand, the configuration in Figure 7.7, admits safe Nash equilibrium. Thus, having a tool to verify and synthesise such scenario is desirable.

The experiment was obtained on a PC with Intel i5-4690S CPU 3.20 GHz machine with 8 GB of RAM running Linux kernel version 4.12.14-300.fc26.x86-64. We varied the size of the grid world ("size") from 3×3 to 10×10, each with a fixed number of obstacles ("# Obs"), randomly distributed on the grid. We report the number Kripke states ("KS"), Kripke edges ("KE"), $\mathcal{G}_{\text{PAR}}$ states ("GS"), $\mathcal{G}_{\text{PAR}}$ edges ("GE"), Non-Emptiness execution time ("ν"), and E-Nash execution time ("ϵ"). We ran the experiment for five replications, and report the average (*ave*), minimum (*min*), and maximum (*max*) times from the replications. The results are reported in Table 7.5, with the following format: $ave(min, max)$.

From the experiment results, we see that EVE works well for Non-Emptiness up until size 10. From the plots in Figure 7.8, we can clearly see that the values of each variable, except for ϵ, grow exponentially. For ϵ (E-Nash), however, it seems to grow

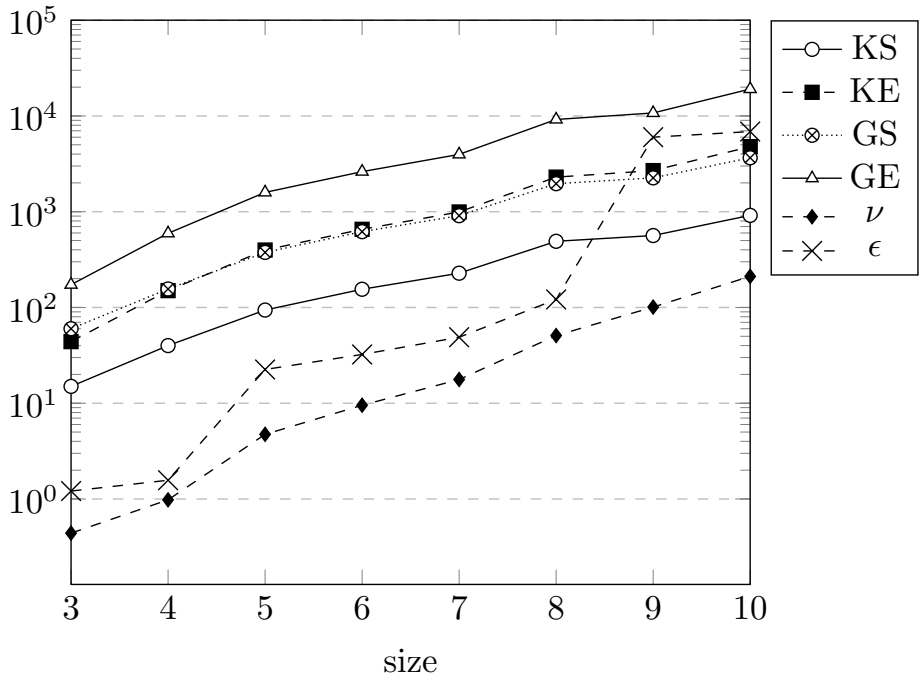

Figure 7.8: Plots from Table 7.5. Y-axis is in logarithmic scale.

faster than the rest. Specifically, it is clearly visible in transitions between numbers that have different size of bit representation, *i.e.*, 4 to 5 and 8 to 9[7]. These jumps correspond to the time used to build deterministic parity automata on words from LTL properties to be checked in E-Nash, which is essentially, bit-for-bit comparisons between the position of agent 1 and 2.

[7]Since the grid coordinate index starts at 0, the "actual" transitions are 3 to 4 and 7 to 8.

Chapter 8

Conclusions

This chapter concludes the main part of the thesis by reviewing the contributions that have been put forward, then providing some discussion to put the work in this thesis into larger context, and finally presenting some pointers to possible future research directions.

8.1 Contributions

The aim of this thesis has been the development of techniques and tools for rational verification, as well as the study of some tractable cases of rational verification and equilibrium design, of multi-agent systems. The main contributions to this area of research are summarised below:

- The theoretical contributions of this thesis are as follows:
  - Development of algorithmic techniques to solve key decision problems in the rational verification framework. The techniques used in this thesis employ parity conditions to reasons about game with perfect recall semantics. In particular, in Chapter 4, we prove that our construction of concurrent parity game preserves the set of Nash equilibria in the original game (Theorem 8). With this result, we then provide Nash equilibrium characterisation (Theorem 11) by reasoning via a collection of (turn-based) parity games. The algorithmic approach is reported in Algorithm 9. The approach is efficient in a sense that it matches the theoretical lower-bound of 2EXPTIME;
  - Identification of computationally tractable cases for rational verification. As reported in Chapter 5, by using a fragment of LTL, we can decrease the

complexity of E-Nash problem to PSPACE-complete (Corollary 14), and even to fixed-parameter tractable (Theorem 15). We also introduce games with mean-payoff objective and prove that the combination of mean-payoff goals with LTL specification, and mean-payoff goals with GR(1) specification, allow us to decrease the complexity to PSPACE-complete (Corollary 17) and NP-complete (Theorem 18), respectively;

  - Development of the concept of equilibrium design for multi-agent systems. We introduce two decision problems called Weak Implementation and Strong Implementation in Chapter 6 and prove that with GR(1) specification, the complexities belong to NP-complete (Theorem 20) and Σ_2^{P}-complete (Theorem 22), respectively. We further prove that the extensions of the aforementioned problems, such as the optimality (Theorems 25 and 32), exactness (Corollaries 26 and 33), and uniqeness (Corollaries 28 and 35), all lie within the polynomial hierarchy.

- Development of EVE (Equilibrium Verification Environment): EVE is a tool developed in Python that can be used for the automated analysis of temporal equilibrium properties of concurrent and multi-agent systems. Systems are modelled using SRML as a collection of independent components (players/agents in a game) and players' goals are expressed using LTL formulae.

- Application examples: various multi-agent systems scenarios have been modelled in SRML and checked by EVE, including communication and networking protocols, bisimilar concurrent-game structures, and multi-robot-like systems. Experimental results have been presented, and performance comparison to other tools (MCMAS and PRALINE) also showed the advantage of our approach (EVE).

The main results of this thesis are also contained in: [Gutierrez et al., 2019b, Gutierrez et al., 2018a, Gutierrez et al., 2019d, Gutierrez et al., 2019c]. Also, the tool EVE can be found at: `http://eve.cs.ox.ac.uk/`.

8.2 Discussion

Equilibrium Analysis in Multi-Agent Systems. Over the past decade, there has been increasing interest in the use of game-theoretic equilibrium concepts such as Nash equilibrium in the analysis of concurrent and multi-agent systems (see, *e.g.*, [Almagor et al., 2018, Aminof et al., 2016, Bouyer et al., 2015a, Fisman et al.,

2010, Gutierrez et al., 2017a, Gutierrez et al., 2017b, Kupferman et al., 2016]). This work, as well as the work carried out in this thesis, views a concurrent system as a game, with system components (agents) corresponding to players in the game, which are assumed to be acting rationally in pursuit of their individual preferences. Preferences may be specified by associating with each player a temporal logic goal formula, which the player desires to see satisfied, or by assuming that players receive rewards in each state the system visits, and seek to maximise the average reward they receive (the *mean payoff*).

The key decision problems in such settings relate to what temporal logic properties hold on computations of the system that may be generated by players choosing strategies that form a game-theoretic (Nash) equilibrium. These problems are typically computationally complex, since they subsume temporal logic synthesis [Pnueli and Rosner, 1989]. If players have LTL goals, for example, then checking whether an LTL formula holds on some Nash equilibrium path in a concurrent game is 2EXPTIME-complete [Fisman et al., 2010, Gutierrez et al., 2015b, Gutierrez et al., 2017b], rather than only PSPACE-complete as it is the case for model checking, certainly a computational barrier for the practical analysis and automated verification of reactive, concurrent, and multi-agent systems modelled as multi-player games. Furthermore, from verification point of view, one obvious question is whether assuming agents to be perfectly rational is a reasonable assumption. If we view the agents as *people*, then clearly using the concept of Nash equilibrium (with perfect rationality assumption) is not appropriate [Mailath, 1998, Gigerenzer and Selten, 2002]. However, when we consider the agents as *computer programs*, in principle, we can design them to act rationally. Although, it should be noted that the expensive computational price of rational verification (as well as, synthesis), may indicate that other (weaker) solution concepts (e.g., approximate Equilibrium) are more appropriate with resource bounded agents.

Automata and logic. In computer science, a common technique to reason about Nash equilibria in multi-player games is using alternating parity *automata on infinite trees* (APTs [Löding, 2012]). This approach is used to do rational synthesis [Fisman et al., 2010, Kupferman et al., 2016]; equilibrium checking and rational verification [Wooldridge et al., 2016, Gutierrez et al., 2015b, Gutierrez et al., 2017b]; and model checking of logics for strategic reasoning capable to specify the existence of a Nash equilibrium in concurrent game structures [Alur et al., 2002], both in two-player games [Chatterjee et al., 2010b, Finkbeiner and Schewe, 2010] and in multi-player

games [Laroussinie and Markey, 2015, Mogavero et al., 2014]. In cases where players' goals are simpler than general LTL formulae, *e.g.*, for reachability or safety goals, alternating Büchi automata can be used instead [Bouyer et al., 2015a]. The technique developed in this thesis is different from all these automata-based approaches, and in some cases more general, as it can be used to handle either a more complex model of strategies or a more complex type of goals, and delivers an immediate procedure to synthesise individual strategies for players in the game, while being amenable to implementation.

Equilibrium design vs. mechanism design – connections with Economic theory. Although equilibrium design is closely related to mechanism design, as typically studied in game theory [Hurwicz and Reiter, 2006], the two are not exactly the same. Two key features in mechanism design are the following. Firstly, in a mechanism design problem, the designer is not given a game structure, but instead is asked to provide one; in that sense, a mechanism design problem is closer to a rational synthesis problem [Fisman et al., 2010, Gutierrez et al., 2015b]. Secondly, in a mechanism design problem, the designer is only interested in the game's outcome, which is given by the payoffs of the players in the game; however, in equilibrium design, while the designer is interested in the payoffs of the players as these may need to be perturbed by its budget, the designer is also interested – and in fact primarily interested – in the satisfaction of a temporal logic goal specification, which the players in the game do not take into consideration when choosing their individual rational choices; in that sense, equilibrium design is closer to rational verification [Gutierrez et al., 2017b] than to mechanism design. Thus, equilibrium design is a new computational problem that sits somewhere in the middle between mechanism design and rational verification/synthesis. Technically, in equilibrium design we go beyond rational synthesis and verification through the additional design of subsidy schemes for incentivising behaviours in a concurrent and multi-agent system, but we do not require such subsidy schemes to be incentive compatible mechanisms, as in mechanism design theory, since the principal may want to reward only a group of players in the game so that its temporal logic goal is satisfied, while rewarding other players in the game in an unfair way – thus, leading to a game with a suboptimal social welfare measure. In this sense, equilibrium design falls short with respect to the more demanding social welfare requirements often found in mechanism design theory.

Equilibrium design vs. rational verification – connections with Computer science. Typically, in rational synthesis and verification [Fisman et al., 2010, Gutierrez et al., 2015b, Gutierrez et al., 2017b, Kupferman et al., 2016] we want to check whether a property is satisfied on some/every Nash equilibrium computation run of a reactive, concurrent, and multi-agent system. These verification problems are primarily concerned with qualitative properties of a system, while assuming rationality of system components. However, little attention is paid to quantitative properties of the system. This drawback has been recently identified and some work has been done to cope with questions where both qualitative and quantitative concerns are considered [Almagor et al., 2018, Bohy et al., 2013, Chatterjee and Doyen, 2012, Chatterjee et al., 2010a, Chatterjee et al., 2005, Gutierrez et al., 2017c, Velner et al., 2015]. Equilibrium design is new and different approach where this is also the case. More specifically, as in a mechanism design problem, through the introduction of an external principal – the designer in the equilibrium design problem – we can account for overall qualitative properties of a system (the principal's goal given by an LTL or a GR(1) specification) as well as for quantitative concerns (optimality of solutions constrained by the budget to allocate additional rewards/resources). Our framework also mixes qualitative and quantitative features in a different way: while system components are only interested in maximising a quantitative payoff, the designer is primarily concerned about the satisfaction of a qualitative (logic) property of the system, and only secondarily about doing it in a quantitatively optimal way.

Equilibrium design vs. repair games and normative systems – connections with AI. In recent years, there has been an interest in the analysis of rational outcomes of multi-agent systems modelled as multi-player games. This has been done both with modelling and with verification purposes. In those multi-agent settings, where AI agents can be represented as players in a multi-player game, a focus of interest is on the analysis of (Nash) equilibria in such games [Bouyer et al., 2015a, Gutierrez et al., 2017b]. However, it is often the case that the existence of Nash equilibria in a multi-player game with temporal logic goals may not be guaranteed [Gutierrez et al., 2015b, Gutierrez et al., 2017b]. For this reason, there has been already some work on the introduction of desirable Nash equilibria in multi-player games [Almagor et al., 2015, Perelli, 2019]. This problem has been studied as a repair problem [Almagor et al., 2015] in which either the preferences of the players (given by winning conditions) or the actions available in the game are modified; the latter one also being achieved with the use of normative systems [Perelli, 2019]. In equilibrium design, we

do not directly modify the preferences of agents in the system, since we do not alter their goals or choices in the game, but we indirectly influence their rational behaviour by incentivising players to visit, or to avoid, certain states of the overall system. We studied how to do this in an (individually) optimal way with respect to the preferences of the principal in the equilibrium design problem. However, this may not always be possible, for instance, because the principal's temporal logic specification goal is just not achievable, or because of constraints given by its limited budget.

8.3 Future Work

Chapter 4 gives a solution to the temporal equilibrium problem (both automated synthesis and formal verification) in a noncooperative setting. In future work, we plan to investigate the cooperative games setting [Ågotnes et al., 2009]. We also plan to investigate if our main algorithms can be extended to decidable classes of imperfect information games, for instance, as those studied to model the behaviour of multi-agent systems in [Gutierrez et al., 2018b, Belardinelli et al., 2017, Aminof et al., 2014, Berthon et al., 2017]. Considering other solution concepts, such as subgame perfect Equilibrium (and its associated refinement concepts) may also be an obvious future direction. From performance point of view, one may also consider employing randomised algorithm, to carry out rational verification (c.f., [Grosu and Smolka, 2005]). Whenever possible, such studies will be complemented with practical implementations in EVE. Finally, extensions to epistemic systems and quantitative information in the context of multi-agent systems may be another avenue for further applications [Herzig et al., 2016, Belardinelli and Lomuscio, 2009].

In Chapter 5, we have identified some tractable cases for rational verification. The existence of such cases is a good news from practical point of view. In the future, it is worthwile to consider extending EVE to implement such algorithms.

As discussed in Chapter 6, a key difference with mechanism design is that social welfare requirements are not considered [Maschler et al., 2013]. However, a benevolent principal might not see optimality as an individual concern, and instead consider the welfare of the players in the design of a subsidy scheme. In that case, concepts such as the *utilitarian social welfare* may be undesirable as the social welfare maximising the payoff received by players might allocate all the budget to only one player, and none to the others. A potentially better option is to improve fairness in the allocation of the budget by maximising the *egalitarian social welfare*. Finally, given that the

complexity of equilibrium design is much better than that of rational synthesis/verification, we should be able to have efficient implementations, for instance, as an extension of EVE [Gutierrez et al., 2018a].

Bibliography

[ari, 1996] (1996). ARIANE 5 failure - full report. `http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html`. Accessed: 2017-05-15.

[lis, 2016] (2016). A collection of well-known software failures. `http://www.cse.psu.edu/~gxt29/bug/softwarebug.html`. Accessed: 2017-05-15.

[Ågotnes et al., 2009] Ågotnes, T., van der Hoek, W., and Wooldridge, M. (2009). Reasoning about coalitional games. *Artificial Intelligence*, 173(1):45–79.

[Agotnes and Walther, 2009] Agotnes, T. and Walther, D. (2009). A logic of strategic ability under bounded memory. *Journal of Logic, Language, and Information*, 18(1):55–77.

[Aleksandrowicz et al., 2017] Aleksandrowicz, G., Chockler, H., Halpern, J. Y., and Ivrii, A. (2017). The computational complexity of structure-based causality. *Journal of Artificial Intelligence Research*, 58(1):431–451.

[Almagor et al., 2015] Almagor, S., Avni, G., and Kupferman, O. (2015). Repairing Multi-Player Games. In *26th International Conference on Concurrency Theory (CONCUR 2015)*, volume 42 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 325–339.

[Almagor et al., 2018] Almagor, S., Kupferman, O., and Perelli, G. (2018). Synthesis of controllable nash equilibria in quantitative objective game. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 35–41.

[Alur et al., 2001] Alur, R., de Alfaro, L., Grosu, R., Henzinger, T. A., Kang, M., Kirsch, C. M., Majumdar, R., Mang, F., and Wang, B. Y. (2001). jmocha: a model checking tool that exploits design structure. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pages 835–836.

[Alur and Henzinger, 1994] Alur, R. and Henzinger, T. A. (1994). A really temporal logic. *Journal of the ACM*, 41(1):181–203.

[Alur and Henzinger, 1999] Alur, R. and Henzinger, T. A. (1999). Reactive modules. *Formal Methods in System Design*, 15(11):7–48.

[Alur et al., 2002] Alur, R., Henzinger, T. A., and Kupferman, O. (2002). Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713.

[Alur et al., 1998a] Alur, R., Henzinger, T. A., Kupferman, O., and Vardi, M. Y. (1998a). Alternating refinement relations. In *CONCUR*, volume 1466 of *LNCS*, pages 163–178. Springer.

[Alur et al., 1998b] Alur, R., Henzinger, T. A., Mang, F. Y. C., Qadeer, S., Rajamani, S. K., and Tasiran, S. (1998b). Mocha: Modularity in model checking. In *Proceedings of the 10th International Conference on Computer Aided Verification*, CAV '98, pages 521–525.

[Aminof et al., 2016] Aminof, B., Malvone, V., Murano, A., and Rubin, S. (2016). Graded strategy logic: Reasoning about uniqueness of nash equilibria. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, AAMAS 2016*, pages 698–706.

[Aminof et al., 2014] Aminof, B., Mogavero, F., and Murano, A. (2014). Synthesis of hierarchical systems. *Science of Computer Programming*, 83:56–79.

[Arnold and Crubille, 1988] Arnold, A. and Crubille, P. (1988). A linear algorithm to solve fixed-point equations on transition systems. *Information Processing Letters*, 29(2):57–66.

[Belardinelli and Lomuscio, 2009] Belardinelli, F. and Lomuscio, A. (2009). Quantified epistemic logics for reasoning about knowledge in multi-agent systems. *Artificial Intelligence*, 173(9-10):982–1013.

[Belardinelli et al., 2017] Belardinelli, F., Lomuscio, A., Murano, A., and Rubin, S. (2017). Verification of multi-agent systems with imperfect information and public actions. In *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '17, pages 1268–1276.

[Berthon et al., 2017] Berthon, R., Maubert, B., and Murano, A. (2017). Decidability results for atl* with imperfect information and perfect recall. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 1250–1258.

[Biere et al., 2003] Biere, A., Cimatti, A., Clarke, E. M., Strichman, O., and Zhu, Y. (2003). Bounded model checking. *Advances in Computers*, 58:117–148.

[Binmore, 1992] Binmore, K. (1992). *Fun and Games: A Text on Game Theory*. D. C. Heath and Company: Lexington, MA.

[Blackburn et al., 2001] Blackburn, P., de Rijke, M., and Venema, Y. (2001). *Modal Logic*. Cambridge University Press, New York, NY, USA.

[Bloem et al., 2012] Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., and Sa'ar, Y. (2012). Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78(3):911–938.

[Bohy et al., 2013] Bohy, A., Bruyère, V., Filiot, E., and Raskin, J. (2013). Synthesis from LTL Specifications with Mean-Payoff Objectives. In *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2013*, pages 169–184.

[Boker et al., 2014] Boker, U., Chatterjee, K., Henzinger, T. A., and Kupferman, O. (2014). Temporal Specifications with Accumulative Values. *ACM Transactions on Computational Logic*, 15(4):27:1–27:25.

[Bouyer et al., 2015a] Bouyer, P., Brenguier, R., Markey, N., and Ummels, M. (2015a). Pure nash equilibria in concurrent deterministic games. *Logical Methods in Computer Science*, 11(2):1–72.

[Bouyer et al., 2015b] Bouyer, P., Gardy, P., and Markey, N. (2015b). Weighted strategy logic with boolean goals over one-counter games. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015*, pages 69–83.

[Brafman et al., 1997] Brafman, R. I., Latombe, J.-C., Moses, Y., and Shoham, Y. (1997). Applications of a logic of knowledge to motion planning under uncertainty. *Journal of the ACM*, 44(5):633–668.

[Brenguier, 2013] Brenguier, R. (2013). Praline: A tool for computing nash equilibria in concurrent games. In *Proceedings of the 25th International Conference on Computer Aided Verification - Volume 8044*, CAV 2013, pages 890–895.

[Büchi, 1962] Büchi, J. R. (1962). On a decision method in restricted second order arithmetic. In Nagel, E., Suppes, P., and Tarski, A., editors, *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science (LMPS'60)*, pages 1–11.

[Burch et al., 1990] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. J. (1990). Symbolic model checking: 10ˆ20 states and beyond. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90)*, pages 428–439.

[Calude et al., 2017] Calude, C. S., Jain, S., Khoussainov, B., Li, W., and Stephan, F. (2017). Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 252–263.

[Cermák et al., 2014] Cermák, P., Lomuscio, A., Mogavero, F., and Murano, A. (2014). MCMAS-SLK: A model checker for the verification of strategy logic specifications. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, pages 525–532.

[Cermák et al., 2018] Cermák, P., Lomuscio, A., Mogavero, F., and Murano, A. (2018). Practical verification of multi-agent systems against slk specifications. *Information and Computation*, 261(Part):588–614.

[Chaochen et al., 1993] Chaochen, Z., Hansen, M. R., and Sestoft, P. (1993). Decidability and undecidability results for duration calculus. In *Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '93, pages 58–68.

[Chaochen et al., 1991] Chaochen, Z., Hoare, C., and Ravn, A. P. (1991). A calculus of durations. *Information Processing Letters*, 40(5):269 – 276.

[Chatterjee and Doyen, 2012] Chatterjee, K. and Doyen, L. (2012). Energy parity games. *Theoretical Computer Science*, 458:49–60.

[Chatterjee et al., 2010a] Chatterjee, K., Doyen, L., Henzinger, T., and Raskin, J. (2010a). Generalized mean-payoff and energy games. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010*, pages 505–516.

[Chatterjee et al., 2005] Chatterjee, K., Henzinger, T. A., and Jurdzinski, M. (2005). Mean-payoff parity games. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 178–187.

[Chatterjee et al., 2010b] Chatterjee, K., Henzinger, T. A., and Piterman, N. (2010b). Strategy logic. *Information and Computation*, 208(6):677–693.

[Chen et al., 2012] Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., and Simaitis, A. (2012). Automatic verification of competitive stochastic systems. In Flanagan, C. and König, B., editors, *Proc. 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214 of *LNCS*, pages 315–330.

[Chen et al., 2013a] Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., and Simaitis, A. (2013a). Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92.

[Chen et al., 2013b] Chen, T., Forejt, V., Kwiatkowska, M. Z., Parker, D., and Simaitis, A. (2013b). Prism-games: A model checker for stochastic multi-player games. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013*, volume 7795 of *LNCS*, pages 185–191. Springer.

[Chockler and Halpern, 2004] Chockler, H. and Halpern, J. (2004). Responsibility and Blame: A Structural-Model Approach. *Journal of Artificial Intelligence Research*, 22:93–115.

[Clarke and Emerson, 1981] Clarke, E. M. and Emerson, E. A. (1981). Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer-Verlag: Berlin, Germany.

[Clarke et al., 1986] Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263.

[Clarke et al., 2002] Clarke, E. M., Grumberg, O., and Peled, D. A. (2002). *Model Checking*. MIT Press, Cambridge, MA, USA.

[Daniele et al., 1999] Daniele, M., Giunchiglia, F., and Vardi, M. Y. (1999). Improved automata generation for linear temporal logic. In *Proceedings of the 11th International Conference on Computer Aided Verification*, CAV '99, pages 249–260.

[De Nicola and Vaandrager, 1990] De Nicola, R. and Vaandrager, F. (1990). Action versus state based logics for transition systems. In *Proceedings of the LITP Spring School on Theoretical Computer Science on Semantics of Systems of Concurrent Processes*, pages 407–419.

[De Nicola and Vaandrager, 1995] De Nicola, R. and Vaandrager, F. W. (1995). Three logics for branching bisimulation. *Journal of the ACM*, 42(2):458–487.

[Dima and Tiplea, 2011] Dima, C. and Tiplea, F. L. (2011). Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoRR*, abs/1102.4225.

[Emerson, 1990] Emerson, E. A. (1990). Temporal and modal logic. In *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, pages 996–1072. Elsevier Science Publishers B.V.: Amsterdam.

[Emerson and Clarke, 1980] Emerson, E. A. and Clarke, E. M. (1980). Characterizing correctness properties of parallel programs using fixpoints. In *Automata, Languages and Programming, 7th Colloquium, ICALP*, pages 169–181.

[Emerson and Halpern, 1986] Emerson, E. A. and Halpern, J. Y. (1986). "sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178.

[Emerson and Jutla, 1991] Emerson, E. A. and Jutla, C. S. (1991). Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 368–377.

[Engelhardt et al., 2000] Engelhardt, K., Meyden, R. v. d., and Moses, Y. (2000). A program refinement framework supporting reasoning about knowledge and time. In *Proceedings of the Third International Conference on Foundations of Software Science and Computation Structures*, FOSSACS '00, pages 114–129.

[Etessami and Holzmann, 2000] Etessami, K. and Holzmann, G. J. (2000). Optimizing büchi automata. In *Proceedings of 11th International Conference Concurrency Theory, CONCUR 2000*, pages 153–167.

[Fagin et al., 1995] Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning About Knowledge*. The MIT Press: Cambridge, MA.

[Finkbeiner and Schewe, 2010] Finkbeiner, B. and Schewe, S. (2010). Coordination logic. In *Proceedings of 24th International Workshop on Computer Science Logic, CSL*, pages 305–319.

[Fisac et al., 2019] Fisac, J. F., Bronstein, E., Stefansson, E., Sadigh, D., Sastry, S. S., and Dragan, A. D. (2019). Hierarchical game-theoretic planning for autonomous vehicles. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 9590–9596.

[Fischer and Michael, 1982] Fischer, M. J. and Michael, A. (1982). Sacrificing serializability to attain high availability of data in an unreliable network. In *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, PODS '82, pages 70–75.

[Fisman et al., 2010] Fisman, D., Kupferman, O., and Lustig, Y. (2010). Rational synthesis. In *16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2010*, pages 190–204.

[Friedmann and Lange, 2010] Friedmann, O. and Lange, M. (2010). The pgsolver collection of parity game solvers – version 3.

[Fritz and Wilke, 2002] Fritz, C. and Wilke, T. (2002). State space reductions for alternating büchi automata. In *Proceedings of the 22nd Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, pages 157–168.

[Gammie and van der Meyden, 2004] Gammie, P. and van der Meyden, R. (2004). MCK: model checking the logic of knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification, CAV*, pages 479–483.

[Gao et al., 2017] Gao, T., Gutierrez, J., and Wooldridge, M. J. (2017). Iterated boolean games for rational verification. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*, pages 705–713.

[Garey et al., 1976] Garey, M. R., Graham, R. L., and Johnson, D. S. (1976). Some np-complete geometric problems. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, STOC*, pages 10–22.

[Gastin and Oddoux, 2001] Gastin, P. and Oddoux, D. (2001). Fast LTL to büchi automata translation. In *Proceedings of the 13th International Conference on Computer Aided Verification, CAV 2001*, pages 53–65.

[Gastin and Oddoux, 2019] Gastin, P. and Oddoux, D. (2019). LTL 2 BA: fast translation from ltl formulae to büchi automata. `http://www.lsv.fr/~gastin/ltl2ba/`. Accessed: 09-09-2019.

[Gerth et al., 1995] Gerth, R., Peled, D. A., Vardi, M. Y., and Wolper, P. (1995). Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th International Symposium on Protocol Specification, Testing and Verification, PSTV*, pages 3–18.

[Giannakopoulou and Lerda, 2002] Giannakopoulou, D. and Lerda, F. (2002). From states to transitions: Improving translation of LTL formulae to büchi automata. In *Proceedings of the 22nd International Conference on Formal Techniques for Networked and Distributed Systems - FORTE 2002*, pages 308–326.

[Gifford, 1979] Gifford, D. K. (1979). Weighted voting for replicated data. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, SOSP '79, pages 150–162.

[Gigerenzer and Selten, 2002] Gigerenzer, G. and Selten, R., editors (2002). *Bounded Rationality: The Adaptive Toolbox*, volume 1. The MIT Press, 1 edition.

[Grosu and Smolka, 2005] Grosu, R. and Smolka, S. A. (2005). Monte carlo model checking. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'05, pages 271–286.

[Gutierrez et al., 2017a] Gutierrez, J., Harrenstein, P., Perelli, G., and Wooldridge, M. (2017a). Nash equilibrium and bisimulation invariance. In *28th International Conference on Concurrency Theory, CONCUR 2017*, pages 17:1–17:16.

[Gutierrez et al., 2019a] Gutierrez, J., Harrenstein, P., Perelli, G., and Wooldridge, M. J. (2019a). Nash equilibrium and bisimulation invariance. *Logical Methods in Computer Science*, 15(3).

[Gutierrez et al., 2013] Gutierrez, J., Harrenstein, P., and Wooldridge, M. (2013). Iterated boolean games. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 932–938.

[Gutierrez et al., 2014] Gutierrez, J., Harrenstein, P., and Wooldridge, M. (2014). Reasoning about equilibria in game-like concurrent systems. In *Proceedings of the 14th International Conference Principles of Knowledge Representation and Reasoning, KR 2014*.

[Gutierrez et al., 2015a] Gutierrez, J., Harrenstein, P., and Wooldridge, M. (2015a). Expresiveness and complexity results for strategic reasoning. In *Proceedings of the 26th International Conference on Concurrency Theory, CONCUR 2015*, pages 268–282.

[Gutierrez et al., 2015b] Gutierrez, J., Harrenstein, P., and Wooldridge, M. (2015b). Iterated Boolean Games. *Information and Computation*, 242:53–79.

[Gutierrez et al., 2017b] Gutierrez, J., Harrenstein, P., and Wooldridge, M. (2017b). From model checking to equilibrium checking: Reactive modules for rational verification. *Artificial Intelligence*, 248:123 – 157.

[Gutierrez et al., 2017c] Gutierrez, J., Murano, A., Perelli, G., Rubin, S., and Wooldridge, M. (2017c). Nash equilibria in concurrent games with lexicographic preferences. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 1067–1073.

[Gutierrez et al., 2018a] Gutierrez, J., Najib, M., Perelli, G., and Wooldridge, M. (2018a). Eve: A tool for temporal equilibrium analysis. In *Automated Technology for Verification and Analysis, ATVA 2018*, pages 551–557.

[Gutierrez et al., 2019b] Gutierrez, J., Najib, M., Perelli, G., and Wooldridge, M. (2019b). Automated temporal equilibrium analysis: Verification and synthesis of multi-player games. Under Review.

[Gutierrez et al., 2019c] Gutierrez, J., Najib, M., Perelli, G., and Wooldridge, M. (2019c). Equilibrium Design for Concurrent Games. In *30th International Conference on Concurrency Theory (CONCUR 2019)*, pages 22:1–22:16.

[Gutierrez et al., 2019d] Gutierrez, J., Najib, M., Perelli, G., and Wooldridge, M. (2019d). On computational tractability for rational verification. In *Proceedings of*

the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, pages 329–335.

[Gutierrez et al., 2018b] Gutierrez, J., Perelli, G., and Wooldridge, M. (2018b). Imperfect information in reactive modules games. *Information and Computation*, 261(Part):650–675.

[Halpern et al., 1983] Halpern, J., Manna, Z., and Moszkowski, B. (1983). A hardware semantics based on temporal intervals. Technical report, Stanford, CA, USA.

[Halpern and Moses, 1990] Halpern, J. Y. and Moses, Y. (1990). Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587.

[Halpern and Shoham, 1991] Halpern, J. Y. and Shoham, Y. (1991). A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962.

[Hansson and Jonsson, 1994] Hansson, H. and Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535.

[Hennessy and Milner, 1985] Hennessy, M. and Milner, R. (1985). Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161.

[Herzig et al., 2016] Herzig, A., Lorini, E., Maffre, F., and Schwarzentruber, F. (2016). Epistemic boolean games based on a logic of visibility and control. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, pages 1116–1122.

[Holzmann, 1997] Holzmann, G. (1997). The Spin model checker. *IEEE Transactions on Software Engineering*, 23(5):279–295.

[Hurwicz and Reiter, 2006] Hurwicz, L. and Reiter, S. (2006). *Designing Economic Mechanisms*. Cambridge University Press.

[Jurdzinski, 1998] Jurdzinski, M. (1998). Deciding the winner in parity games is in UP ∩ co-up. *Information Processing Letters*, 68(3):119–124.

[Kamp, 1968] Kamp, H. (1968). *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA.

[Kirilenko et al., 2017] Kirilenko, A., Kyle, A. S., Samadi, M., and Tuzun, T. (2017). The flash crash: High-frequency trading in an electronic market. *The Journal of Finance*, 72(3):967–998.

[Krentel, 1988] Krentel, M. (1988). The Complexity of Optimization Problems. *Journal of Computer and System Sciences*, 36(3):490 – 509.

[Kripke, 1963] Kripke, S. (1963). Semantical analysis of modal logic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96.

[Kuhn, 2003] Kuhn, H. W. (2003). *Lectures on the Theory of Games (AM-37)*. Princeton University Press.

[Kupferman, 2018] Kupferman, O. (2018). Automata theory and model checking. In *Handbook of Model Checking.*, pages 107–151.

[Kupferman et al., 2016] Kupferman, O., Perelli, G., and Vardi, M. Y. (2016). Synthesis with rational environments. *Annals of Mathematics and Artificial Intelligence*, 78(1):3–20.

[Kwiatkowska et al., 2011] Kwiatkowska, M., Norman, G., and Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*, pages 585–591.

[Ladin et al., 1992] Ladin, R., Liskov, B., Shrira, L., and Ghemawat, S. (1992). Providing high availability using lazy replication. *ACM Transactions on Computer Systems*, 10(4):360–391.

[Lamport, 1980] Lamport, L. (1980). "sometime" is sometimes "not never" - on the temporal logic of programs. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages, POPL 1980*, pages 174–185.

[Laroussinie and Markey, 2015] Laroussinie, F. and Markey, N. (2015). Augmenting ATL with strategy contexts. *Information and Computation*, 245:98–123.

[Lewis, 1918] Lewis, C. I. (1918). *A survey of symbolic logic*. Berkeley University of California Press.

[Lewis and Langford, 1932] Lewis, C. I. and Langford, C. H. (1932). *Symbolic logic*. Century philosophy series. Century Co, New York.

[Löding, 2012] Löding, C. (2012). Basics on tree automata. In *Modern Applications of Automata Theory.*, pages 79–110.

[Lomuscio et al., 2017] Lomuscio, A., Qu, H., and Raimondi, F. (2017). MCMAS: an open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer*, 19(1):9–30.

[Lomuscio and Raimondi, 2006] Lomuscio, A. and Raimondi, F. (2006). MCMAS: a tool for verifying multi-agent systems. In *Proceedings of The Twelfth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2006)*.

[Mailath, 1998] Mailath, G. J. (1998). Do people play nash equilibrium? lessons from evolutionary game theory. *Journal of Economic Literature*, 36(3):1347–1374.

[Manna and Pnueli, 1992] Manna, Z. and Pnueli, A. (1992). *The temporal logic of reactive and concurrent systems - specification*. Springer.

[Maschler et al., 2013] Maschler, M., Solan, E., and Zamir, S. (2013). *Game Theory*. Cambridge University Press. Cambridge Books Online.

[MCK Web, 2019a] MCK Web (2019a). MCK. `http://cgi.cse.unsw.edu.au/~mck/pmck/`. Accessed: 2019-09-18.

[MCK Web, 2019b] MCK Web (2019b). MCK documentation. `http://cgi.cse.unsw.edu.au/~mck/pmck/mcks/documentation`.

[MCMAS Web, 2019] MCMAS Web (2019). VAS Verification of Autonomous Systems. `http://vas.doc.ic.ac.uk/software/mcmas/`. Accessed: 2019-09-18.

[McMillan, 1993a] McMillan, K. L. (1993a). *The SMV System*, pages 61–85. Springer US, Boston, MA.

[McMillan, 1993b] McMillan, K. L. (1993b). *Symbolic Model Checking*. Kluwer Academic Publishers: Dordrecht, The Netherlands.

[McNaughton, 1993] McNaughton, R. (1993). Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149 – 184.

[Milner, 1980] Milner, R. (1980). *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer.

[Mogavero et al., 2012] Mogavero, F., Murano, A., Perelli, G., and Vardi, M. Y. (2012). What makes ATL* decidable? A decidable fragment of strategy logic. In *Proceedings of the 23rd Internatinal Conference on Concurrency Theory, CONCUR 2012*, volume 7454 of *LNCS*, pages 193–208. Springer.

[Mogavero et al., 2014] Mogavero, F., Murano, A., Perelli, G., and Vardi, M. Y. (2014). Reasoning about strategies: On the model-checking problem. *ACM Transactions on Computational Logic*, 15(4):34:1–34:47.

[Mogavero et al., 2010] Mogavero, F., Murano, A., and Vardi, M. Y. (2010). Reasoning about strategies. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010*, pages 133–144.

[Nash, 1951] Nash, J. (1951). Non-cooperative games. *Annals of Mathematics*, 54(2):286–295.

[Nisan et al., 2007] Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V., editors (2007). *Algorithmic Game Theory*. Cambridge University Press: Cambridge, England.

[Ohrstrom and Hasle, 1995] Ohrstrom, P. and Hasle, P. F. (1995). *Temporal Logic from Ancient Ideas to Artificial Intelligence*. Kluwer Academic Publishers, Dordrecht.

[Osborne and Rubinstein, 1994] Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory*. The MIT Press: Cambridge, MA.

[Papadimitriou, 1977] Papadimitriou, C. (1977). The Euclidean Travelling Salesman Problem is NP-complete. *Theoretical Computer Science*, 4(3):237 – 244.

[Papadimitriou, 1984] Papadimitriou, C. (1984). On the Complexity of Unique Solutions. *Journal of the ACM*, 31(2):392–400.

[Papadimitriou, 1994] Papadimitriou, C. (1994). *Computational complexity*. Addison-Wesley, Reading, Massachusetts.

[Papadimitriou and Yannakakis, 1984] Papadimitriou, C. and Yannakakis, M. (1984). The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244 – 259.

[Perelli, 2019] Perelli, G. (2019). Enforcing equilibria in multi-agent systems. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pages 188–196.

[PGSolver, 2019] PGSolver (2019). PGSolver. `https://github.com/tcsprojects/pgsolver`. Accessed: 09-09-2019.

[Piterman, 2007] Piterman, N. (2007). From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3):1–21.

[Piterman and Pnueli, 2006] Piterman, N. and Pnueli, A. (2006). Faster solutions of rabin and streett games. In *Proceedings of the 21th IEEE Symposium on Logic in Computer Science, LICS 2006*, pages 275–284.

[Pnueli, 1977] Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 46–57.

[Pnueli and Rosner, 1989] Pnueli, A. and Rosner, R. (1989). On the synthesis of a reactive module. In *Proceedings of the Sixteenth ACM Symposium on the Principles of Programming Languages (POPL)*, pages 179–190.

[Prezza, 2016] Prezza, N. (2016). CTLSAT: CTL (computation tree logic) SAT solver. `http://github.com/nicolaprezza/CTLSAT`. Accessed: 2016-09-23.

[Prior, 1957] Prior, A. N. (1957). *Time and Modality*. John Locke Lectures. Clarendon Press.

[Prior, 1967] Prior, A. N. (1967). *Past, Present and Future*. Oxford, Clarendon Press.

[Prior, 1968] Prior, A. N. (1968). *Papers on time and tense*. Oxford, Clarendon Press.

[PRISM Web, 2019a] PRISM Web (2019a). PRISM - probabilistic symbolic model checker. `http://www.prismmodelchecker.org/`. Accessed: 2019-09-18.

[PRISM Web, 2019b] PRISM Web (2019b). PRISM-games. `http://www.prismmodelchecker.org/games/`. Accessed: 2019-09-18.

[PRISM Web, 2019c] PRISM Web (2019c). PRISM-games - examples. `http://www.prismmodelchecker.org/games/examples.php`. Accessed: 2019-09-18.

[Queille and Sifakis, 1982] Queille, J.-P. and Sifakis, J. (1982). Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351.

[Rauch Henzinger and Telle, 1996] Rauch Henzinger, M. and Telle, J. (1996). Faster Algorithms for the Nonemptiness of Streett Automata and for Communication Protocol Pruning. In *Proceedings of the 5th Scandinavian Workshop on Algorithm Theory, SWAT*, pages 16–27.

[Reniers and Willemse, 2010] Reniers, M. A. and Willemse, T. A. C. (2010). Folk theorems on the correspondence between state-based and event-based systems. *CoRR*, abs/1011.0136.

[Reynaud, 2016] Reynaud, D. (2016). Mr.waffles. `http://mrwaffles.gforge.inria.fr/`. Accessed: 2016-09-23.

[Russell et al., 2016] Russell, S., Dewey, D., and Tegmark, M. (2016). Research Priorities for Robust and Beneficial Artificial Intelligence. *arXiv:1602.03506 [cs, stat]*. arXiv: 1602.03506.

[S. Heubach and T. Mansour, 2009] S. Heubach and T. Mansour (2009). *Combinatorics of Compositions and Words: Solutions Manual*. Chapman & Hall/CRC.

[Savitch, 1970] Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192.

[Shapley, 1953] Shapley, L. S. (1953). Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095–1100.

[Shoham and Leyton-Brown, 2008] Shoham, Y. and Leyton-Brown, K. (2008). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press: Cambridge, England.

[Sistla and Clarke, 1985] Sistla, A. P. and Clarke, E. M. (1985). The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749.

[Sistla et al., 1987] Sistla, A. P., Vardi, M. Y., and Wolper, P. (1987). The complementation problem for büchi automata with appplications to temporal logic. *Theoretical Computer Science*, 49:217–237.

[Stefansson et al., 2019] Stefansson, E., Fisac, J. F., Sadigh, D., Sastry, S. S., and Johansson, K. H. (2019). Human-robot interaction for truck platooning using hierarchical dynamic games. In *18th European Control Conference, ECC 2019, Naples, Italy, June 25-28, 2019*, pages 3165–3172.

[Tarski, 1955] Tarski, A. (1955). A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309.

[Toumi et al., 2015] Toumi, A., Gutierrez, J., and Wooldridge, M. (2015). *Proceedings of the 12th International Colloquium on Theoretical Aspects of Computing, ICTAC 2015*, chapter A Tool for the Automated Verification of Nash Equilibria in Concurrent Games, pages 583–594.

[Ummels and Wojtczak, 2011] Ummels, M. and Wojtczak, D. (2011). The Complexity of Nash Equilibria in Limit-Average Games. In *Proceedings of the 22nd International Conference on Concurrency Theory, CONCUR*, pages 482–496.

[van Benthem, 2002] van Benthem, J. (2002). Extensive games as process models. *Journal of Logic, Language and Information*, 11(3):289–313.

[van der Hoek et al., 2005] van der Hoek, W., Lomuscio, A., and Wooldridge, M. (2005). On the complexity of practical ATL model checking. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2006)*, Hakodate, Japan.

[van Glabbeek and Weijland, 1996] van Glabbeek, R. J. and Weijland, W. P. (1996). Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600.

[Vardi, 1995] Vardi, M. Y. (1995). Alternating automata and program verification. In *Computer Science Today: Recent Trends and Developments*, pages 471–485.

[Vardi and Wolper, 1986] Vardi, M. Y. and Wolper, P. (1986). An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings of the Symposium on Logic in Computer Science (LICS '86)*, pages 332–344.

[Velner et al., 2015] Velner, Y., Chatterjee, K., Doyen, L., Henzinger, T., Rabinovich, A., and Raskin, J. (2015). The Complexity of Multi-Mean-Payoff and Multi-Energy Games. *Information and Computation*, 241:177–196.

[Venema, 1991] Venema, Y. (1991). A modal logic for chopping intervals. *Journal of Logic and Computation*, 1(4):453–476.

[Vester, 2013] Vester, S. (2013). Alternating-time temporal logic with finite-memory strategies. In *Proceedings Fourth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2013*, pages 194–207.

[Weiß, 1999] Weiß, G., editor (1999). *Multi-Agent Systems*. The MIT Press: Cambridge, MA.

[Wooldridge, 2002] Wooldridge, M. (2002). *An Introduction to Multiagent Systems*. John Wiley & Sons.

[Wooldridge et al., 2013] Wooldridge, M., Endriss, U., Kraus, S., and Lang, J. (2013). Incentive engineering for Boolean games. *Artificial Intelligence*, 195:418 – 439.

[Wooldridge et al., 2016] Wooldridge, M., Gutierrez, J., Harrenstein, P., Marchioni, E., Perelli, G., and Toumi, A. (2016). Rational verification: From model checking to equilibrium checking. In *Proceedings of the 30th Conference on Artificial Intelligence, AAAI*, pages 4184–4191.

[Wuu and Bernstein, 1984] Wuu, G. T. and Bernstein, A. J. (1984). Efficient solutions to the replicated log and dictionary problems. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, pages 233–242.

[Zwick and Paterson, 1996] Zwick, U. and Paterson, M. (1996). The Complexity of Mean Payoff Games on Graphs. *Theoretical Computer Science*, 158(1):343 – 359.