Automated Temporal Equilibrium Analysis: Verification and Synthesis of Multi-Player Games

Julian Gutierrez[a], Muhammad Najib[a], Giuseppe Perelli[b], Michael Wooldridge[a]

[a]*Department of Computer Science, University of Oxford*
[b]*Department of Informatics, University of Leicester*

Abstract

In the context of multi-agent systems, the *rational verification* problem is concerned with checking which temporal logic properties will hold in a system when its constituent agents are assumed to behave rationally and strategically in pursuit of individual objectives. Typically, those objectives are expressed as temporal logic formulae which the relevant agent desires to see satisfied. Unfortunately, rational verification is computationally complex, and requires specialised techniques in order to obtain practically useful implementations. In this paper, we present such a technique. This technique relies on a reduction of the rational verification problem to the solution of a collection of parity games. One key aspect of our approach is that it preserves equilibria across bisimilar systems, which other existing approaches to rational verification do not. Our approach has been implemented in the *E*quilibrium *V*erification *E*nvironment (EVE) system. The EVE system takes as input a model of a concurrent/multi-agent system represented using the Simple Reactive Modules Language (SRML), where agent goals are expressed using Linear Temporal Logic (LTL), together with a claim about the equilibrium behaviour of the system, also expressed as an LTL formula. EVE can then check whether the LTL claim holds on some (or every) computation of the system that could arise through agents choosing Nash equilibrium strategies; it can also check whether a system has a Nash equilibrium, and synthesise

*Corresponding author: Julian Gutierrez.
Email addresses: julian.gutierrez@cs.ox.ac.uk (Julian Gutierrez), mnajib@cs.ox.ac.uk (Muhammad Najib), giuseppe.perelli@leicester.ac.uk (Giuseppe Perelli), michael.wooldridge@cs.ox.ac.uk (Michael Wooldridge)

individual strategies for players in the multi-player game. After presenting our basic framework, we describe our new technique and prove its correctness. We then describe our implementation in the EVE system, and present experimental results which show that EVE performs favourably in comparison to other existing tools that support rational verification.

Keywords: Automated synthesis, Bisimulation invariance, Nash equilibrium, Logic-based multi-player games, Temporal logic and rational verification.

---

1. Introduction

Concurrent and multi-agent systems can be naturally understood and modelled as multi-player games [1, 2]. 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 [3] 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[1].

Rational Verification and Synthesis. In more detail, the two main problems of interest to us are the *rational verification* and automated *synthesis* problems for concurrent/multi-

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

agent systems modelled as multi-player games. In the *rational verification* problem, we desire to check which temporal logic properties are satisfied by the system/game *in equilibrium*, that is, assuming players select strategies that form a Nash equilibrium. A little more formally, let $P_1, \ldots, P_n$ be the agents in our concurrent/multi-agent system, and let $\mathrm{NE}(P_1, \ldots, P_n)$ denote the set of all computation runs of the system that could be generated by agents selecting strategies that form a Nash equilibrium. Finally, let φ be a temporal logic formula. Then, in the rational verification problem, we want to know whether for some/every run $\pi \in \mathrm{NE}(P_1, \ldots, P_n)$ we have $\pi \models \varphi$.

In the automated *synthesis* problem, on the other hand, we additionally desire to *construct* a profile of strategies for players so that the resulting profile is an equilibrium of the game, and induces a computation run that satisfies a given property of interest, again expressed as a temporal logic formula. That is, we are given the system $P_1, \ldots, P_n$, and a temporal logic property φ, and we are asked to compute Nash equilibrium strategies $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$, one for each player in the game, that would result in φ being satisfied in the run $\pi(\vec{\sigma})$ that would be generated when these strategies are enacted.

The Role of Bisimilarity. One crucial aspect of rational verification and synthesis is the role of *bisimilarity* [4, 5, 6, 7]. 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 computer process 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 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* and has been widely used for decades for the semantic analysis (*e.g.*, for modular and compositional reasoning) and formal verification (*e.g.*, for temporal logic model checking) of concurrent and distributed systems. Unfortunately, as shown in [8, 9], 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 [8] and further studied in [9]. However, no algorithmic solutions to these two problems were presented in either [8] or [9]. Instead, the focus in [8, 9] was on classifying classes of games and investigating models of strategies where different kinds of properties (not necessarily temporal logic properties) could be preserved in equilibrium by bisimilarity.

Our Approach. In this paper, we present an approach to the rational verification and automated synthesis problems for concurrent and multi-agent systems, using 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. In particular, we develop a novel technique that can be used for both rational verification and automated synthesis using a reduction to the solution of a collection of *parity games.* The technique avoids any complex automata constructions and as a consequence can be efficiently implemented making use of powerful techniques for parity games and temporal logic synthesis and verification.

The main decision problem that we consider is that of Non-Emptiness, the problem of checking if the set of Nash equilibria in a multi-player game is empty. As we will later show, rational verification and synthesis can be reduced to this problem. If we consider concurrent and multi-player games in which players have goals expressed as temporal logic formulae, this problem is known to be 2EXPTIME-complete for a wide range of system representations and temporal logic languages. For instance, for games with perfect information played on labelled graphs, the problem is 2EXPTIME-complete when goals are given as Linear Temporal Logic (LTL) formulae [10], and 2EXPTIME-hard when goals are given in branching-time temporal logic [11]. The problem is 2EXPTIME-complete even if succinct representations [12, 13] or only two-player games [14] are considered, and becomes undecidable if imperfect information and more than two players are allowed [15], showing the very high complexity of solving this problem, from both practical and theoretical viewpoints.

A common feature of the results above mentioned is that—modulo minor variations—their solutions are, in the end, reduced to the construction of an alternating parity automaton over *infinite trees* (APT [16]) which is then checked for non-emptiness. Here, we present a novel, simpler, and more direct technique for checking the existence of Nash equilibria in games where players have goals expressed in LTL. In particular, our technique does not rely on the solution of an APT. Instead, we reduce the problem to the solution of (a collection of) parity games [17], which are widely used for synthesis and verification.

Formally, 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 $V = V_0 \cup V_1$ 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 $\alpha : V \to \mathbb{N}$ is a labelling priority function. It is known that solving a parity game (checking which player has a winning strategy) is an NP $\cap$ coNP problem [18], polynomial in the number of states and transitions, and exponential in the number of priorities [19]. 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.

Our technique uses parity games in the following way. We take as input a game G (representing a concurrent and multi-agent system) and build a parity game H whose sets of states and transitions are doubly exponential in the size of the input but with priority function only exponential in the size of the input game. Using a deterministic Streett automaton on *infinite words* (DSW [20]), we solve the parity game, leading to a decision procedure that is, overall, in 2EXPTIME, and, therefore, given known hardness results, optimal.

The EVE System. The technique outlined above and described in detail in this paper has been successfully implemented in the *E*quilibrium *V*erification *E*nvironment (EVE) system [21]. EVE takes as input a model of a concurrent/multi-agent system, in which agents are specified using the Simple Reactive Modules Language (SRML) [22, 2], and preferences for agents are defined by associating with each agent a goal, represented as a formula of LTL [23].

Now, given a specification of a multi-agent system and player preferences, the EVE system can: (i) check for the existence of a Nash equilibrium in a multi-player game; (ii) check whether a given LTL formula is satisfied on some or every Nash equilibrium of the system; and (iii) synthesise individual player strategies in the game. As we will show in the paper, EVE performs favourably compared with other existing tools that support rational verification. Moreover, EVE is the first and only tool for automated temporal equilibrium analysis with respect to a model of multi-player games where Nash equilibrium is preserved under bisimilarity, in the way that we describe above.[2]

Note that we believe our choice of the Reactive Modules language is a very natural one [24]: The language is both widely used in practical model checking systems, such as MOCHA [25] and PRISM [26], and close to real-world (declarative) programming models and specification languages.

Structure of the paper. The remainder of this article is structured as follows.

- Section 2 presents the relevant background on games, logic, and automata.
- In Section 3, we formalise the main problem of interest and give a high-level description of the core decision procedure for temporal equilibrium analysis developed in this paper.

[2]Other tools to compute Nash equilibria exist, but they do not use our model of strategies, with which Nash equilibria is preserved by bisimilarity. A comparison with those other techniques for equilibrium analysis are discussed in more detail later.

- In Sections 4, 5, and 6, we describe in detail our main decision procedure for temporal equilibrium analysis, prove its correctness, and show that it is essentially optimal with respect to computational complexity.
- In Section 7, we show how to use our main decision procedure to do formal verification and automated synthesis of logic-based multi-player games.
- In Section 8, we describe the EVE system, and give detailed experimental results which demonstrate that EVE performs favourably in comparison with other tools that support rational verification.
- In Section 9, we conclude, discuss relevant related work, and propose some avenues for future work.

Note that source code for EVE is available online[3], and EVE can also be used as a web service.[4]

2. Preliminaries

Games. A *concurrent multi-player game structure* (CMGS) is a tuple

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

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, and $\mathsf{tr} : \mathrm{St} \times \vec{\mathrm{Ac}} \to \mathrm{St}$ is a (deterministic) *transition function* where $\vec{\mathrm{Ac}} = \mathrm{Ac}_1 \times \cdots \times \mathrm{Ac}_n$ denotes the action profile set. We refer to a profile of actions $\vec{a} = (a_1, \ldots, a_n) \in \vec{\mathrm{Ac}}$ as a *direction*, with directions typically denoted by $d, d', \ldots$ etc. We also consider *partial* directions. 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 $\vec{a}_{\mathrm{N}\setminus\{i\}}$. Finally, for two directions $\vec{a}$ and $\vec{a}'$, we write $(\vec{a}_A, \vec{a}'_{-A})$

[3] `https://github.com/eve-mas/eve-parity`

[4] See `http://eve.cs.ox.ac.uk/`, where both instructions and examples to use the tool and the web service can be found.

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 $\vec{a}'$.

Whenever there is $\vec{a}$ such that $\mathsf{tr}(s, \vec{a}) = s'$, we say that s' is *accessible* from s. A *path* $\pi = s_0, s_1, \ldots \in \mathrm{St}^\omega$ is an infinite sequence of states such that, for every $k \in \mathbb{N}$, s_{k+1} is accessible from s_k. 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 $k \in \mathbb{N}$, $\pi_{k+1} = \mathsf{tr}(\pi_k, \vec{a}_k)$.

A multi-player game structure defines the dynamic structure of a game, but lacks a central aspect of games in the sense of game theory: preferences, which give games their strategic structure. A *multi-player game* is obtained from a structure $\mathcal{M}$ by associating each player with a goal. In this paper, we consider multi-player games with parity and Linear Temporal Logic (LTL) goals.

LTL [23] extends classical propositional logic with two operators, $\mathbf{X}$ ("next") and $\mathbf{U}$ ("until"), that can be used to express properties of paths. The syntax of LTL is defined with respect to a set AP of propositional variables as follows:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \,\mathbf{U}\, \varphi$$

where $p \in \mathrm{AP}$.

We interpret formulae of LTL with respect to pairs (π, t), where π is a path over some multi-player game, $t \in \mathbb{N}$ is a temporal index into π, and $\lambda : \mathrm{St} \to 2^{\mathrm{AP}}$ is a labelling function, that indicates which propositional variables are true in every state. Formally, the semantics of LTL is given by the following rules:

$$\begin{array}{lll}
(\pi, t) \models \top & & \\
(\pi, t) \models p & \text{iff} & p \in \lambda(\pi_t) \\
(\pi, t) \models \neg\varphi & \text{iff} & \text{it is not the case that } (\pi, t) \models \varphi \\
(\pi, t) \models \varphi \vee \psi & \text{iff} & (\pi, t) \models \varphi \text{ or } (\pi, t) \models \psi \\
(\pi, t) \models \mathbf{X}\varphi & \text{iff} & (\pi, t+1) \models \varphi \\
(\pi, t) \models \varphi \,\mathbf{U}\, \psi & \text{iff} & \text{for some } t' \geq t : \big((\pi, t') \models \psi \text{ and} \\
 & & \quad \text{for all } t \leq t'' < t' : (\pi, t'') \models \varphi\big).
\end{array}$$

If $(\pi, 0) \models \varphi$, we write $\pi \models \varphi$ and say that π *satisfies* φ.

Definition 1. A *(concurrent multi-player)* LTL *game* is a tuple

$$\mathcal{G}_{\mathsf{LTL}} = (\mathcal{M}, \lambda, (\gamma_i)_{i \in \mathrm{N}})$$

where $\lambda : \mathrm{St} \to 2^{\mathrm{AP}}$ is a labelling function on the set of states St of $\mathcal{M}$, and each γ_i is the goal of player i, given as an LTL formula over AP.

To define multi-player games with parity goals we consider priority functions. Let $\alpha : \mathrm{St} \to \mathbb{N}$ be a priority function. A path π satisfies $\alpha : \mathrm{St} \to \mathbb{N}$, and write $\pi \models \alpha$ in that 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 2. A *(concurrent multi-player) Parity game* is a tuple

$$\mathcal{G}_{\mathrm{PAR}} = (\mathcal{M}, (\alpha_i)_{i \in \mathrm{N}})$$

where $\alpha_i : \mathrm{St} \to \mathbb{N}$ 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*, $\delta_i : S_i \times \vec{\mathrm{Ac}} \to S_i$ is a deterministic *internal transition function*, and $\tau_i : S_i \to \mathrm{Ac}_i$ an *action function*. Let Σ_i be the set of strategies for player i. A strategy is *memoryless* in $\mathcal{G}$ from s if $S_i = \mathrm{St}$, $s_i^0 = s$, 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

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

Nash equilibrium. Since the outcome of a game determines if a player goal is satisfied, we can define a preference relation $\succeq_i$ over outcomes for each player i. Let w_i be γ_i if $\mathcal{G}$ is an LTL game, and be α_i if $\mathcal{G}$ is a Parity game. Then, for two strategy profiles $\vec{\sigma}$ and $\vec{\sigma}'$ in $\mathcal{G}$, we have

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

On this basis, we can define the concept of Nash equilibrium [3] for a multi-player game with LTL or 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 [9] we showed 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 $\mathcal{G}$.

Automata. 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 run $\pi = q^0, q^1, q^2 \ldots$, let $\mathit{inf}(\pi)$ denote the set of states occurring infinitely often in the run:

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

A run 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:

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

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 run 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 $\mathit{inf}(\pi) \cap E_k = \varnothing$ or $\mathit{inf}(\pi) \cap C_k \neq \varnothing$.

3. A Decision Procedure using Parity Games

We are now in a position to state the problem Non-Emptiness formally:

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 Parity game $\mathcal{G}_{\mathrm{PAR}}$ from an input LTL game $\mathcal{G}_{\mathsf{LTL}}$. 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 $\mathcal{G}_{\mathrm{PAR}}$ which cannot be a part of a Nash equilibrium, thus producing a modified Parity game, $\mathcal{G}_{\mathrm{PAR}}^{-L}$. 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 1 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 $\mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$ and $\mathcal{G}_{\mathrm{PAR}}^{-L}$, two key constructions used in our decision procedure.

Algorithm 1: Nash equilibrium via Parity games

Input: An LTL game $\mathcal{G}_{\mathsf{LTL}} = (\mathrm{N}, (\mathrm{Ac}_i)_{i \in \mathrm{N}}, \mathrm{St}, s_0, \mathsf{tr}, \lambda, (\gamma_i)_{i \in \mathrm{N}})$.

Output: "Yes" if $\mathrm{NE}(\mathcal{G}_{\mathsf{LTL}}) \neq \emptyset$; "No" otherwise.

$\mathcal{G}_{\mathrm{PAR}} \Longleftarrow \mathcal{G}_{\mathsf{LTL}}$; `/* from Section 4 (Theorem 1) */`

foreach $W \subseteq \mathrm{N}$ **do**

    foreach $j \in L = \mathrm{N} \setminus W$ **do**

        Compute $\mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$; `/* from Section 5 (Theorem 2) */`

    end

    Compute $\mathcal{G}_{\mathrm{PAR}}^{-L}$

    foreach $i \in W$ **do**

        Compute $\mathcal{A}_i$ and $\mathcal{S}_i$ from $\mathcal{G}_{\mathrm{PAR}}^{-L}$

    end

    if $\mathcal{L}(\bigtimes_{i \in W}(\mathcal{S}_i)) \neq \emptyset$; `/* from Section 5 (Theorem 3) */`

    then

        return "Yes"

    end

end

return "No"

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 $\mathcal{G}_{\mathrm{PAR}}$ is doubly exponential in the size of the goals of the input LTL game $\mathcal{G}_{\mathsf{LTL}}$, but the priority functions set $(\alpha_i)_{i\in\mathrm{N}}$ is only (singly) exponential. Then, in the second step, reasoning takes only polynomial time in the size of the underlying concurrent game structure of $\mathcal{G}_{\mathrm{PAR}}$, 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 2EXPTIME complexity.

4. From LTL to Parity

We now describe how to realise line 3 of Algorithm 1, 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 $\mathcal{G}_{\mathrm{PAR}}$. This result allows us to shift reasoning on the set of Nash equilibria of $\mathcal{G}_{\mathsf{LTL}}$ into reasoning on the set of Nash equilibria of $\mathcal{G}_{\mathrm{PAR}}$. The basic idea behind this step of the decision procedure is to transform all LTL goals $(\gamma_i)_{i\in\mathrm{N}}$ in $\mathcal{G}_{\mathsf{LTL}}$ into a collection of DPWs, denoted by $(\mathcal{A}_{\gamma_i})_{i\in\mathrm{N}}$, that will be used to build the underlying CMGS of $\mathcal{G}_{\mathrm{PAR}}$. We construct $\mathcal{G}_{\mathrm{PAR}}$ as follows.

In general, using the results in [27, 28], from any LTL formula φ over AP one can build a DPW $\mathcal{A}_{\varphi} = \langle 2^{\mathrm{AP}}, Q, q^0, \rho, \alpha\rangle$ such that, $\mathcal{L}(\mathcal{A}_{\varphi}) = \{\pi \in (2^{\mathrm{AP}})^{\omega} : \pi \models \varphi\}$, 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 $|\mathrm{AP}|$ and the size of the range of α is singly exponential in $|\mathrm{AP}|$. Using this construction we can define, for each LTL goal γ_i, a DPW $\mathcal{A}_{\gamma_i}$.

Definition 3. Let $\mathcal{G}_{\mathsf{LTL}} = (\mathcal{M}, \lambda, (\gamma_i)_{i\in\mathrm{N}})$ be an LTL game whose underlying CMGS is $\mathcal{M} = (\mathrm{N}, (\mathrm{Ac}_i)_{i\in\mathrm{N}}, \mathrm{St}, s_0, \mathsf{tr})$, and let $\mathcal{A}_{\gamma_i} = \langle 2^{\mathrm{AP}}, Q_i, q_i^0, \rho_i, \alpha_i\rangle$ be the DPW corresponding to player i's goal γ_i in $\mathcal{G}_{\mathsf{LTL}}$. The *Parity game* $\mathcal{G}_{\mathrm{PAR}}$ *associated to* $\mathcal{G}_{\mathsf{LTL}}$ is $\mathcal{G}_{\mathrm{PAR}} = (\mathcal{M}', (\alpha'_i)_{i\in\mathrm{N}})$, where $\mathcal{M}' = (\mathrm{N}, (\mathrm{Ac}_i)_{i\in\mathrm{N}}, \mathrm{St}', s'_0, \mathsf{tr}')$ and $(\alpha'_i)_{i\in\mathrm{N}}$ are as follows:

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

Intuitively, the game $\mathcal{G}_{\mathrm{PAR}}$ 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 $\mathcal{G}_{\mathrm{PAR}}$ 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 $\mathcal{G}_{\mathrm{PAR}}$ 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 $\mathcal{G}_{\mathrm{PAR}}$ is the same, since for every state $s \in \mathrm{St}$ and action profile $\vec{a}$, it follows that $\vec{a}$ is available in s if and only if it is available in $(s, q_1, \dots, q_n) \in \mathrm{St}'$, for all $(q_1, \dots, q_n) \in \bigtimes_{i \in \mathrm{N}} Q_i$. Using this correspondence between strategies in $\mathcal{G}_{\mathsf{LTL}}$ and strategies in $\mathcal{G}_{\mathrm{PAR}}$, we can prove the following Lemma, which states an invariance result between $\mathcal{G}_{\mathsf{LTL}}$ and $\mathcal{G}_{\mathrm{PAR}}$ with respect to the satisfaction of players' goals.

Lemma 1 (Goals satisfaction invariance)**.** *Let $\mathcal{G}_{\mathsf{LTL}}$ be an* LTL *game and $\mathcal{G}_{\mathrm{PAR}}$ 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 $\mathcal{G}_{\mathrm{PAR}}$.*

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 $i \in \mathrm{N}$, 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 $\mathcal{G}_{\mathrm{PAR}}$ 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 $\mathcal{G}_{\mathrm{PAR}}$, 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 other direction, observe that all implications used above are equivalences. Using those equivalences one can reason backwards to prove the statement. □

Using Lemma 1 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 1 (Nash equilibrium invariance)**.** *Let* $\mathcal{G}_{\mathsf{LTL}}$ *be an* LTL *game and* $\mathcal{G}_{\text{PAR}}$ *its associated Parity game. Then,* $\text{NE}(\mathcal{G}_{\mathsf{LTL}}) = \text{NE}(\mathcal{G}_{\text{PAR}})$.

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 1, we known 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 1, 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 $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\mathsf{LTL}})$. □

5. Characterising Nash Equilibria

Thanks to Theorem 1, 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 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 [8]. We will show that a similar characterisation holds 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 $s \in \mathrm{St}$ of a Parity game $\mathcal{G}_{\mathrm{PAR}}$, that is, such that $s_i^0 = s$ for every $i \in \mathrm{N}$, we write $\mathcal{G}_{\mathrm{PAR}}, \vec{\sigma}, s \models \alpha_i$ if $\pi(\vec{\sigma}) \models \alpha_i$ in $\mathcal{G}_{\mathrm{PAR}}$. Moreover, if $s = s_0$ is the initial state of the game, we omit it and simply write $\mathcal{G}_{\mathrm{PAR}}, \vec{\sigma} \models \alpha_i$ in such a case.

Definition 4 (Punishment strategies and regions)**.** For a Parity game $\mathcal{G}_{\mathrm{PAR}}$ and a player $i \in \mathrm{N}$, 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 $\mathcal{G}_{\mathrm{PAR}}, (\vec{\sigma}_{-i}, \sigma'_i), s \not\models \alpha_i$. A state s is *punishing* for i if there exists a punishment strategy profile against i in s. By $\mathrm{Pun}_i(\mathcal{G}_{\mathrm{PAR}})$ we denote the set of punishing states, the *punishment region*, for i in $\mathcal{G}_{\mathrm{PAR}}$.

To understand the meaning of a punishment strategy profile, it is useful to think of a modification of the game $\mathcal{G}_{\mathrm{PAR}}$, 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 i does not achieve α_i. This scenario is represented by a two-player zero-sum game in which the winning strategies of the (coalition) player, denoted by $-i$, correspond (one-to-one) to the punishment strategies in the original game $\mathcal{G}_{\mathrm{PAR}}$. As described in [8], 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 $\mathrm{Pun}_i(\mathcal{G}_{\mathrm{PAR}})$, 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 5. For a (concurrent multi-player) Parity game

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

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

- $V_0 = \mathrm{St}$ and $V_1 = \mathrm{St} \times \vec{\mathrm{Ac}}_{-i}$;

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

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

The formal connection between the notion of punishment in $\mathcal{G}_{\mathrm{PAR}}$ and the set of winning strategies in $\mathcal{G}^i_{\mathrm{PAR}}$ is established in the following theorem, where by $\mathrm{Win}_{\mathrm{o}}(\mathcal{G}^j_{\mathrm{PAR}})$ we denote the winning region of Player 0 in $\mathcal{G}^j_{\mathrm{PAR}}$, that is, the states from which Player 0, representing the set of players $-j = \mathrm{N} \setminus \{j\}$ (the coalition of players not including j), has a memoryless winning strategy against player j in the two-player zero-sum parity game $\mathcal{G}^j_{\mathrm{PAR}}$.

Theorem 2. *For all states $s \in \mathrm{St}$, it is the case that $s \in \mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$ if and only if $s \in \mathrm{Win}_{\mathrm{o}}(\mathcal{G}^j_{\mathrm{PAR}})$. In other words, it holds that $\mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}}) = \mathrm{Win}_{\mathrm{o}}(\mathcal{G}^j_{\mathrm{PAR}}) \cap \mathrm{St}$.*

Proof. The proof goes by double inclusion. From left to right, assume $s \in \mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$ and let $\vec{\sigma}_{-j}$ be a punishment strategy profile against player j in s, *i.e.*, such that $\mathcal{G}_{\mathrm{PAR}}, (\vec{\sigma}_{-j}, \sigma'_j) \not\models \alpha_j$, for every strategy $\sigma'_j \in \Sigma_j$ of player j. We now define a strategy σ_0 for player 0 in $\mathcal{G}^j_{\mathrm{PAR}}$ 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 $\mathcal{G}^j_{\mathrm{PAR}}$ 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 $\text{Inf}(\lambda'(\pi')) + 1 = \text{Inf}(\lambda'(\pi'_{\mathsf{odd}})) + 1 \cup \text{Inf}(\lambda'(\pi'_{\mathsf{even}})) + 1 = \text{Inf}(\lambda(\pi))$ and so π' is winning for player 0 in $\mathcal{G}^j_{\text{PAR}}$ and σ_0 is a winning strategy.

From right to left, let $s \in \text{St} \cap \text{Win}_{\text{o}}(\mathcal{G}^j_{\text{PAR}})$ and let σ_0 be a winning strategy for Player 0 in $\mathcal{G}^j_{\text{PAR}}$, and assume σ_0 is memoryless. Now, for every player i, with $i \neq j$, define the memoryless strategy σ_i in $\mathcal{G}_{\text{PAR}}$ such that, for every $s' \in \text{St}$, if $\sigma_0(s') = (s', \vec{a}_{-j})$, then $\sigma_i(s') = (\vec{a}_{-j})_i$ [5], *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 $\mathcal{G}_{\text{PAR}}$, starting from s, that is compatible with $\vec{\sigma}_{-j}$. Thus, there exists a play π' in $\mathcal{G}^i_{\text{PAR}}$, compatible with σ_0, such that $\pi = \pi'_{\mathsf{odd}}$. Moreover, since $\pi'_{\mathsf{odd}} = \pi'_{\mathsf{even}}$, we have that $\text{Inf}(\lambda'(\pi')) = \text{Inf}(\lambda'(\pi'_{\mathsf{odd}})) \cup \text{Inf}(\lambda'(\pi'_{\mathsf{even}})) = \text{Inf}(\lambda(\pi)) - 1$. 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 5 and Theorem 2 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.

Corollary 1. *Computing* $\text{Pun}_i(\mathcal{G}_{\text{PAR}})$ *can be done in polynomial time with respect to the size of the underlying graph of the game* $\mathcal{G}_{\text{PAR}}$ *and exponential in the size of the priority function* α_i*, that is, to the size of the range of* α_i*. Moreover, there is a memoryless*

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

$$
\begin{array}{c}
\qquad\qquad\qquad\qquad\qquad\qquad s' \text{ — } \sigma_i^{\mathrm{pun}j} \longrightarrow \cdots \\
\qquad\qquad\qquad\qquad\qquad \nearrow \\
\qquad\qquad\qquad\qquad ((\vec{a}_k)_{-j}, a'_j) \\
\qquad\qquad\qquad\qquad \diagup \\
s_0 \text{ —— } \vec{a}_0 \longrightarrow s_1 \text{ —— } \vec{a}_1 \longrightarrow \cdots \text{ — } \vec{a}_{k-1} \to s_k \text{ — } \vec{a}_k \longrightarrow s_{k+1} \text{ — } \vec{a}_{k+1} \to \cdots
\end{array}
$$

Figure 2: Representation of the strategy σ_i. At the beginning, player i follows 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 $\sigma_i^{\mathrm{pun}j}$, to employ the punishment strategy against player j.

strategy $\vec{\sigma}_i$ that is a punishment against player i in every state $s \in \mathrm{Pun}_i(\mathcal{G}_{\mathrm{PAR}})$.

As described in [8], 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 6. 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, we have $\mathsf{tr}(\pi_j, ((\vec{a}_k)_{-j}, a'_j)) \in \mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$, where π is the only play in $\mathcal{G}_{\mathrm{PAR}}$ starting from s and generated by η.

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 runs which are *ultimately periodic*, 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. Moreover, since in every run π there are players who get their goals achieved in π (and therefore do not have an incentive to deviate from π) and players who do not get their goals achieve in π (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 $W(\mathcal{G}_{\mathrm{PAR}}, \vec{\sigma}) = \{i \in \mathrm{N} : \mathcal{G}_{\mathrm{PAR}}, \vec{\sigma} \models \alpha_i\}$ denote the set of player that get their goals achieved in $\pi(\vec{\sigma})$, also written as $W(\mathcal{G}_{\mathrm{PAR}}, \pi)$ if $\pi \models \alpha_i$ for some i and path π in $\mathcal{G}_{\mathrm{PAR}}$.

Theorem 3 (Nash equilibrium characterisation)**.** *For a Parity game $\mathcal{G}_{\mathrm{PAR}}$, there is a Nash Equilibrium strategy profile $\vec{\sigma} \in \mathrm{NE}(\mathcal{G}_{\mathrm{PAR}})$ if and only if there is an ultimately periodic action profile run η such that, for every player $j \in L = \mathrm{N} \setminus W(\mathcal{G}_{\mathrm{PAR}}, \pi)$, the run η is punishing-secure for 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, let $\vec{\sigma} \in \mathrm{NE}(\mathcal{G}_{\mathrm{PAR}})$ and η be the ultimately periodic sequence of action profiles generated by $\vec{\sigma}$. Moreover, 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 $W(\mathcal{G}_{\mathrm{PAR}}, \vec{\sigma}) = W(\mathcal{G}_{\mathrm{PAR}}, \eta)$, 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 $\vec{\sigma}_{-j}$. Thus, we have $\pi_h = \pi'_h$ for all $h \leq k$ and $\pi_{k+1} \neq \pi'_{k+1}$. Hence $\pi'_{k+1} = \mathsf{tr}(\pi'_k, (\eta_k)_{-j}, a'_j) = \mathsf{tr}(\pi_k, (\eta_k)_{-j}, a'_j) \in \mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$ and $\mathcal{G}_{\mathrm{PAR}}, (\vec{\sigma}_{-j}, \sigma'_j) \not\models \alpha_j$, 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. $\square$

6. Computing Nash Equilibria

Theorem 3 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 $W \subseteq \mathrm{N}$ of players in a given game $\mathcal{G}_{\mathrm{PAR}}$, which are assumed to get their goals achieved. Now, due to

Theorem 3, 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 $j \in L = \mathrm{N} \setminus W$;
- $\mathcal{G}_{\mathrm{PAR}}, \pi \models \alpha_i$, 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 $\bigcap_{j \in L} \mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$. 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 $\mathsf{tr}(s, (\vec{a}_{-j}, a'_j)) \notin \mathrm{Pun}_j(\mathcal{G}_{\mathrm{PAR}})$, for some $a'_j \in \mathrm{Ac}_j$. Therefore, an action profile run that executes the partial profile $\vec{a}_{-j}$ over s cannot be punishment-secure, and so all edges from s, labelled $\vec{a}_{-j}$, can also be removed. After doing this for every $j \in L$, we obtain $\mathcal{G}^{-L}_{\mathrm{PAR}}$, the game resulting from $\mathcal{G}_{\mathrm{PAR}}$ after the removal of the states and edges just described. As a consequence, $\mathcal{G}^{-L}_{\mathrm{PAR}}$ 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 $\mathcal{G}^{-L}_{\mathrm{PAR}}$ 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 $\mathcal{G}^{-L}_{\mathrm{PAR}}$ 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 [20]). 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 [20]. Finally, as we fixed W at the *beginning*, all we need to do is to use the procedure just described for each $W \subseteq \mathrm{N}$, if needed (see Algorithm 1), obtaining an *optimal* decision procedure that has only exponential time and polynomial space complexity in $|\mathrm{N}|$, the number of agents in the system.[6]

7. 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 3, so we only need to spell it out here. 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 a (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 $i \in \mathrm{N}$ 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 1 and Theorem 1, 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 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 [13, 30, 2], to Non-Emptiness.

Given: Game $\mathcal{G}_{\mathsf{LTL}}$, LTL formula φ.

[6]Some previous techniques, *e.g.* [29], to the computation of pure Nash equilibria are not optimal as they have exponential space complexity in the number of players $|\mathrm{N}|$.

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

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 CMGSs 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 [13] and for multi-player games with LTL goals modelled using SRML [2], which was first presented in [31].

Lemma 2. *Let G be a game and φ be an* LTL *formula. There is a game H of constant size in G, such that $\mathrm{NE}(H) \neq \emptyset$ if and only if $\exists \vec{\sigma} \in \mathrm{NE}(G).\ \pi(\vec{\sigma}) \models \varphi$.*

However, since we have Algorithm 1 at our disposal, an easier—and more direct—solution can be obtained. To solve E-Nash we can modify line 12 of Algorithm 1 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 the Nash equilibrium run that is found as a witness. The new line 12 is "**if** $\mathcal{L}(\bigtimes_{i \in W}(\mathcal{S}_i) \times \mathcal{S}_\varphi) \neq \emptyset$ **then**", 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 1'. We can then use Algorithm 1' to solve A-Nash, also as described in [31]: essentially, we can check whether Algorithm 1'$(\mathcal{G}_{\mathsf{LTL}}, \neg\varphi)$ 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", the case when $(\mathcal{G}_{\mathsf{LTL}}, \varphi)$ is not an instance of A-Nash, then a strategy profile is synthesised from Algorithm 1' which corresponds to a counter-example for $(\mathcal{G}_{\mathsf{LTL}}, \varphi) \in$ A-Nash. It should be easy to see that implementing Algorithm 1' is straightforward from Algorithm 1. Also, as already known, it is also easy to see that Algorithm 1' solves Non-Emptiness if and only if $(\mathcal{G}_{\mathsf{LTL}}, \top) \in$ E-Nash.

8. Implementation

We have implemented the decision procedures presented in this paper. Our implementation uses SRML [22] as a modelling language. SRML is based on the Reactive Modules language [24] which is used in a number of verification tools, including PRISM [26] and MOCHA [25]. The tool that implements our algorithms is called EVE (for *E*quilibrium *V*erification *E*nvironment) [21]. EVE is the *first and only tool* able to analyse the linear temporal logic properties that hold in equilibrium in a concurrent, reactive, and multi-agent system within a bisimulation-invariant framework. It is also the only tool that supports all of the following combined features: a high-level description language using SRML, general-sum multi-player games with LTL goals, bisimulation-invariant strategies, and perfect recall. It is also the only tool for Nash equilibrium analysis that relies on a procedure based on the solution of parity games, which has allowed us to solve the (rational) synthesis problem for individual players in the system using very powerful techniques originally developed to solve the synthesis problem from (linear-time) temporal logic specifications.

To the best of our knowledge, there are only two other tools that can be used to reason about temporal logic equilibrium properties of concurrent/multi-agent systems: PRALINE [32] and MCMAS [33, 34].

PRALINE allows one to compute a Nash equilibrium in a game played in a concurrent game structure [32]. The underlying technique uses alternating Büchi automata and relies on the solution of a two-player zero-sum game called the 'suspect game' [29]. PRALINE can be used to analyse games with different kinds of players goals (*e.g.*, reachability, safety, and others), but does not permit LTL goals, and does not compute bisimulation-invariant strategies.

MCMAS is a model checking tool for multi-agent systems [35]. Since it can be used to model check Strategy Logic (SL [10]) formulae [34], and SL can express the existence of a Nash equilibrium, one can model a multi-agent system in MCMAS and check for the existence of a Nash equilibrium in such a system using SL. However, MCMAS only supports SL with memoryless strategies (while our implementation does not have this restriction) and, as PRALINE, does not compute bisimulation-invariant strategies either.

From the many differences between PRALINE, MCMAS, and EVE (and their associated underlying reasoning and verification techniques), one of the most important ones is bisimulation-invariance, a feature needed to be able to do verification and synthesis, *e.g.*, when using symbolic methods with OBDDs or some model-minimisation techniques. Not being bisimulation-invariant also means that in some cases PRALINE, MCMAS, and EVE would deliver completely different answers. For instance, unlike EVE, with PRALINE and MCMAS it may be the case that for two bisimilar systems PRALINE and MCMAS would compute a Nash equilibrium in one of them and none in the other. A particular instance is the "motivating example" in [9]. Since the two systems there are bisimilar, EVE is able to compute a bisimulation-invariant Nash equilibrium in both systems, while PRALINE and MCMAS cannot. The experiment supporting this claim is reported in Section 8.4 along with the performance results. Indeed, even in cases where all tools are able to compute a Nash equilibrium, EVE outperforms the other two tools as the size of the input system grows, despite the fact that the model of strategies we use in our procedure is *richer* in the sense that it takes into account more information of the underlying game.[7]

8.1. Tool Description

Modelling Language. Systems in EVE are specified with the *Simple Reactive Modules Language* (SRML [22]), that can be used to model non-deterministic systems. Each system component (agent/player) in SRML is represented as a *module*, which consists of an *interface* that defines the name of the module and lists a non-empty set of Boolean variables controlled by the module, and a set of *guarded commands*, which define the choices available to the module at each state. There are two kinds of guarded

[7] As mentioned before, not all games can be tested in all tools since, for instance, PRALINE does not support LTL objectives, but only goals expressed directly as Büchi conditions.

commands: **init**, used for initialising the variables, and **update**, used for updating variables subsequently.

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 the 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 only that it is *enabled* for execution, and thus *may be chosen.* If no guarded command of a module is enabled in some state, then that module has no choice and the values of the variables controlled by it remain unchanged in the next state.

Formally, a guarded command g over a set of 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 controlled variables in g and we denote this set by $ctr(g)$. If no guarded command of a module is enabled, then the values of all variables in $ctr(g)$ are unchanged. A set of guarded commands is said to be *disjoint* if their controlled variables are mutually disjoint. To make it clearer, here is an example of a guarded command:

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

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 true and variable *q* will be assigned false.

module $toggle$ **controls** x

init

$:: \top \rightsquigarrow x' := \top;$

$:: \top \rightsquigarrow x' := \bot;$

update

$:: \neg x \rightsquigarrow x' := \top;$

$:: x \rightsquigarrow x' := \bot;$

Figure 3: Example of module toggle in SRML.

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 a finite set of **init** guarded commands, such that for all $g \in I_i$, we have $ctr(g) \subseteq \Phi_i$, and U_i a finite set of **update** guarded commands, such that for all $g \in U_i$, we have $ctr(g) \subseteq \Phi_i$. Figure 3 shows a module named $toggle$ that controls a Boolean variable named x. There are two **init** guarded commands and two **update** guarded commands. The **init** guarded commands define two choices for the initialisation of variable x: true or false. The first **update** guarded command says that if x has the value of true, then the corresponding choice is to assign it to false, while the second command says that if x has the value of false, then it can be assigned to true. Intuitively, the module would choose (in a non-deterministic manner) an initial value for x, and then on subsequent rounds toggles this value. In this particular example, the **init** commands are non-deterministic, while the **update** commands are deterministic. We refer to [2] for further details on the semantics of SRML. In particular, in Figure 12 of [2], we detail how to build a Kripke structure that models the behaviour of an SRML system. In addition, we associate each module with a goal, which is specified as an LTL formula.

Automated Temporal Equilibrium Analysis. 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 1 to solve Non-Emptiness. The main idea behind this algorithm is illustrated

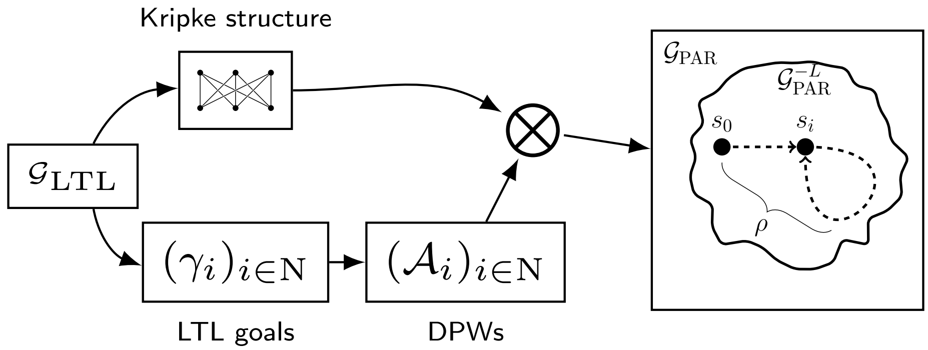

Figure 4: High-level workflow of EVE.

in Figure 4.

EVE was developed in Python and available online from [36]. EVE takes as input a concurrent and multi-agent system described in SRML 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* (*every*) Nash equilibrium of the system, and "NO" otherwise.

8.2. Case Studies

In this section, we present two examples from the literature of concurrent and distributed systems to illustrate 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 [2]), the second one is played on a graph, *e.g.*, as done in [37] with the use of concurrent game structures. Both of these models of games (modelling approaches) can be used within our tool. We will also use these two examples to evaluate EVE's practical performance (and compare it against MCMAS and PRALINE) in Section 8.3.

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

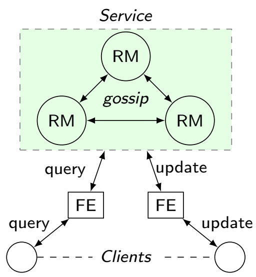

Figure 5: Gossip framework structure.

```
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 6: SRML machine readable code for module RM_1 as written in EVE's input code.

solve problems in many large-scale distributed systems, such as *peer-to-peer* and *cloud* computing systems. Ladin *et al.* [38] developed a framework to provide high availability services via replication which is based on the gossip approach first introduced in [39, 40]. 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 5.

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[8], 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 6, the module RM1 controls a variable: s1. Its value being true signifies that RM1 is in servicing mode; otherwise, 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 $\mathbf{GF}\,\neg\,\texttt{s1}$, which expresses that RM1's goal is to gossip infinitely often: "always" (**G**) "eventually" (**F**) gossip ($\neg\texttt{s1}$).

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

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

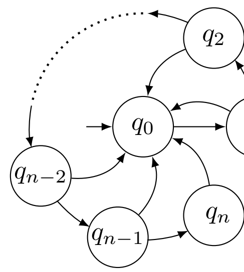

Figure 7: Gifford's protocol modelled as a game.

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.

Replica Control Protocol. Consensus is a key issue in distributed computing and multi-agent systems. An important application domain is in maintaining data consistency. Gifford [41] 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 $\mathrm{N} = \{1, \ldots, n\}$ 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 $\mathrm{N} \setminus \{i\}$ 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}[9]. The voting process then restarts from q_1. The protocol's structure is shown in Figure 7. 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 8.3. 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, each player is allowed to request read/write 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, where the game is played.

8.3. Experiment I

In order to evaluate the practical performance of our tool and approach (against MCMAS and PRALINE), we present results on the temporal equilibrium analysis for the examples in Section 8.2. 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[10] 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 [31]. Time-out ("TO") was fixed to be 7200 seconds.

From the experiment results shown in Table 1 and 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 with more than 100 edges, while EVE could handle up to 6000 edges (for Non-Emptiness).

[9] We assume arithmetic modulo $(|\mathrm{N}| + 1)$ in this example.

[10] 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 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 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

8.4. Experiment II

This experiment is taken from the motivating examples in [9]. 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 edges; then, for each case, we report (i) the total running time[11] ("time") and (ii) whether the tools find any Nash equilibria ("NE").

Table 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 [9]) 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 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 [9]), found a Nash equilibrium in the game. We can also see that EVE outperformed MCMAS on games with 14 or more states. In fact, MCMAS timed-out[12] 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 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 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.

[11] Similarly to Experiment I (Section 8.3), we added to PRALINE's running time the time needed to convert LTL games into its input to carry out a fairer comparison.

[12] We fixed the time-out value to be 3600 seconds (1 hour).

Table 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 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

8.5. 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

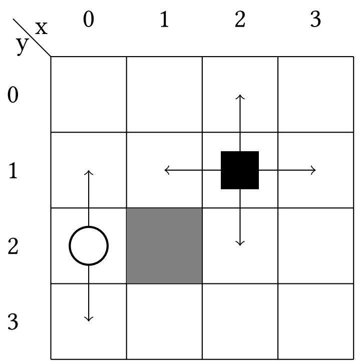

Figure 8: Example of a 4×4 grid world.

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 8; 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 can move at most one step; (2) at each timestep, each agent has to make a move, that is, she cannot stay at the same position for two consecutive timesteps; (3) the goal of each agent is, as stated previously, to eventually reach the opposite corner of her initial position. Furthermore, for E-Nash, 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.

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 num-

Table 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	$19102(14113, 25993)$	$211.30(152.74, 311.43)$	$6871.16(6340.64, 7650.87)$

ber of obstacles ("# Obs"), randomly distributed on the grid. We report the number Kripke states ("KS"), Kripke edges ("KE"), $\mathcal{G}_{\mathrm{PAR}}$ states ("GS"), $\mathcal{G}_{\mathrm{PAR}}$ 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 5, with the following format: *ave*(*min*, *max*).

From the experiment results, we see that EVE works well for Non-Emptiness up

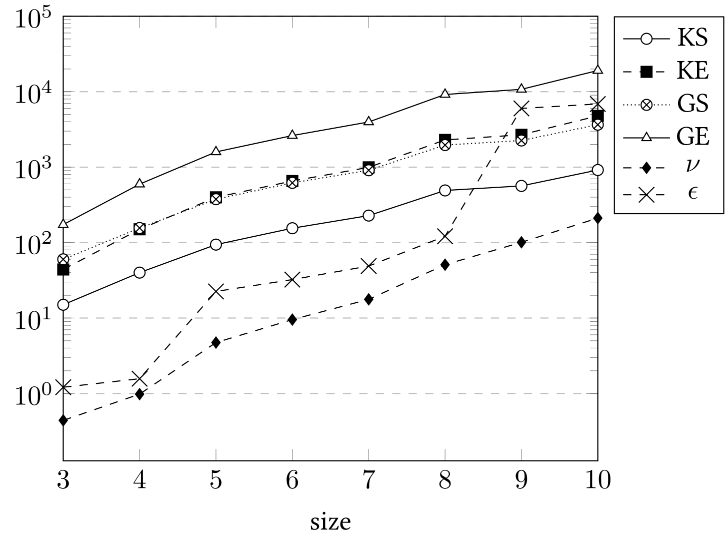

Figure 9: Plots from Table 5. Y-axis is in logarithmic scale.

until size 10. From the plots in Figure 9, we can clearly see that the values of each variable, except for ϵ, grow exponentially. For ϵ (E-Nash), however, it seems to grow 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[13]. 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.

9. Concluding Remarks and Related Work

Equilibrium Analysis in Multi-Agent Systems. The verification problem [42], as conventionally formulated, is concerned with checking that some property, usually defined using a modal or a temporal logic [43], holds on some or on every computation run of a system. In a game-theoretic setting, this can be a very strong requirement – and in some cases even inappropriate – since only some computations of the system will arise (be sustained) as the result of agents in the system choosing strategies in equilibrium, that is, due to strategic and rational play. This has motivated an alterna-

[13]Since the grid coordinate index starts at 0, the "actual" transitions are 3 to 4 and 7 to 8.

tive approach, *rational verification* [2, 30], for the analysis of multi-agent systems. In rational verification, we ask if a given temporal property holds on some or every computation run that can be sustained by agents choosing Nash equilibrium strategies. Rational verification can be reduced to the NON-EMPTINESS problem, as stated in this paper; cf., [31]. As a consequence, along with the polynomial transformations in [31], our results provide a complete framework (theory, algorithms, and implementation) for automated temporal equilibrium analysis, specifically, to do rational synthesis and formal verification of logic-based multi-agent systems. The framework, in particular, provides a concrete and algorithmic solution to the rational synthesis problem as studied in [12], where the Boolean case was given an interesting automata-theoretic solution via (an extension of) Strategy Logic [14].

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 [16]). This approach is used to do rational synthesis [12, 44]; equilibrium checking and rational verification [30, 13, 2]; and model checking of logics for strategic reasoning capable to specify the existence of a Nash equilibrium in concurrent game structures [37], both in two-player games [14, 45] and in multi-player games [46, 10]. 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 [29]. *Our technique 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.

Tools and algorithms. In theory, the kind of equilibrium analysis that can be done using MCMAS [33, 47, 48] and PRALINE [32, 29] rely on the automata-based approach. However, the algorithms that are actually implemented have a different flavour. MCMAS uses a procedure for SL which works as a *labelling algorithm* since it only considers memoryless strategies [48]. On the other hand, PRALINE, which works for Büchi definable objectives, uses a procedure based on the *'suspect game'* [29]. Despite some similarities between our construction and the suspect game, introduced in [29], the two procedures are substantially different. Unlike our procedure, the suspect game is a standard two-player zero-sum turn-based game $\mathcal{H}(\mathcal{G}, \pi)$, constructed from a game $\mathcal{G}$ and a possible path π, in which one of the players ("Eve") has a winning strategy if, and only if, π can be sustained by a Nash equilibrium in $\mathcal{G}$. The overall procedure in [29] relies on the construction of such a game, whose size (space complexity) is exponential in the number of agents [29, Section 4.3]. Instead, our procedure solves, independently, a collection of parity games that avoids an exponential use of space but may require to be executed exponentially many times. Key to the correctness of our approach is that we deal with parity conditions, which are prefix-independent, ensuring that punishment strategies do not depend on the history of the game. Regarding similarities, our procedure also checks for the existence of a path sustained by a Nash Equilibrium, but our algorithm does this for every subset $W \subseteq \mathrm{N}$ of agents, if needed. Doing this (*i.e.*, trading exponential space for exponential time), at every call of this subroutine, our algorithm avoids building an exponentially sized game, like $\mathcal{H}$. On the other hand, from a practical point of view, avoiding the construction of such an exponential sized game leads to better performance (running times), even in cases where no Nash equilibrium exists, when our subroutine is necessarily called exponentially many times. In addition to all of the above, neither the algorithm used for MCMAS nor the one used for PRALINE computes pure Nash equilibria in a bisimulation-invariant framework, as our procedure does. While MCMAS and PRALINE are the two closest tools to EVE, they are not the only available options to reason about games. For instance, PRISM-games [49], EAGLE [50], and UPPAAL [51] are other interesting tools to reason about games. PRISM-games allows one to do strategy synthesis for turn-based stochastic games as well as model checking for long-run, average, and ratio rewards properties. However, PRISM-games does not support any reasoning about equilibria. Contrarily, EAGLE is a tool specifically designed to reason about pure Nash equilibria in multi-player games. EAGLE considers games where goals are given as CTL formulae and allows one to check if a given strategy profile is a Nash equilibrium of a given multi-agent system. This decision problem, called Membership within the rational verification framework [30], is, theoretically, simpler than Non-Emptiness: while the former can be solved in EXPTIME (for branching-time goals expressed using CTL formulae [11]), the latter is 2EXPTIME-complete for LTL goals, and even 2EXPTIME-hard for CTL goals and nondeterministic strategies [11]. UPPAAL is another tool that can be used to analyse equilibrium behaviour in a system [52, 53]. However, UPPAAL differs from EVE is various critical ways: *e.g.*, it works in a quantitative setting, uses statistical model checking, and most importantly, computes approximate Nash equilibria of a game.

Parity games and bisimulation-invariance. Unlike other approaches to rational synthesis and temporal equilibrium analysis, *e.g.* [48, 29, 12, 2], we employ parity games [17], which are an intuitively *simple verification model* with an abundant associated set of algorithmic solutions [54]. In particular, strategies in our framework, as in [2], can depend on players' actions, leading to a much richer game-theoretic setting where *Nash equilibrium is invariant under bisimilarity* [9], a desirable property for concurrent and reactive systems [4, 5, 6, 7]. Bisimulation invariance, in turn, enables the use of standard verification techniques for temporal logics when reasoning about (pure Nash) equilibria. Our reasoning and verification approach applies to multi-player games that are concurrent and synchronous, with perfect recall and perfect information, and which can be represented in a high-level, succinct manner using SRML [22].

Main features of our framework. The technique developed in this paper, and its associated implementation, 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 [9]. 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 [22]). 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 [15, 55].

Future Work. This paper 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 [56]. The paper also solves the problem in practice for perfect information games. 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 [15, 57, 58, 59]. 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 [60, 61].

Acknowledgements. The authors gratefully acknowledge the financial support of the ERC Advanced Investigator Grant 291528 ("RACE") at Oxford. Giuseppe Perelli conducted this research while being member of the University of Oxford, working on the aforementioned Grant. Muhammad Najib was supported by the Indonesia Endowment Fund for Education (LPDP). Part of this paper, focussing on the EVE system, appeared in the Proceedings of ATVA'18 [21].

References

[1] Y. Shoham, K. Leyton-Brown, Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, CUP, 2008.

[2] J. Gutierrez, P. Harrenstein, M. Wooldridge, From model checking to equilibrium checking: Reactive modules for rational verification, Artificial Intelligence 248 (2017) 123–157.

[3] M. J. Osborne, A. Rubinstein, A Course in Game Theory, MIT Press, 1994.

[4] R. Milner, A Calculus of Communicating Systems, Vol. 92 of LNCS, Springer, 1980.

[5] M. Hennessy, R. Milner, Algebraic laws for nondeterminism and concurrency, J. ACM 32 (1) (1985) 137–161.

[6] R. De Nicola, F. W. Vaandrager, Three logics for branching bisimulation, J. ACM 42 (2) (1995) 458–487.

[7] R. J. van Glabbeek, W. P. Weijland, Branching time and abstraction in bisimulation semantics, J. ACM 43 (3) (1996) 555–600.

[8] J. Gutierrez, P. Harrenstein, M. Wooldridge, Expresiveness and complexity results for strategic reasoning, in: CONCUR, Vol. 42 of LIPIcs, Schloss Dagstuhl, 2015, pp. 268–282.

[9] J. Gutierrez, P. Harrenstein, G. Perelli, M. Wooldridge, Nash equilibrium and bisimulation invariance, in: CONCUR, Vol. 85 of LIPIcs, Schloss Dagstuhl, 2017, pp. 17:1–17:16.

[10] F. Mogavero, A. Murano, G. Perelli, M. Y. Vardi, Reasoning about strategies: On the model-checking problem, ACM Transactions on Computational Logic 15 (4) (2014) 34:1–34:47.

[11] J. Gutierrez, P. Harrenstein, M. Wooldridge, Reasoning about equilibria in game-like concurrent systems, Annals of Pure Applied Logic 168 (2) (2017) 373–403.

[12] D. Fisman, O. Kupferman, Y. Lustig, Rational synthesis, in: TACAS, Vol. 6015 of LNCS, Springer, 2010, pp. 190–204.

[13] J. Gutierrez, P. Harrenstein, M. Wooldridge, Iterated boolean games, Information and Computation 242 (2015) 53–79.

[14] K. Chatterjee, T. A. Henzinger, N. Piterman, Strategy logic, Information and Computation 208 (6) (2010) 677–693.

[15] J. Gutierrez, G. Perelli, M. Wooldridge, Imperfect information in reactive modules games, Information and Computation 261 (Part) (2018) 650–675.

[16] C. Löding, Basics on tree automata, in: Modern Applications of Automata Theory, Indian Institute of Science, Bangalore, India, 2012, pp. 79–110.

[17] E. A. Emerson, C. S. Jutla, Tree automata, mu-calculus and determinacy, in: FOCS, IEEE, 1991, pp. 368–377.

[18] M. Jurdzinski, Deciding the winner in parity games is in UP $\cap$ co-up, Information Processing Letters 68 (3) (1998) 119–124.

[19] C. S. Calude, S. Jain, B. Khoussainov, W. Li, F. Stephan, Deciding parity games in quasipolynomial time, in: STOC, ACM, 2017, pp. 252–263.

[20] O. Kupferman, Automata theory and model checking, Handbook of TCS.

[21] J. Gutierrez, M. Najib, G. Perelli, M. Wooldridge, Eve: A tool for temporal equilibrium analysis, in: ATVA, Vol 11138 of LNCS, Springer, Cham, 2018, pp. 551–557.

[22] W. van der Hoek, A. Lomuscio, M. Wooldridge, On the complexity of practical ATL model checking, in: AAMAS, ACM, 2006, pp. 201–208.

[23] A. Pnueli, The temporal logic of programs, in: FOCS, IEEE, 1977, pp. 46–57.

[24] R. Alur, T. A. Henzinger, Reactive modules, Formal Methods in System Design 15 (1) (1999) 7–48.

[25] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, S. Tasiran, MOCHA: modularity in model checking, in: CAV, Vol. 1427 of LNCS, Springer, 1998, pp. 521–525.

[26] M. Z. Kwiatkowska, G. Norman, D. Parker, PRISM: probabilistic model checking for performance and reliability analysis, SIGMETRICS Performance Evaluation Review 36 (4) (2009) 40–45.

[27] A. P. Sistla, M. Y. Vardi, P. Wolper, The complementation problem for büchi automata with appplications to temporal logic, Theoretical Computer Science 49 (1987) 217–237.

[28] N. Piterman, From nondeterministic büchi and streett automata to deterministic parity automata, Logical Methods in Computer Science 3 (3) (2007) 1–21.

[29] P. Bouyer, R. Brenguier, N. Markey, M. Ummels, Pure nash equilibria in concurrent deterministic games, Logical Methods in Computer Science 11 (2) (2015) 1–72.

[30] M. Wooldridge, J. Gutierrez, P. Harrenstein, E. Marchioni, G. Perelli, A. Toumi, Rational verification: From model checking to equilibrium checking, in: AAAI, 2016, pp. 4184–4191.

[31] T. Gao, J. Gutierrez, M. Wooldridge, Iterated boolean games for rational verification, in: AAMAS, ACM, 2017, pp. 705–713.

[32] R. Brenguier, PRALINE: A tool for computing nash equilibria in concurrent games, in: CAV, Vol. 8044 of LNCS, Springer, 2013, pp. 890–895.

[33] P. Cermák, A. Lomuscio, F. Mogavero, A. Murano, MCMAS-SLK: A model checker for the verification of strategy logic specifications, in: CAV, Vol. 8559 of LNCS, Springer, 2014, pp. 525–532.

[34] P. Cermák, A. Lomuscio, F. Mogavero, A. Murano, Practical Verification of Multi-Agent Systems Against SLK Specifications, Information and Computation 261 (Part) (2018) 588–614.

[35] A. Lomuscio, H. Qu, F. Raimondi, MCMAS: an open-source model checker for the verification of multi-agent systems, STTT 19 (1) (2017) 9–30.

[36] EVE: A tool for temporal equilibrium analysis, Available online, `https://github.com/eve-mas/eve-parity` and from `http://eve.cs.ox.ac.uk/eve` (August 2018).

[37] R. Alur, T. A. Henzinger, O. Kupferman, Alternating-time temporal logic, Journal of the ACM 49 (5) (2002) 672–713.

[38] R. Ladin, B. Liskov, L. Shrira, S. Ghemawat, Providing high availability using lazy replication, ACM Trans. Comput. Syst. 10 (4) (1992) 360–391.

[39] M. J. Fischer, A. Michael, 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, ACM, New York, NY, USA, 1982, pp. 70–75.

[40] G. T. Wuu, A. J. Bernstein, Efficient solutions to the replicated log and dictionary problems, in: Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, PODC '84, ACM, New York, NY, USA, 1984, pp. 233–242.

[41] D. K. Gifford, Weighted voting for replicated data, in: Proceedings of the Seventh ACM Symposium on Operating Systems Principles, SOSP '79, ACM, New York, NY, USA, 1979, pp. 150–162.

[42] E. M. Clarke, O. Grumberg, D. A. Peled, Model Checking, MIT Press, Cambridge, MA, USA, 2002.

[43] E. A. Emerson, Temporal and modal logic, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), MIT Press, Cambridge, MA, USA, 1990, pp. 995–1072.

[44] O. Kupferman, G. Perelli, M. Y. Vardi, Synthesis with rational environments, Annals of Mathematics and Artificial Intelligence 78 (1) (2016) 3–20.

[45] B. Finkbeiner, S. Schewe, Coordination logic, in: CSL, Vol. 6247 of LNCS, Springer, 2010, pp. 305–319.

[46] F. Laroussinie, N. Markey, Augmenting ATL with strategy contexts, Information and Computation 245 (2015) 98–123.

[47] P. Cermák, A. Lomuscio, A. Murano, Verifying and synthesising multi-agent systems against one-goal strategy logic specifications, in: AAAI, AAAI Press, 2015, pp. 2038–2044.

[48] P. Cermák, A. Lomuscio, F. Mogavero, A. Murano, Practical verification of multi-agent systems against slk specifications, Information and Computation 261 (Part) (2018) 588–614.

[49] M. Kwiatkowska, D. Parker, C. Wiltsche, Prism-games 2.0: A tool for multi-objective strategy synthesis for stochastic games, in: TACAS, Vol. 9636 of LNCS, Springer, 2016, pp. 560–566.

[50] A. Toumi, J. Gutierrez, M. Wooldridge, A tool for the automated verification of nash equilibria in concurrent games, in: ICTAC, Vol. 9399 of LNCS, Springer, 2015, pp. 583–594.

[51] A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, Uppaal SMC tutorial, STTT 17 (4) (2015) 397–415.

[52] A. David, P. G. Jensen, K. G. Larsen, M. Mikucionis, J. H. Taankvist, Uppaal stratego, in: C. Baier, C. Tinelli (Eds.), TACAS, Vol. 9035 of LNCS, Springer, 2015, pp. 206–211.

[53] P. E. Bulychev, A. David, K. G. Larsen, A. Legay, M. Mikucionis, Computing nash equilibrium in wireless ad hoc networks: A simulation-based approach, in: J. Reich, B. Finkbeiner (Eds.), Proceedings Second International Workshop on Interactions, Games and Protocols, IWIGP, Vol. 78 of EPTCS, 2012, pp. 1–14.

[54] O. Friedmann, M. Lange, Solving parity games in practice, in: ATVA, Vol. 5799 of LNCS, Springer, 2009, pp. 182–196.

[55] M. Ummels, D. Wojtczak, The complexity of nash equilibria in stochastic multiplayer games, Logical Methods in Computer Science 7 (3) (2011) 1–45.

[56] T. Ågotnes, W. van der Hoek, M. Wooldridge, Reasoning about coalitional games, Artificial Intelligence 173 (1) (2009) 45–79.

[57] F. Belardinelli, A. Lomuscio, A. Murano, S. Rubin, Verification of multi-agent systems with imperfect information and public actions, in: AAMAS, ACM, 2017, pp. 1268–1276.

[58] B. Aminof, F. Mogavero, A. Murano, Synthesis of hierarchical systems, Science of Computer Programming 83 (2014) 56–79.

[59] R. Berthon, B. Maubert, A. Murano, Decidability results for atl* with imperfect information and perfect recall, in: AAMAS, ACM, 2017, pp. 1250–1258.

[60] A. Herzig, E. Lorini, F. Maffre, F. Schwarzentruber, Epistemic boolean games based on a logic of visibility and control, in: IJCAI, IJCAI/AAAI Press, 2016, pp. 1116–1122.

[61] F. Belardinelli, A. Lomuscio, Quantified epistemic logics for reasoning about knowledge in multi-agent systems, Artificial Intelligence 173 (9-10) (2009) 982–1013.