

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

Julian Gutierrez<sup>a</sup>, Muhammad Najib<sup>b</sup>, Giuseppe Perelli<sup>c</sup>, Michael Wooldridge<sup>d</sup>

<sup>a</sup>*Faculty of Information Technology, Monash University*

<sup>b</sup>*Department of Computer Science, University of Kaiserslautern*

<sup>c</sup>*Department of Computer, Automatic, and Management Engineering, Sapienza University of Rome*

<sup>d</sup>*Department of Computer Science, University of Oxford*

---

## 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 useable 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. Our approach has been implemented in the *Equilibrium Verification Environment* (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 represented as Linear Temporal Logic (LTL) formulae, 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 individual strategies for players in the multi-player game. After

---

\*Corresponding author: Julian Gutierrez.

Email addresses: julian.gutierrez@monash.edu (Julian Gutierrez), najib@cs.uni-kl.de (Muhammad Najib), perelli@diag.uniroma1.it (Giuseppe Perelli), michael.wooldridge@cs.ox.ac.uk (Michael Wooldridge)

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:* Multi-agent systems, Temporal logic, Nash equilibrium, Bisimulation invariance, Rational verification, Model checking, Synthesis.

---

## 1. Introduction

The deployment of AI technologies in a wide range of application areas over the past decade has brought the problem of *verifying* such systems into sharp focus. Verification is the problem of ensuring that a particular system is correct with respect to some specification. The most successful approach to automated formal verification is that of *model checking* [1]. With this approach, we first derive a finite state abstract model of the system  $\mathcal{S}$  being studied; a common approach involves representing the system as a directed graph in which vertices correspond to states of the system, and edges correspond to the execution of program instructions, or the performance of actions; branching in the graph represents either input from the environment, or choices available to components of the system. With this approach, the directed graph is typically referred to as a labelled transition system, or Kripke structure: each path through the transition system represents a possible execution or computation of the system  $\mathcal{S}$ . Correctness properties of interest are expressed as formulae  $\varphi$  of propositional temporal logic—the most popular such logics for this purpose are Linear Temporal Logic (LTL) and the Computation Tree Logic (CTL). In the case of properties  $\varphi$  expressed as LTL formulae, we typically want to check whether  $\varphi$  is satisfied on some or all possible computations of  $\mathcal{S}$ , that is, on some or all possible paths through the transition system/Kripke structure representing  $\mathcal{S}$ .

Great advances have been made in model checking since the approach was first proposed in the early 1980s, and the technique is now widely used in industry. Nevertheless, the verification of practical software systems is by no means a solved problem, and remains the subject of intense ongoing research. The verification of AI systems,

24 however, raises a distinctive new set of challenges. The present paper is concerned  
25 with the problem of verifying *multi-agent systems*, which are AI systems consisting of  
26 multiple interacting semi-autonomous software components known as *agents* [2, 3].  
27 Software agents were originally proposed in the late 1980s, but it is only over the past  
28 decade that the software agent paradigm has been widely adopted. At the time of  
29 writing, software agents are ubiquitous: we have software agents in our phone (*e.g.*,  
30 Siri), processing requests online, automatically trading in global markets, controlling  
31 complex navigation systems (*e.g.*, those in self-driving cars), and even carrying out  
32 tasks on our behalf in our homes (*e.g.*, Alexa). Typically, these agents do not work in  
33 isolation: they may interact with humans or with other software agents. The field of  
34 multi-agent systems is concerned with understanding and engineering systems that  
35 have these characteristics.

36 We typically assume that agents are acting in pursuit of goals or preferences that  
37 are delegated to them by their users. However, whether an agent is able to achieve  
38 its goal, or the extent to which it can bring about its preferences, will be directly  
39 influenced by the behaviour of other agents. Thus, to act optimally, an agent must  
40 reason *strategically*, taking into account the goals/preferences of other agents, and the  
41 fact that they too will be acting strategically in the pursuit of these, taking into account  
42 the goals/preferences of other agents and their own strategic behaviour. *Game theory*  
43 is the mathematical theory of strategic interaction, and as such, it provides a natural  
44 set of tools for reasoning about multi-agent systems [4].

45 With respect to the problem of verifying multi-agent systems, the relevance of  
46 game theory is as follows. Suppose we are interested in whether a multi-agent system  
47  $\mathcal{S}$ , populated by self-interested agents, might exhibit some property represented by an  
48 LTL formula  $\varphi$ . We can, of course, directly apply standard model checking techniques,  
49 to determine whether  $\varphi$  holds on some or all computations of  $\mathcal{S}$ . However, given that  
50 our agents are assumed to act rationally, whether  $\varphi$  holds on some or all computations  
51 is not relevant if the computations in question involve irrational choices on behalf of  
52 some agents in the system. A much more relevant question, therefore, is whether  $\varphi$   
53 holds on some or all computations *that could result from agents in the system making*  
54 *rational choices*. This raises the question of what counts as a rational choice by the

55 agents in the system, and for this game theory provides a number of answers, in the  
56 form of *solution concepts* such as Nash equilibrium [4, 3]. Thus, from the point of view  
57 of game theory, *correct behaviour* would correspond to *rational behaviour* according  
58 to some game theoretic solution concept, which is another way of saying that agents  
59 in the system will act *optimally* with respect to their preferences/goals, under the  
60 assumption that other agents do the same.

61 This approach to reasoning about the behaviour of multi-agent AI systems es-  
62 tablishes a natural connection between multi-agent systems and multi-player games:  
63 agents correspond to players, computations of the multi-agent system correspond to  
64 plays of the game, individual agent behaviours correspond to player strategies (which  
65 define how players make choices in the system over time), and correct behaviour  
66 would correspond to rational behaviour—in our case, player behaviour that is con-  
67 sistent with the set of Nash equilibria of the multi-player game, whenever such a  
68 set is non-empty. Our main interest in this paper is the development of the theory,  
69 algorithms, and tools for the automated game theoretic analysis of concurrent and  
70 multi-agent systems, and in particular, the analysis of temporal logic properties that  
71 will hold in a multi-agent system under the assumption that players choose strategies  
72 which form a Nash equilibrium<sup>1</sup>.

73 The connection between AI systems (modelled as multi-agent systems) and multi-  
74 player games is well-established, but one may still wonder why *correct behaviour* for  
75 the AI system should correspond to *rational behaviour* in the multi-player game. This  
76 is a legitimate question, especially, because game theory offers very many different  
77 notions of rationality, and therefore of optimal behaviour in the system/game. For  
78 instance, solution concepts such as subgame-perfect Nash equilibrium (SPNE) and  
79 strong Nash equilibrium (SNE) are refinements of Nash equilibrium where the notion  
80 of rationality needs to satisfy stronger requirements. Consequently, there may be  
81 executions of a multi-agent system that would correspond to a Nash equilibrium of  
82 the associated multi-player game (thus, regarded as correct behaviours of the multi-

---

<sup>1</sup>Although in this work we focus on Nash equilibrium, a similar methodology may be applied using refinements of Nash equilibrium and other solution concepts.

agent system), but which do not correspond to a subgame-perfect Nash equilibrium or to a strong Nash equilibrium of the associated multi-player game. We do not argue that Nash equilibrium is the only solution concept of relevance in the game theoretic analysis of multi-agent systems, but we believe (as do many others [3, 5, 6]) that Nash equilibrium is a natural and appropriate starting point for such an analysis. Taking Nash equilibrium as our baseline notion of rationality in multi-player games, and therefore of correctness in multi-agent systems, we focus our study on two problems related to the temporal equilibrium analysis of multi-agent systems [7, 8], as we now explain.

*Synthesis and Rational Verification.* The two main problems of interest to us are the *rational verification* and *automated synthesis* problems for concurrent and multi-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, temporal logic properties satisfied by executions of the multi-agent system generated by strategies that form a Nash equilibrium. A little more formally, let  $P_1, \dots, P_n$  be the agents in our concurrent and multi-agent system, and let  $\text{NE}(P_1, \dots, P_n)$  denote the set of all executions, hereafter called runs, of the system that could be generated by agents selecting strategies that form a Nash equilibrium. Finally, let  $\varphi$  be an LTL formula. Then, in the rational verification problem, we want to know whether for some/every run  $\pi \in \text{NE}(P_1, \dots, P_n)$  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 multi-player game, and induces a run that satisfies a given property of interest, again expressed as a temporal logic formula. That is, we are given the system  $P_1, \dots, P_n$ , and a temporal logic property  $\varphi$ , and we are asked to compute Nash equilibrium strategies  $\vec{\sigma} = (\sigma_1, \dots, \sigma_n)$ , one for each player in the game, that would result in  $\varphi$  being satisfied in the run  $\pi(\vec{\sigma})$  that would be generated when these strategies are enacted.

111 *Our Approach.* In this paper, we present a new approach to the rational verifica-  
 112 tion and automated synthesis problems for concurrent and multi-agent systems. In  
 113 particular, we develop a novel technique that can be used for both rational verifica-  
 114 tion and automated synthesis using a reduction to the solution of a collection of  
 115 *parity games*. The technique can be efficiently implemented making use of power-  
 116 ful techniques for parity games and temporal logic synthesis and verification, and has  
 117 been deployed in the *Equilibrium Verification Environment* (EVE [9]), which supports  
 118 high-level descriptions of systems/games using the *Simple Reactive Modules Language*  
 119 (SRML [10, 7]) and temporal logic specifications given by Linear Temporal Logic for-  
 120 mulae [11].

121 The central decision problem that we consider is that of NON-EMPTYNESS, the prob-  
 122 lem of checking if the set of Nash equilibria in a multi-player game is empty; as we will  
 123 later show, rational verification and synthesis can be reduced to this problem. If we  
 124 consider concurrent and multi-player games in which players have goals expressed  
 125 as temporal logic formulae, this problem is known to be 2EXPTIME-complete for a  
 126 wide range of system representations and temporal logic languages. For instance, for  
 127 games with perfect information played on labelled graphs, the problem is 2EXPTIME-  
 128 complete when goals are given as LTL formulae [12], and 2EXPTIME-hard when goals  
 129 are given in CTL [13]. The problem is 2EXPTIME-complete even if succinct represen-  
 130 tations [14, 15] or only two-player games [16] are considered, and becomes undecid-  
 131 able if imperfect information and more than two players are allowed [17], showing  
 132 the very high complexity of solving this problem, from both practical and theoretical  
 133 viewpoints.

134 A common feature of the results above mentioned is that—modulo minor variations—  
 135 their solutions are, in the end, reduced to the construction of an alternating parity  
 136 automaton over *infinite trees* (APT [18]) which are then checked for non-emptiness.  
 137 Here, we present a novel, simpler, and more direct technique for checking the ex-  
 138 istence of Nash equilibria in games where players have goals expressed in LTL. In  
 139 particular, our technique does not rely on the solution of an APT. Instead, we reduce  
 140 the problem to the solution of (a collection of) parity games [19], which are widely  
 141 used for synthesis and verification problems.

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  $\alpha : V \rightarrow \mathbb{N}$  is a labelling priority function. Player 0 wins if the smallest priority that occurs infinitely often in the infinite play is even. Otherwise, player 1 wins. It is known that solving a parity game (checking which player has a winning strategy) is in  $\text{NP} \cap \text{coNP}$  [20], and can be solved in quasi-polynomial time [21]<sup>2</sup>.

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 [22]), we then solve the parity game, leading to a decision procedure that is, overall, in 2EXPTIME, and, therefore, given the hardness results we mentioned above, essentially optimal.

*Context.* Games have several dimensions: for example, they may be cooperative or non-cooperative; have perfect or imperfect information; have perfect or imperfect recall; be stochastic or not; amongst many other features. Each of these aspects will have a modelling and computational impact on the work to be developed, and so it is important to be precise about the nature of the games we are studying, and therefore the assumptions underpinning our approach.

Our framework considers non-cooperative multi-player general-sum games with perfect information, with Nash equilibrium as the main game-theoretic solution concept. The games are played on finite structures (state-transition structures induced by high-level SRML descriptions), with players having goals (preferences over plays) given by LTL formulae and deterministic strategies represented by finite-state machines with output (Moore machines, sometimes referred to as transducers). Because

---

<sup>2</sup>Despite more than 30 years of research, and promising practical performance for algorithms to solve them, it remains unknown whether parity games can be solved in polynomial time.

169 of the features of our framework – chiefly, the fact that players have LTL goals and  
170 games are played on finite structures – considering deterministic strategies modelled  
171 as finite-state machines does not represent a restriction: in our framework, anything  
172 that a player can achieve with a perfect-recall strategy can also be achieved with a  
173 finite-state machine strategy (see, *e.g.*, [15] for the formal results).

174 Finally, we note that our games have equilibria that are *bisimulation invariant*: that  
175 is, bisimilar structures have the same set of Nash equilibria. This is a highly desirable  
176 property, and to the best of our knowledge, in this respect our work is unique in the  
177 computer science and multi-agent systems literatures.

178 *The EVE System.* The technique outlined above and described in detail in this pa-  
179 per has been successfully implemented in the *Equilibrium Verification Environment*  
180 (EVE) system [23]. EVE takes as input a model of a concurrent and multi-agent  
181 system, in which agents are specified using the Simple Reactive Modules Language  
182 (SRML) [10, 7], and preferences for agents are defined by associating with each agent  
183 a goal, represented as a formula of LTL [11]. Note that we believe our choice of the  
184 Reactive Modules language is a very natural one [24]: The language is both widely  
185 used in practical model checking systems, such as MOCHA [25] and PRISM [26], and  
186 close to real-world (declarative) programming models and specification languages.

187 Now, given a specification of a multi-agent system and player preferences, the  
188 EVE system can: (i) check for the existence of a Nash equilibrium in a multi-player  
189 game; (ii) check whether a given LTL formula is satisfied on some or every Nash  
190 equilibrium of the system; and (iii) synthesise individual player strategies in the game.  
191 As we will show in the paper, EVE performs favourably compared with other existing  
192 tools that support rational verification. Moreover, EVE is the first and only tool for  
193 automated temporal equilibrium analysis for a model of multi-player games where  
194 Nash equilibria are preserved under bisimilarity<sup>3</sup>.

195 Note that our approach may be used to model a wide range of multi-agent systems.

---

<sup>3</sup>Other tools to compute Nash equilibria exist, but they do not use our model of strategies. A comparison with those other techniques for equilibrium analysis are discussed later.



196 For example, as shown in [7], it is easy to capture multi-agent STRIPS systems [27].

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

- 198 • Section 2 presents the relevant background on games, logic, and automata.
- 199 • In Section 3, we formalise the main problem of interest and give a high-level  
200 description of the core decision procedure for temporal equilibrium analysis  
201 developed in this paper.
- 202 • In Sections 4, 5, and 6, we describe in detail our main decision procedure for  
203 temporal equilibrium analysis, prove its correctness, and show that it is essen-  
204 tially optimal with respect to computational complexity.
- 205 • In Section 7, we show how to use our main decision procedure to do rational  
206 verification and automated synthesis of logic-based multi-player games.
- 207 • In Section 8, we describe the EVE system, and give detailed experimental results  
208 which demonstrate that EVE performs favourably in comparison with other  
209 tools that support rational verification.
- 210 • In Section 9, we conclude, discuss relevant related work, and propose some  
211 avenues for future work.

212 The source code for EVE is available online<sup>4</sup>, and the system can also be accessed via  
213 the web<sup>5</sup>.

## 214 2. Preliminaries

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

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

215 where  $\mathbf{N} = \{1, \dots, n\}$  is a set of *players*, each  $\mathbf{Ac}_i$  is a set of *actions*,  $\mathbf{St}$  is a set  
216 of *states*, with a designated *initial* state  $s_0$ . With each player  $i \in \mathbf{N}$  and each state

---

<sup>4</sup>See <https://github.com/eve-mas/eve-parity>

<sup>5</sup>See <http://eve.cs.ox.ac.uk/>

217  $s \in \text{St}$ , we associate a non-empty set  $\text{Ac}_i(s)$  of *available* actions that, intuitively,  $i$   
 218 can perform when in state  $s$ . We refer to a profile of actions  $\vec{a} = (a_1, \dots, a_n) \in \vec{\text{Ac}} =$   
 219  $\text{Ac}_1 \times \dots \times \text{Ac}_n$  as a *direction*. We also consider *partial* directions. A direction  $\vec{a}$  is  
 220 available in state  $s$  if for all  $i$  we have  $a_i \in \text{Ac}_i(s)$ . Write  $\vec{\text{Ac}}(s)$  for the set of available  
 221 directions in state  $s$ . For a given set of players  $A \subseteq \mathbb{N}$  and an action profile  $\vec{a}$ , we let  
 222  $\vec{a}_A$  and  $\vec{a}_{-A}$  be two tuples of actions, respectively, one for each player in  $A$  and one  
 223 for each player in  $\mathbb{N} \setminus A$ . We also write  $\vec{a}_i$  for  $\vec{a}_{\{i\}}$  and  $\vec{a}_{-i}$  for  $\vec{a}_{\mathbb{N} \setminus \{i\}}$ . Furthermore, for  
 224 two directions  $\vec{a}$  and  $\vec{a}'$ , we write  $(\vec{a}_A, \vec{a}'_{-A})$  to denote the direction where the actions  
 225 for players in  $A$  are taken from  $\vec{a}$  and the actions for players in  $\mathbb{N} \setminus A$  are taken from  
 226  $\vec{a}'$ . Finally,  $\text{tr}$  is a *deterministic transition function*, which associate each state  $s$  and  
 227 every available direction  $\vec{a}$  in  $s$  a state  $s' \in \text{St}$ .

228 Whenever there is  $\vec{a}$  such that  $\text{tr}(s, \vec{a}) = s'$ , we say that  $s'$  is *accessible* from  $s$ . A  
 229 *path*  $\pi = s_0, s_1, \dots \in \text{St}^\omega$  is an infinite sequence of states such that, for every  $k \in \mathbb{N}$ ,  
 230  $s_{k+1}$  is accessible from  $s_k$ . By  $\pi_k$  we refer to the  $(k+1)$ -th state in  $\pi$  and by  $\pi_{\leq k}$  to  
 231 the (finite) prefix of  $\pi$  up to the  $(k+1)$ -th element. An *action profile run* is an infinite  
 232 sequence  $\eta = \vec{a}_0, \vec{a}_1, \dots$  of action profiles. Note that, since  $\mathcal{M}$  is deterministic (*i.e.*,  
 233 the transition function  $\text{tr}$  is deterministic), for a given state  $s_0$ , an action profile run  
 234 uniquely determines the path  $\pi$  in which, for every  $k \in \mathbb{N}$ ,  $\pi_{k+1} = \text{tr}(\pi_k, \vec{a}_k)$ .

235 A CGS is a type of concurrent system. As such, behaviourally equivalent CGSs  
 236 should give rise to strategically equivalent games. However, that is not always the  
 237 case. A comprehensive study of this issue can be found in [28, 29] where the strate-  
 238 gic power of games is compared using one of the most important behavioural (also  
 239 called observational) equivalences in concurrency, namely bisimilarity, which is usu-  
 240 ally defined over Kripke structures or labelled transition systems (see, *e.g.*, [30, 31]).  
 241 However, the equivalence can be uniformly defined for general CGSs, where direc-  
 242 tions play the role of, for instance, actions in transition systems. Formally, let  $M =$   
 243  $(\mathbb{N}, (\text{Ac}_i)_{i \in \mathbb{N}}, \text{St}, s_0, \text{tr})$  and  $M' = (\mathbb{N}, (\text{Ac}_i)_{i \in \mathbb{N}}, \text{St}', s'_0, \text{tr}')$  be two CGSs, and  $\lambda :$   
 244  $\text{St} \rightarrow \text{AP}$  and  $\lambda' : \text{St}' \rightarrow \text{AP}$  be two labelling functions over a set of propositional  
 245 variables  $\text{AP}$ . A *bisimulation*, denoted by  $\sim$ , between states  $s^* \in \text{St}$  and  $t^* \in \text{St}'$  is  
 246 a non-empty binary relation  $R \subseteq \text{St} \times \text{St}'$ , such that  $s^* R t^*$  and for all  $s, s' \in \text{St}$ ,  
 247  $t, t' \in \text{St}'$ , and  $\vec{a} \in \vec{\text{Ac}}$ :

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

251 Then, if there is a bisimulation between two states  $s^*$  and  $t^*$ , we say that they are  
 252 *bisimilar* and write  $s^* \sim t^*$  in such a case. We also say that CGSs  $M$  and  $M'$  are  
 253 *bisimilar* (in symbols  $M \sim M'$ ) if  $s_0 \sim s'_0$ . Bisimilar structures satisfy the same set  
 254 of temporal logic properties, a desirable property that will be relevant later.

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

LTL [11] extends classical propositional logic with two operators, **X** (“next”) and  
**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:

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

260 where  $p \in \text{AP}$ . The remaining classical logical connectives are defined in terms of  
 261  $\neg$  and  $\vee$  in the usual way. Two key derived LTL operators are **F** (“eventually”) and  
 262 **G** (“always”), which are defined in terms of **U** as follows:  $\mathbf{F}\varphi = \top \mathbf{U} \varphi$  and  $\mathbf{G}\varphi =$   
 263  $\neg \mathbf{F} \neg \varphi$ .

We interpret formulae of LTL with respect to tuples  $(\pi, t, \lambda)$ , where  $\pi$  is a path  
 over some multi-player game,  $t \in \mathbb{N}$  is a temporal index into  $\pi$ , and  $\lambda : \text{St} \rightarrow 2^{\text{AP}}$   
 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:

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

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

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

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

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

267 To define multi-player games with parity goals we consider priority functions.  
268 Let  $\alpha : \text{St} \rightarrow \mathbb{N}$  be a priority function. A path  $\pi$  satisfies  $\alpha : \text{St} \rightarrow \mathbb{N}$ , and write  
269  $\pi \models \alpha$  in that case, if the minimum number occurring infinitely often in the infinite  
270 sequence  $\alpha(\pi_0), \alpha(\pi_1), \alpha(\pi_2), \dots$  is even.

271 Observe that parity conditions are *prefix-independent*, that is, for every path  $\pi$  and  
272 a finite sequence  $h \in \text{St}^*$ , it holds that  $h \cdot \pi \models \alpha$  if and only if  $\pi \models \alpha$ .

**Definition 2.** A (concurrent multi-player) Parity game is a tuple

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

273 where  $\alpha_i : \text{St} \rightarrow \mathbb{N}$  is the goal of player  $i$ , given as a priority function over  $\text{St}$ .

274 Hereafter, for statements regarding either LTL or Parity games, we will simply  
275 denote the underlying structure as  $\mathcal{G}$ . Games are played by each player  $i$  selecting  
276 a *strategy*  $\sigma_i$  that will define how to make choices over time. Formally, for a given  
277 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

278 output (a transducer), where  $S_i$  is a finite and non-empty set of *internal states*,  $s_i^0$  is  
 279 the *initial state*,  $\delta_i : S_i \times \vec{Ac} \rightarrow S_i$  is a deterministic *internal transition function*,  
 280 and  $\tau_i : S_i \rightarrow Ac_i$  an *action function*. Let  $\Sigma_i$  be the set of strategies for player  $i$ .  
 281 A strategy is *memoryless* in  $\mathcal{G}$  from  $s$  if  $S_i = \text{St}$ ,  $s_i^0 = s$ , and  $\delta_i = \text{tr}$ . Once every  
 282 player  $i$  has selected a strategy  $\sigma_i$ , a *strategy profile*  $\vec{\sigma} = (\sigma_1, \dots, \sigma_n)$  results and the  
 283 game has an *outcome*, a path in  $\mathcal{M}$ , which we will denote by  $\pi(\vec{\sigma})$ . Because strategies  
 284 are deterministic,  $\pi(\vec{\sigma})$  is the unique path induced by  $\vec{\sigma}$ , that is, the infinite sequence  
 285  $s_0, s_1, s_2, \dots$  such that

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

288 Note that the path induced by the strategy profile  $\vec{\sigma}(\sigma_1, \dots, \sigma_n)$  from state  $s_0$   
 289 corresponds to the one generated by the finite transducer  $T_{\vec{\sigma}}$  obtained from the com-  
 290 position of the strategies  $\sigma_i$ 's in  $\vec{\sigma}$ , with input set  $\text{St}$  and output set  $\vec{Ac}$ , where the  
 291 initial input is  $s_0$ . Since such transducer is finite, the generated path  $\pi$  is *ultimately*  
 292 *periodic*, that is, there exists  $p, r \in \mathbb{N}$  such that  $\pi_k = \pi_{k+r}$  for every  $p \leq k$ . This means  
 293 that, after the prefix  $\pi_{\leq p}$ , the path loops indefinitely over the sequence  $\pi_{p+1} \dots \pi_{p+r}$ .

**Nash equilibrium.** Since the outcome of a game determines if a player goal is sat-  
 isfied, we can define a preference relation  $\succeq_i$  over outcomes for each player  $i$ . Let  $w_i$   
 be  $\gamma_i$  if  $\mathcal{G}$  is an LTL game, and be  $\alpha_i$  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 [4] for a multi-player  
 game with LTL or parity goals: given a game  $\mathcal{G}$ , a strategy profile  $\vec{\sigma}$  is a *Nash equilib-*  
*rium* of  $\mathcal{G}$  if, for every player  $i$  and strategy  $\sigma'_i \in \Sigma_i$ , we have

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

294 where  $(\vec{\sigma}_{-i}, \sigma'_i)$  denotes  $(\sigma_1, \dots, \sigma_{i-1}, \sigma'_i, \sigma_{i+1}, \dots, \sigma_n)$ , the strategy profile where  
 295 the strategy of player  $i$  in  $\vec{\sigma}$  is replaced by  $\sigma'_i$ . Let  $\text{NE}(\mathcal{G})$  denote the set of Nash

equilibria of  $\mathcal{G}$ . In [28, 29] 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  $\{\Sigma_i\}_{i \in \mathbb{N}}$   
 is a set of *bisimulation-invariant strategies* and that  $\text{NE}(\mathcal{G})$  is the set of bisimulation-  
 invariant Nash equilibrium profiles of  $\mathcal{G}$ .

**Automata.** A *deterministic automaton on infinite words* is a tuple

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

where  $Q$  is a finite set of states,  $\rho : Q \times \text{AP} \rightarrow Q$  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  $\{F_1, \dots, F_n\}$  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*  $\alpha : Q \rightarrow \mathbb{N}$  that maps states to priorities such that  $\alpha(q) = k$  if and only  
 if  $q \in F_k$ . For a run  $\pi = q^0, q^1, q^2 \dots$ , let  $\text{inf}(\pi)$  denote the set of states occurring  
 infinitely often in the run:

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

A run  $\pi$  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:

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

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

**Example.** In order to illustrate the usage of our framework, consider the following  
 example. Suppose we have two robots/agents inhabiting a grid world (an abstraction  
 of some environment, *e.g.*, a warehouse) with dimensions  $n \times n$ . Initially, the agents

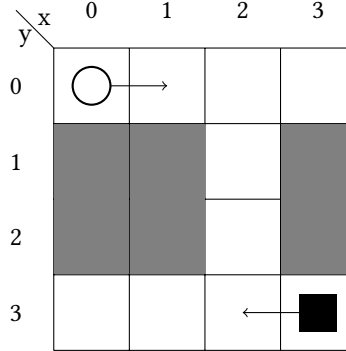


Figure 1: Example of a  $4 \times 4$  grid world.

are located at some corners of the grid; The agents are each able to move around the grid in directions *north*, *south*, *east*, and *west*. The goal of each agent is to reach the opposite corner. For instance, if agent  $i$ 's initial position is  $(0, 0)$ , then the goal is to reach position  $(n - 1, n - 1)$ . A number of obstacles may also appear on the grid. The agents are not allowed to move into a coordinate occupied by an obstacle or outside the grid world. To make it clearer, consider the configuration shown in Figure 1; a (grey) filled square depicts an obstacle. Agent 1, depicted by ■, can only move west to  $(2, 3)$ , whereas agent 2, depicted by ○, can only move east to  $(1, 0)$ .

In this example we make the following assumptions: (1) at each timestep, each agent has to make a move, that is, she cannot stay at the same position for two consecutive timesteps, and she can only move at most one step; (2) the goal of each agent is, as stated previously, to eventually reach the opposite corner of her initial position. From system design point of view, the question that may be asked is: can we synthesise a strategy profile such that it induces a stable (Nash equilibrium) run and at the same time ensures that the agents never crash into each other?

Checking the existence of such strategy profile is not trivial. For instance, the configuration in Figure 1 does not admit any safe Nash equilibrium runs, that is, where all agents get their goals achieved without crashing into each other. On the other hand, the configuration in Figure 2, admits safe Nash equilibrium. Thus, having a tool to verify and synthesise such scenario is desirable, and we will see later in Section 8.5 how to encode and check such systems using our tool.

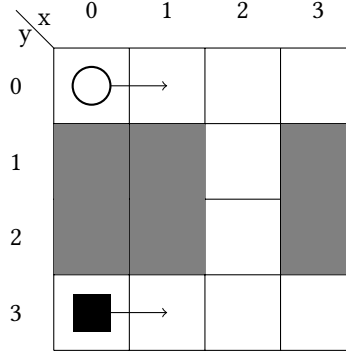


Figure 2: A  $4 \times 4$  grid world with safe Nash equilibrium.

### 330 3. A Decision Procedure using Parity Games

331 We are now in a position to formally state the NON-EMPTYNESS problem:

332 *Given:* An LTL Game  $\mathcal{G}_{\text{LTL}}$ .

333 *Question:* Is it the case that  $\text{NE}(\mathcal{G}_{\text{LTL}}) \neq \emptyset$ ?

334 As indicated before, we solve both verification and synthesis through a reduction  
 335 to the above problem. The technique we develop consists of three steps. First, we  
 336 build a Parity game  $\mathcal{G}_{\text{PAR}}$  from an input LTL game  $\mathcal{G}_{\text{LTL}}$ . Then—using a characteri-  
 337 zation of Nash equilibrium (presented later) that separates players in the game into  
 338 those that achieve their goals in a Nash equilibrium (the “winners”,  $W$ ) and those that  
 339 do not achieve their goals (the “losers”,  $L$ )—for each set of players in the game, we  
 340 eliminate nodes and paths in  $\mathcal{G}_{\text{PAR}}$  which cannot be a part of a Nash equilibrium, thus  
 341 producing a modified Parity game,  $\mathcal{G}_{\text{PAR}}^{-L}$ . Finally, in the third step, we use Streett au-  
 342 tomata on infinite words to check if the obtained Parity game witnesses the existence  
 343 of a Nash equilibrium. The overall algorithm is presented in Algorithm 1 which also  
 344 includes some comments pointing to the relevant Sections/Theorems. The first step  
 345 is contained in line 3, while the third step is in lines 12–14. The rest of the algorithm  
 346 is concerned with the second step. In the sections that follow, we will describe each  
 347 step of the algorithm and, in particular, what are and how to compute  $\text{Pun}_j(\mathcal{G}_{\text{PAR}})$   
 348 and  $\mathcal{G}_{\text{PAR}}^{-L}$ , two key constructions used in our decision procedure.



---

**Algorithm 1:** Nash equilibrium via Parity games

---

```
1 Input: An LTL game  $\mathcal{G}_{\text{LTL}} = (\mathcal{N}, (\text{Ac}_i)_{i \in \mathcal{N}}, \text{St}, s_0, \text{tr}, \lambda, (\gamma_i)_{i \in \mathcal{N}})$ .
2 Output: “Yes” if  $\text{NE}(\mathcal{G}_{\text{LTL}}) \neq \emptyset$ ; “No” otherwise.
3  $\mathcal{G}_{\text{PAR}} \Leftarrow \mathcal{G}_{\text{LTL}}$ ; /* from Section 4 (Theorem 1) */
4 foreach  $W \subseteq \mathcal{N}$  do
5   foreach  $j \in L = \mathcal{N} \setminus W$  do
6     Compute  $\text{P}_{\text{un}_j}(\mathcal{G}_{\text{PAR}})$ ; /* from Section 5 (Theorem 2) */
7   end
8   Compute  $\mathcal{G}_{\text{PAR}}^{-L}$ 
9   foreach  $i \in W$  do
10    Compute  $\mathcal{A}_i$  and  $\mathcal{S}_i$  from  $\mathcal{G}_{\text{PAR}}^{-L}$ 
11  end
12  if  $\mathcal{L}(\bigtimes_{i \in W} (\mathcal{S}_i)) \neq \emptyset$ ; /* from Section 5 (Theorem 3) */
13    then
14      return “Yes”
15    end
16 end
17 return “No”
```

---

349 **Complexity.** The procedure presented above runs in doubly exponential time, match-  
 350 ing the *optimal* upper bound of the problem. In the first step we obtain a doubly ex-  
 351ponential blowup. The underlying structure  $\mathcal{M}$  of the obtained Parity game  $\mathcal{G}_{\text{PAR}}$   
 352 is doubly exponential in the size of the goals of the input LTL game  $\mathcal{G}_{\text{LTL}}$ , but the  
 353 priority functions set  $(\alpha_i)_{i \in \mathbb{N}}$  is only (singly) exponential. Then, in the second step,  
 354 reasoning takes only polynomial time in the size of the underlying concurrent game  
 355 structure of  $\mathcal{G}_{\text{PAR}}$ , but exponential time in both the number of players and the size of  
 356 the priority functions set. Finally, the third step takes only polynomial time, leading  
 357 to an overall 2EXPTIME complexity.

#### 358 4. From LTL to Parity

359 We now describe how to realise line 3 of Algorithm 1, and in doing so we prove a  
 360 strong correspondence between the set of Nash equilibria of the input LTL game  $\mathcal{G}_{\text{LTL}}$   
 361 and the set of Nash equilibria of its associated Parity game  $\mathcal{G}_{\text{PAR}}$ . This result al-  
 362lows us to shift reasoning on the set of Nash equilibria of  $\mathcal{G}_{\text{LTL}}$  into reasoning on  
 363 the set of Nash equilibria of  $\mathcal{G}_{\text{PAR}}$ . The basic idea behind this step of the decision  
 364 procedure is to transform all LTL goals  $(\gamma_i)_{i \in \mathbb{N}}$  in  $\mathcal{G}_{\text{LTL}}$  into a collection of DPWs,  
 365 denoted by  $(\mathcal{A}_{\gamma_i})_{i \in \mathbb{N}}$ , that will be used to build the underlying CGS of  $\mathcal{G}_{\text{PAR}}$ . We  
 366 construct  $\mathcal{G}_{\text{PAR}}$  as follows.

367 In general, using the results in [32, 33], from any LTL formula  $\varphi$  over AP one can  
 368 build a DPW  $\mathcal{A}_\varphi = \langle 2^{\text{AP}}, Q, q^0, \rho, \alpha \rangle$  such that,  $\mathcal{L}(\mathcal{A}_\varphi) = \{\pi \in (2^{\text{AP}})^\omega : \pi \models \varphi\}$ ,  
 369 that is, the language accepted by  $\mathcal{A}_\varphi$  is exactly the set of words over  $2^{\text{AP}}$  that are  
 370 models of  $\varphi$ . The size of  $Q$  is doubly exponential in  $|\text{AP}|$  and the size of the range  
 371 of  $\alpha$  is singly exponential in  $|\text{AP}|$ . Using this construction we can define, for each  
 372 LTL goal  $\gamma_i$ , a DPW  $\mathcal{A}_{\gamma_i}$ .

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

- 378 •  $\text{St}' = \text{St} \times \prod_{i \in \mathbb{N}} Q_i$  and  $s'_0 = (s_0, q_1^0, \dots, q_n^0)$ ;
- 379 • for each state  $(s, q_1, \dots, q_n) \in \text{St}'$  and action profile  $\vec{a}$ ,
- 380  $\text{tr}'((s, q_1, \dots, q_n), \vec{a}) = (\text{tr}(s, \vec{a}), \rho_1(q_1, \lambda(s)), \dots, \rho_n(q_n, \lambda(s)))$ ;
- 381 •  $\alpha'_i(s, q_1, \dots, q_n) = \alpha_i(q_i)$ .

382 Intuitively, the game  $\mathcal{G}_{\text{PAR}}$  is the product of the LTL game  $\mathcal{G}_{\text{LTL}}$  and the collec-  
 383 tion of parity (word) automata  $\mathcal{A}_{\gamma_i}$  that recognise the models of each player's goal.  
 384 Informally, the game executes in parallel the original LTL game together with the au-  
 385 tomata built on top of the LTL goals. At every step of the game, the first component  
 386 of the product state follows the transition function of the original game  $\mathcal{G}_{\text{LTL}}$ , while  
 387 the “automata” components are updated according to the labelling of the current state  
 388 of  $\mathcal{G}_{\text{LTL}}$ . As a result, the execution in  $\mathcal{G}_{\text{PAR}}$  is made, component by component, by the  
 389 original execution, say  $\pi$ , in the LTL game  $\mathcal{G}_{\text{LTL}}$ , paired with the unique runs of the  
 390 DPWs  $\mathcal{A}_{\gamma_i}$  generated when reading the word  $\lambda(\pi)$ .

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

398 **Lemma 1** (Goals satisfaction invariance). *Let  $\mathcal{G}_{\text{LTL}}$  be an LTL game and  $\mathcal{G}_{\text{PAR}}$  its*  
 399 *associated Parity game. Then, for every strategy profile  $\vec{\sigma}$  and player  $i$ , it is the case that*  
 400  *$\pi(\vec{\sigma}) \models \gamma_i$  in  $\mathcal{G}_{\text{LTL}}$  if and only if  $\pi(\vec{\sigma}) \models \alpha_i$  in  $\mathcal{G}_{\text{PAR}}$ .*

401 *Proof.* We prove the statement by double implication. To show the left to right im-  
 402 plication, assume that  $\pi(\vec{\sigma}) \models \gamma_i$  in  $\mathcal{G}_{\text{LTL}}$ , for any player  $i \in \mathbb{N}$ , and let  $\pi$  denote the  
 403 infinite path generated by  $\vec{\sigma}$  in  $\mathcal{G}_{\text{LTL}}$ ; thus, we have that  $\lambda(\pi) \models \gamma_i$ . On the other  
 404 hand, let  $\pi'$  denote the infinite path generated in  $\mathcal{G}_{\text{PAR}}$  by the same strategy profile  $\vec{\sigma}$ .  
 405 Observe that the first component of  $\pi'$  is exactly  $\pi$ . Moreover, consider the  $(i + 1)$ -th  
 406 component  $\rho_i$  of  $\pi'$ . By the definition of  $\mathcal{G}_{\text{PAR}}$ , it holds that  $\rho_i$  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  $\pi'$  according to  $\alpha'_i$  corresponds to the one recognised by  $\mathcal{A}_{\gamma_i}$  in  $\rho_i$ . Thus, since we know that  $\lambda(\pi) \models \gamma_i$ , it follows that  $\rho_i$  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.  $\square$

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  $\mathcal{G}_{\text{LTL}}$  be an LTL game and  $\mathcal{G}_{\text{PAR}}$  its associated Parity game. Then,  $\text{NE}(\mathcal{G}_{\text{LTL}}) = \text{NE}(\mathcal{G}_{\text{PAR}})$ .*

*Proof.* The proof proceeds by double inclusion. First, assume that a strategy profile  $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\text{LTL}})$  is a Nash Equilibrium in  $\mathcal{G}_{\text{LTL}}$  and, by contradiction, it is not a Nash Equilibrium in  $\mathcal{G}_{\text{PAR}}$ . Observe that, due to Lemma 1, we know that the set of players that get their goals satisfied by  $\pi(\vec{\sigma})$  in  $\mathcal{G}_{\text{LTL}}$  (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  $j \in L = N \setminus W$  and a strategy  $\sigma'_j$  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  $\mathcal{G}_{\text{LTL}}$  and so  $\sigma'_j$  would be a beneficial deviation for player  $j$  in  $\mathcal{G}_{\text{LTL}}$  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  $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\text{LTL}})$ .  $\square$

## 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 [34]. 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  $\vec{\sigma} = (\sigma_1, \dots, \sigma_n)$  defined on a state  $s \in \text{St}$  of a Parity game  $\mathcal{G}_{\text{PAR}}$ , that is, such that  $s_i^0 = s$  for every  $i \in \mathbb{N}$ , we write  $\mathcal{G}_{\text{PAR}}, \vec{\sigma}, s \models \alpha_i$  if  $\pi(\vec{\sigma}) \models \alpha_i$  in  $\mathcal{G}_{\text{PAR}}$ . Moreover, if  $s = s_0$  is the initial state of the game, we omit it and simply write  $\mathcal{G}_{\text{PAR}}, \vec{\sigma} \models \alpha_i$  in such a case.

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

To understand the meaning of a punishment (partial) strategy profile, it is useful to think of a modification of the game  $\mathcal{G}_{\text{PAR}}$ , in which player  $i$  still has its goal  $\alpha_i$ , while the rest of the players are collectively playing in an adversarial mode, *i.e.*, trying to make sure that  $i$  does not achieve  $\alpha_i$ . 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  $\mathcal{G}_{\text{PAR}}$ . As described in [34], knowing the set of punishment (partial) strategy profiles in a given game is important to compute its set of Nash Equilibria. For this reason, it is useful to compute the set  $\text{Pun}_i(\mathcal{G}_{\text{PAR}})$ , that is, the set of states in the game from which a given player  $i$  can be punished. (*e.g.*, to deter undesirable unilateral player deviations). To do this, we reduce the problem to computing a winning strategy in a turn-based two-player zero-sum parity game, whose definition is as follows.

**Definition 5.** For a (concurrent multi-player) Parity game

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

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

- $V_0 = \text{St}$  and  $V_1 = \text{St} \times \vec{\text{Ac}}_{-j}$ ;



Figure 3: Sequentialisation of a game. On the left, a representation of a transition from  $s_1$  to  $s_2$  using action profile  $(\vec{a}_{-j}, a_j)$ . 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  $\vec{a}_{-j}$  by coalition  $-j$  in the original game.

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

The formal connection between the notion of punishment in  $\mathcal{G}_{\text{PAR}}$  and the set of winning strategies in  $\mathcal{G}_{\text{PAR}}^j$  is established in the following theorem, where by  $\text{Win}_0(\mathcal{G}_{\text{PAR}}^j)$  we denote the winning region of Player 0 in  $\mathcal{G}_{\text{PAR}}^j$ , that is, the states from which Player 0, representing the set of players  $-j = \mathbb{N} \setminus \{j\}$  (the coalition of players not including  $j$ ), has a memoryless winning strategy against player  $j$  in the two-player zero-sum parity game  $\mathcal{G}_{\text{PAR}}^j$ .

**Theorem 2.** *For all states  $s \in \text{St}$ , it is the case that  $s \in \text{Pun}_j(\mathcal{G}_{\text{PAR}})$  if and only if  $s \in \text{Win}_0(\mathcal{G}_{\text{PAR}}^j)$ . In other words, it holds that  $\text{Pun}_j(\mathcal{G}_{\text{PAR}}) = \text{Win}_0(\mathcal{G}_{\text{PAR}}^j) \cap \text{St}$ .*

*Proof.* The proof goes by double inclusion. From left to right, assume  $s \in \text{Pun}_j(\mathcal{G}_{\text{PAR}})$  and let  $\vec{\sigma}_{-j}$  be a punishment strategy profile against player  $j$  in  $s$ , i.e., such that  $\mathcal{G}_{\text{PAR}}, (\vec{\sigma}_{-j}, \sigma'_j), s \not\models \alpha_j$ , for every strategy  $\sigma'_j \in \Sigma_j$  of player  $j$ . We now define a strategy  $\sigma_0$  for player 0 in  $\mathcal{G}_{\text{PAR}}^j$  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  $\mathcal{G}_{\text{PAR}}^j$  starting from  $s$ , there is a unique finite sequence of action profiles  $\vec{a}_{-j}^0, \dots, \vec{a}_{-j}^k$  and a sequence  $\pi_{\leq k} = s^0, \dots, s^{k+1}$  of states in  $\text{St}^*$  such that

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

Now, for every path  $\pi'_{\leq k}$  of this form that is consistent with  $\vec{\sigma}_{-j}$ , i.e., the sequence  $\vec{a}_{-j}^0, \dots, \vec{a}_{-j}^{k-1}$  is generated by  $\vec{\sigma}_{-j}$ , define  $\sigma_0(\pi'_{\leq k}) = (s^{k+1}, \vec{a}_{-j}^{k+1})$ , where  $\vec{a}_{-j}^{k+1}$  is

the action profile selected by  $\vec{\sigma}_{-j}$ . To prove that  $\sigma_0$  is winning, consider a strategy  $\sigma_1$  for Player 1 and the infinite path  $\pi' = \pi((\sigma_0, \sigma_1))$  generated by  $(\sigma_0, \sigma_1)$ . It is not hard to see that the sequence  $\pi'_{\text{odd}}$  of odd positions in  $\pi'$  belongs to a path  $\pi$  in  $\mathcal{G}_{\text{PAR}}$  and it is consistent with  $\vec{\sigma}_{-j}$ . Thus, since  $\vec{\sigma}_{-j}$  is a punishment strategy,  $\pi'_{\text{odd}}$  does not satisfy  $\alpha_j$ . Moreover, observe that the parity of the sequence  $\pi'_{\text{even}}$  of even positions equals that of  $\pi'_{\text{odd}}$ . Thus, we have that  $\text{Inf}(\lambda'(\pi')) + 1 = \text{Inf}(\lambda'(\pi'_{\text{odd}})) + 1 \cup \text{Inf}(\lambda'(\pi'_{\text{even}})) + 1 = \text{Inf}(\lambda(\pi))$  and so  $\pi'$  is winning for player 0 in  $\mathcal{G}_{\text{PAR}}^j$  and  $\sigma_0$  is a winning strategy.

From right to left, let  $s \in \text{St} \cap \text{Win}_0(\mathcal{G}_{\text{PAR}}^j)$  and let  $\sigma_0$  be a winning strategy for Player 0 in  $\mathcal{G}_{\text{PAR}}^j$ , and assume  $\sigma_0$  is memoryless. Now, for every player  $i$ , with  $i \neq j$ , define the memoryless strategy  $\sigma_i$  in  $\mathcal{G}_{\text{PAR}}$  such that, for every  $s' \in \text{St}$ , if  $\sigma_0(s') = (s', \vec{a}_{-j})$ , then  $\sigma_i(s') = (\vec{a}_{-j})_i$ <sup>6</sup>, i.e., the action that player  $i$  takes in  $\sigma_0$  at  $s'$ . Now, consider the (memoryless) strategy profile  $\vec{\sigma}_{-j}$  given by the composition of all strategies  $\sigma_i$ , and consider a play  $\pi$  in  $\mathcal{G}_{\text{PAR}}$ , starting from  $s$ , that is consistent with  $\vec{\sigma}_{-j}$ . Thus, there exists a play  $\pi'$  in  $\mathcal{G}_{\text{PAR}}^j$ , consistent with  $\sigma_0$ , such that  $\pi = \pi'_{\text{odd}}$ . Moreover, since  $\pi'_{\text{odd}} = \pi'_{\text{even}}$ , we have that  $\text{Inf}(\lambda'(\pi')) = \text{Inf}(\lambda'(\pi'_{\text{odd}})) \cup \text{Inf}(\lambda'(\pi'_{\text{even}})) = \text{Inf}(\lambda(\pi)) - 1$ . Since  $\pi'$  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  $s$ .  $\square$

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. Thus, from Definition 5 and Theorem 2, we have the following corollary.

**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*

---

<sup>6</sup>By an abuse of notation, we let  $\sigma_i(s')$  be the value of  $\tau_i(s')$ .

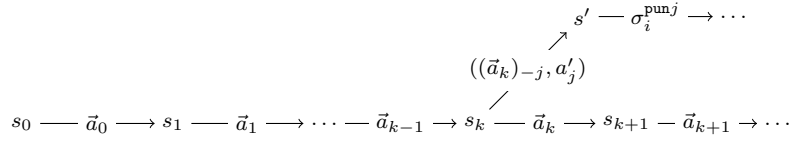


Figure 4: Representation of the strategy  $\sigma_i$ . At the beginning, player  $i$  follows the transducer  $T_\eta$  that generates the action profile run  $\eta$ . The strategy adheres to it until a unilateral deviation from player  $j$  occurs, here represented at the  $k$ -th step of the play. Once the deviation has occurred, and the game entered a state  $s'$ , player  $i$  starts executing the strategy  $\sigma_i^{\text{pun}j}$ , to employ the punishment strategy against player  $j$ .

504 priority function  $\alpha_i$ , that is, to the size of the range of  $\alpha_i$ . Moreover, there is a memoryless  
 505 strategy  $\vec{\sigma}_i$  that is a punishment against player  $i$  in every state  $s \in \text{Pun}_i(\mathcal{G}_{\text{PAR}})$ .

506 As described in [34], in any (infinite) run *sustained* by a Nash equilibrium  $\vec{\sigma}$  in  
 507 deterministic and pure strategies, that is, in  $\pi(\vec{\sigma})$ , it is the case that all players that  
 508 do not get their goals achieved in  $\pi(\vec{\sigma})$  can deviate from such a (Nash equilibrium)  
 509 run only to states where they can be punished by the coalition consisting of all other  
 510 players in the game. To formalise this idea in the present setting, we need one more  
 511 concept about punishments, defined next.

512 **Definition 6.** An action profile run  $\eta = \vec{a}_0, \vec{a}_1, \dots \in \vec{A}c^\omega$  is *punishing-secure* in  $s$  for  
 513 player  $j$  if, for all  $k \in \mathbb{N}$  and  $a'_j$ , we have  $\text{tr}(\pi_j, ((\vec{a}_k)_{-j}, a'_j)) \in \text{Pun}_j(\mathcal{G}_{\text{PAR}})$ , where  
 514  $\pi$  is the only play in  $\mathcal{G}_{\text{PAR}}$  starting from  $s$  and generated by  $\eta$ .

515 Using the above definition, we can characterise the set of Nash equilibria of a  
 516 given game. Recall that strategies are formalised as transducers, *i.e.*, as finite state  
 517 machines with output, so such Nash equilibria strategy profiles produce runs which  
 518 are *ultimately periodic*. Moreover, since in every run  $\pi$  there are players who get their  
 519 goals achieved in  $\pi$  (and therefore do not have an incentive to deviate from  $\pi$ ) and  
 520 players who do not get their goals achieve in  $\pi$  (and therefore may have an incentive  
 521 to deviate from  $\pi$ ), we will also want to explicitly refer to such players. To do that, the  
 522 following notation will be useful: Let  $W(\mathcal{G}_{\text{PAR}}, \vec{\sigma}) = \{i \in \mathbb{N} : \mathcal{G}_{\text{PAR}}, \vec{\sigma} \models \alpha_i\}$  denote  
 523 the set of player that get their goals achieved in  $\pi(\vec{\sigma})$ . We also write  $W(\mathcal{G}_{\text{PAR}}, \pi) =$   
 524  $\{i \in \mathbb{N} : \mathcal{G}_{\text{PAR}}, \pi \models \alpha_i\}$ .



**Theorem 3** (Nash equilibrium characterisation). *For a Parity game  $\mathcal{G}_{\text{PAR}}$ , there is a Nash Equilibrium strategy profile  $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\text{PAR}})$  if and only if there is an ultimately periodic action profile run  $\eta$  such that, for every player  $j \in L = N \setminus W(\mathcal{G}_{\text{PAR}}, \pi)$ , the run  $\eta$  is punishing-secure for  $j$  in state  $s_0$ , where  $\pi$  is the unique path generated by  $\eta$  from  $s_0$ .*

*Proof.* The proof is by double implication. From left to right, for  $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\text{PAR}})$ , let  $\eta$  be the ultimately periodic sequence of action profiles generated by  $\vec{\sigma}$ . Moreover, assume for a contradiction that  $\eta$  is not punishing-secure for some  $j \in L$ . By the definition of punishment-secure, there is  $k \in \mathbb{N}$  and action  $a'_j \in \text{Ac}_j$  for player  $j$  such that  $s' = \text{tr}(\pi_k, ((\vec{a}_k)_{-j}, a'_j)) \notin \text{Pun}_j(\mathcal{G}_{\text{PAR}})$ . Now, consider the strategy  $\sigma'_j$  that follows  $\eta$  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  $\alpha_j$  from that point onwards. Note that such a strategy is guaranteed to exist since  $s' \notin \text{Pun}_j(\mathcal{G}_{\text{PAR}})$ . Therefore,  $\mathcal{G}_{\text{PAR}}, (\vec{\sigma}_{-j}, \sigma'_j) \models \alpha_j$  and so  $\sigma'_j$  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  $\eta$ . First, recall that  $\eta$  can be generated by a finite transducer  $T_\eta = (Q_\eta, q_\eta^0, \delta_\eta, \tau_\eta)$  where  $\delta_\eta : Q_\eta \rightarrow Q_\eta$  and  $\tau_\eta : Q_\eta \rightarrow \vec{\text{Ac}}$ . Moreover, for every player  $i$  and deviating player  $j$ , with  $i \neq j$ , there is a (memoryless) strategy  $\sigma_i^{\text{pun}_j}$  to punish player  $j$  in every state in  $\text{Pun}_j(\mathcal{G}_{\text{PAR}})$ . By suitably combining the transducer with the punishment strategies, we define the following strategy  $\sigma_i = (Q_i, q_i^0, \delta_i, \tau_i)$  for player  $i$  where

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

$$\begin{aligned}
& - \tau_i(s, q, \top) = (\tau_\eta(q))_i, \text{ and} \\
& - \tau_i(s, q, j) = \sigma_i^{\text{pun}j}(s).
\end{aligned}$$

To understand how strategy  $\sigma_i$  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  $\mathsf{T}_\eta$ , which is used to employ the execution of the action profile run  $\eta$ . 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  $\eta$ . At the beginning of the play, strategy  $\sigma_i$  starts executing the actions prescribed by the transducer  $\mathsf{T}_\eta$ . It sticks to it until some losing player  $j$  performs a deviation. In such a case, the third component of the internal state of  $\sigma_i$  switches to remember the deviating player. Moreover, from that point on, it starts executing the punishment strategy  $\sigma_i^{\text{pun}j}$ . Recall that parity conditions are prefix-independent. Therefore, no matter the result of the execution, if all the players start playing according to the punishment strategy  $\sigma_i^{\text{pun}j}$ , the resulting path will not satisfy the parity condition  $\alpha_j$ . Moreover, note that the punishment strategy is *memoryless*. Thus, the application of the punishment strategy takes effect no matter when it is triggered along the execution. Now, define  $\sigma$  to be the collection of all  $\sigma_i$ . It remains to prove that  $\vec{\sigma}$  is a Nash Equilibrium.

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

## 580 6. Computing Nash Equilibria

581 Theorem 3 allows us to reduce the problem of finding a Nash equilibrium to finding  
 582 a path in the game satisfying certain properties, which we will show how to check  
 583 using DPW and DSW automata. To do this, let us fix a given set  $W \subseteq N$  of players  
 584 in a given game  $\mathcal{G}_{\text{PAR}}$ , which are assumed to get their goals achieved. Now, due to  
 585 Theorem 3, we have that an action profile run  $\eta$  corresponds to a Nash equilibrium  
 586 with  $W$  being the set of “winners” in the game if, and only if, the following two  
 587 properties are satisfied:

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

590 where  $\pi$  is, as usual, the path generated by  $\eta$  from  $s^0$ .

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

604 The only thing that remains to be done is to check whether there exists a path in  
 605  $\mathcal{G}_{\text{PAR}}^{-L}$  that satisfies all players in  $W$ . To do this, we use DPW and DSW automata. Since  
 606 players goals are parity conditions, a path satisfying player  $i$  is an accepting run of  
 607 the DPW  $\mathcal{A}^i$  where the set of states and transitions are exactly those of  $\mathcal{G}_{\text{PAR}}^{-L}$  and the  
 608 acceptance condition is given by  $\alpha_i$ . Then, in order to find a path satisfying the goals

609 of all players in  $W$ , we can solve the emptiness problem of the automaton intersection  
 610  $\times_{i \in W} \mathcal{A}^i$ . In general the product intersection of DPW provides a further exponential  
 611 blow-up in the size. However, observe that each  $\mathcal{A}_i$  differs from each other only in  
 612 its acceptance condition  $\alpha_i$ . Moreover, each parity condition  $\alpha = (F_1, \dots, F_n)$  can  
 613 be seen as a Street condition of the form  $((E_1, C_1), \dots, (E_m, C_m))$  with  $m = \lceil \frac{n}{2} \rceil$   
 614 and  $(E_i, C_i) = (F_{2i+1}, \bigcup_{j \leq i} F_{2j})$ , for every  $0 \leq i < m$ . Therefore, the intersection  
 615 language of  $\times_{i \in W} \mathcal{A}^i$  can be recognized by a Street automaton over the same set of  
 616 states and transitions and the concatenation of all the Streett conditions determined  
 617 by the parity conditions of the players in  $W$ . The overall translation is a DSW au-  
 618 tomaton with a number of Streett pairs being logarithmic in the number of its states,  
 619 whose emptiness can be solved in polynomial time [35]. Finally, as we fixed  $W$  at the  
 620 *beginning*, all we need to do is to use the procedure just described for each  $W \subseteq N$ ,  
 621 if needed (see Algorithm 1), obtaining an *optimal* decision procedure that has only  
 622 exponential time and polynomial space complexity in  $|N|$ , the number of agents in  
 623 the system.<sup>7</sup>

624 Concerning the complexity analysis, consider again Algorithm 1 and denote by  
 625  $n$  the number of agents and  $|\text{St}_{\text{LTL}}|$  the number of states. Observe that Line 3 of the  
 626 algorithm builds a multi-agent parity game  $\mathcal{G}_{\text{PAR}}$  by making the product construction  
 627 between  $\mathcal{G}_{\text{LTL}}$  and all the DPW automata  $\mathcal{A}_{\gamma_i}$ , whose state space is  $2^{2^{|\gamma_i|}}$ , and the  
 628 number of priorities is  $2^{|\gamma_i|}$ . Thus, the number of states of  $\mathcal{G}_{\text{PAR}}$  is  $|\text{St}_{\text{PAR}}| = |\text{St}_{\text{LTL}}| \cdot$   
 629  $2^{2^{|\gamma_1|}} \cdot \dots \cdot 2^{2^{|\gamma_n|}}$ . Now, on the one hand, Line 6 requires to solve a parity game on  
 630 the state-graph of  $\mathcal{G}_{\text{PAR}}$  with  $2^{\gamma_i}$  priorities. This is solved by applying Zielonka's  
 631 algorithm [37], that works in time  $(|\text{St}_{\text{PAR}}|)^2 \cdot (|\text{St}_{\text{PAR}}|)^{2^{\gamma_i}}$ , thus polynomial in the  
 632 state space of  $\mathcal{G}_{\text{PAR}}$  and doubly exponential in the size of objectives  $\gamma_i$ 's. On the  
 633 other hand, Line 12 calls for the Non-Emptiness procedure of a DSW whose number  
 634 of Street pairs is linear in the sum of priorities of the automata  $\mathcal{A}_{\gamma_1}, \dots, \mathcal{A}_{\gamma_n}$  and so  
 635 logarithmic in its state-space (that is doubly exponential in the size of the objectives).  
 636 Such procedure is polynomial in the state space of the automaton [35, Corollary 10.8]

---

<sup>7</sup>Some previous techniques, e.g. [36], to the computation of pure Nash equilibria are not optimal as they  
 have exponential space complexity in the number of players  $|N|$ .

and therefore polynomial in  $|\text{St}_{\text{PAR}}|$ . Finally, consider the loops of Line 4 and Line 5, respectively. The first is on all the possible subsets of agents, and thus of length  $2^n$ . The second is on all the possible agents, and thus of length  $n$ . This sums up to an overall complexity for Algorithm 1 of:

$$2^N \cdot N \cdot (|\text{St}_{\text{PAR}}|)^2 \cdot (|\text{St}_{\text{PAR}}|)^{\sum_{i \in N} 2^{\gamma_i}} + |\text{St}_{\text{PAR}}|.$$

Recall that  $|\text{St}_{\text{PAR}}|$  is linear in the set of states of the  $\mathcal{G}_{\text{LTL}}$  and doubly exponential in every objective  $\gamma_i$ 's of the agents. Thus, the procedure is *polynomial* in  $|\text{St}_{\text{LTL}}|$ , exponential in  $N$ , and doubly exponential in the size of the formulas  $|\gamma_1|, \dots, |\gamma_N|$ .

## 7. Synthesis and Verification

We now show how to solve the synthesis and verification problems using NON-EMPTYNESS. For *synthesis*, the solution is already contained in the proof of Theorem 3, so we only need to sketch out the approach 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  $\pi$  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  $\sigma_i$  for each player  $i \in N$  in the game (*i.e.*, including those not in  $W$ ), as follows: while no deviation occurs, play the action that contributes to generate  $\pi$ , and if a deviation of player  $j$  occurs, then play the (memoryless) strategy  $\sigma_i^{\text{pun}j}$  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).

664 For *verification*, one can use a reduction of the following two problems, called  
 665 E-NASH and A-NASH in [15, 8, 7], to NON-EMPTYNESS.

666 *Given:* Game  $\mathcal{G}_{\text{LTL}}$ , LTL formula  $\varphi$ .

667 E-NASH: Is it the case that  $\pi(\vec{\sigma}) \models \varphi$ , for some  $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\text{LTL}})$  ?

668 A-NASH: Is it the case that  $\pi(\vec{\sigma}) \models \varphi$ , for all  $\vec{\sigma} \in \text{NE}(\mathcal{G}_{\text{LTL}})$  ?

669 We write  $(\mathcal{G}_{\text{LTL}}, \varphi) \in \text{E-NASH}$  to denote that  $(\mathcal{G}_{\text{LTL}}, \varphi)$  is an instance of E-NASH, *i.e.*,  
 670 given a game  $\mathcal{G}_{\text{LTL}}$  and a LTL formula  $\varphi$ , the answer to E-NASH problem is a “yes”;  
 671 and, similarly for A-NASH.

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

677 In order to solve E-NASH and A-NASH via NON-EMPTYNESS, one could use the  
 678 following result, whose proof is a simple adaptation of the same result for iterated  
 679 Boolean games [15] and for multi-player games with LTL goals modelled using SRML [7],  
 680 which was first presented in [38].

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

683 However, since we have Algorithm 1 at our disposal, an easier – and more direct  
 684 – solution can be obtained. To solve E-NASH we can modify line 12 of Algorithm 1  
 685 to include the restriction that such an algorithm, which now receives  $\varphi$  as a param-  
 686 eter, returns “Yes” in line 13 if and only if  $\varphi$  is satisfied in some run in the set of  
 687 Nash equilibrium witnesses. The new line 12 is “if  $\mathcal{L}(\times_{i \in W} (\mathcal{S}_i) \times \mathcal{S}_\varphi) \neq \emptyset$ ”, where  
 688  $\mathcal{S}_\varphi$  is the DSW automaton representing  $\varphi$ . All complexities remain the same; the  
 689 modified algorithm for E-NASH is denoted as Algorithm 1'. We can then use Algo-  
 690 rithm 1' to solve A-NASH, also as described in [38]: essentially, we can check whether  
 691 Algorithm 1'( $\mathcal{G}_{\text{LTL}}, \neg\varphi$ ) returns “No” in line 16. If it does, then no Nash equilibrium  
 692 of  $\mathcal{G}_{\text{LTL}}$  satisfies  $\neg\varphi$ , either because no Nash equilibrium exists at all (thus, A-NASH

693 is vacuously true) or because all Nash equilibria of  $\mathcal{G}_{\text{LTL}}$  satisfy  $\varphi$ , then solving A-  
 694 NASH positively. Note that in this case, since A-NASH is solved positively when the  
 695 algorithm returns “No” in line 16, then no specific Nash equilibrium strategy profile  
 696 is synthesised, as expected. However, if the algorithm returns “Yes”, that is, the case  
 697 when the answer to A-NASH problem with  $(\mathcal{G}_{\text{LTL}}, \varphi)$  instance is negative, then a strat-  
 698 egy profile is synthesised from Algorithm 1’ which corresponds to a counter-example  
 699 for  $(\mathcal{G}_{\text{LTL}}, \varphi) \in \text{A-NASH}$ . It should be easy to see that implementing E-NASH and A-  
 700 NASH is straightforward from Algorithm 1. Also, as already known, it is also easy to  
 701 see that Algorithm 1’ solves NON-EMPTYNESS if and only if  $(\mathcal{G}_{\text{LTL}}, \top) \in \text{E-NASH}$ .

## 702 8. Implementation

703 We have implemented the decision procedures presented in this paper. Our im-  
 704 plementation uses SRML [10] as a modelling language. SRML is based on the Reac-  
 705 tive Modules language [24] which is used in a number of verification tools, including  
 706 PRISM [26] and MOCHA [25]. The tool that implements our algorithms is called EVE  
 707 (for *Equilibrium Verification Environment*) [23]. EVE is the *first and only tool* able  
 708 to analyse the linear temporal logic properties that hold in equilibrium in a concur-  
 709 rent, reactive, and multi-agent system within a bisimulation-invariant framework. It  
 710 is also the only tool that supports all of the following combined features: a high-level  
 711 description language using SRML, general-sum multi-player games with LTL goals,  
 712 bisimulation-invariant strategies, and perfect recall. It is also the only tool for Nash  
 713 equilibrium analysis that relies on a procedure based on the solution of parity games,  
 714 which has allowed us to solve the (rational) synthesis problem for individual play-  
 715 ers in the system using very powerful techniques originally developed to solve the  
 716 synthesis problem from (linear-time) temporal logic specifications.

717 To the best of our knowledge, there are only two other tools that can be used  
 718 to reason about temporal logic equilibrium properties of concurrent/multi-agent sys-  
 719 tems: PRALINE [39] and MCMAS [40, 41].

720 PRALINE allows one to compute a Nash equilibrium in a game played in a con-  
 721 current game structure [39]. The underlying technique uses alternating Büchi au-

722 tomata and relies on the solution of a two-player zero-sum game called the ‘suspect  
 723 game’ [36]. PRALINE can be used to analyse games with different kinds of players  
 724 goals (e.g., reachability, safety, and others), but does not permit LTL goals, and does  
 725 not compute bisimulation-invariant strategies.

726 MCMAS is a model checking tool for multi-agent systems [42]. Since it can be  
 727 used to model check Strategy Logic (SL [12]) formulae [41], and SL can express the  
 728 existence of a Nash equilibrium, one can model a multi-agent system in MCMAS and  
 729 check for the existence of a Nash equilibrium in such a system using SL. However, MC-  
 730 MAS only supports SL with memoryless strategies (while our implementation does  
 731 not have this restriction) and, as PRALINE, does not compute bisimulation-invariant  
 732 strategies either.

733 From the many differences between PRALINE, MCMAS, and EVE (and their asso-  
 734 ciated underlying reasoning and verification techniques), one of the most important  
 735 ones is bisimulation-invariance, a feature needed to be able to do verification and syn-  
 736 thesis, e.g., when using symbolic methods with OBDDs or some model-minimisation  
 737 techniques. Not being bisimulation-invariant also means that in some cases PRALINE,  
 738 MCMAS, and EVE would deliver completely different answers. For instance, unlike  
 739 EVE, with PRALINE and MCMAS it may be the case that for two bisimilar systems  
 740 PRALINE and MCMAS would compute a Nash equilibrium in one of them and none  
 741 in the other. A particular instance is the “motivating example” in [28]. Since the two  
 742 systems there are bisimilar, EVE is able to compute a bisimulation-invariant Nash  
 743 equilibrium in both systems, while PRALINE and MCMAS, both of which are not us-  
 744 ing bisimulation-invariant model of strategies, cannot. The experiment supporting  
 745 this claim is reported in Section 8.4 along with the performance results. Indeed, even  
 746 in cases where all tools are able to compute a Nash equilibrium, EVE outperforms the  
 747 other two tools as the size of the input system grows, despite the fact that the model  
 748 of strategies we use in our procedure is *richer* in the sense that it takes into account  
 749 more information of the underlying game.<sup>8</sup>

---

<sup>8</sup>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.



### 750 8.1. Tool Description

751 *Modelling Language.* Systems in EVE are specified with the *Simple Reactive Modules*  
 752 *Language* (SRML [10]), that can be used to model non-deterministic systems. Each  
 753 system component (agent/player) in SRML is represented as a *module*, which con-  
 754 sists of an *interface* that defines the name of the module and lists a non-empty set of  
 755 Boolean variables controlled by the module, and a set of *guarded commands*, which de-  
 756 fine the choices available to the module at each state. There are two kinds of guarded  
 757 commands: **init**, used for initialising the variables, and **update**, used for updating  
 758 variables subsequently.

759 A guarded command has two parts: a “condition” part (the “guard”) and an “ac-  
 760 tion” part. The “guard” determines whether a guarded command can be executed or  
 761 not given the current state, while the “action” part defines how to update the value  
 762 of (some of) the variables controlled by a corresponding module. Intuitively,  $\varphi \rightsquigarrow \alpha$   
 763 can be read as “if the condition  $\varphi$  is satisfied, then *one* of the choices available to the  
 764 module is to execute  $\alpha$ ”. Note that the value of  $\varphi$  being true does not guarantee the  
 765 execution of  $\alpha$ , but only that it is *enabled* for execution, and thus *may be chosen*. If  
 766 no guarded command of a module is enabled in some state, then that module has no  
 767 choice and the values of the variables controlled by it remain unchanged in the next  
 768 state.

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  $ctr(g) \subseteq \Phi_i$ . A  
 guarded command  $g$  over a set of variables  $\Phi$  is an expression

$$g : \quad \varphi \rightsquigarrow x'_1 := \psi_1; \dots; x'_k := \psi_k$$

where the guard  $\varphi$  is a propositional logic formula over  $\Phi$ , each  $x_i$  is a member of  
 $\Phi$  and  $\psi_i$  is a propositional logic formula over  $\Phi$ . 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

```

module toggle controls  $x$ 

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

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

```

Figure 5: Example of module toggle in SRML.

variables  $x_1, \dots, x_k$  are controlled variables in  $g \in U_i$  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:

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

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

773 Figure 5 shows a module named *toggle* that controls a Boolean variable named  
 774  $x$ . There are two **init** guarded commands and two **update** guarded commands. The  
 775 **init** guarded commands define two choices for the initialisation of variable  $x$ : true or  
 776 false. The first **update** guarded command says that if  $x$  has the value of true, then  
 777 the corresponding choice is to assign it to false, while the second command says that  
 778 if  $x$  has the value of false, then it can be assigned to true. Intuitively, the module  
 779 would choose (in a non-deterministic manner) an initial value for  $x$ , and then on  
 780 subsequent rounds toggles this value. In this particular example, the **init** commands  
 781 are non-deterministic, while the **update** commands are deterministic. We refer to [7]  
 782 for further details on the semantics of SRML. In particular, in Figure 12 of [7], we detail

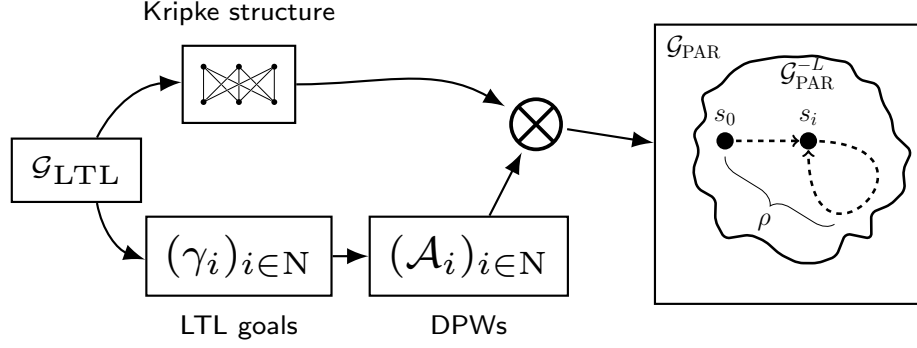


Figure 6: High-level workflow of EVE.

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.

At this point, readers might notice that the way SRML modules are defined leads to the possibility of having multiple initial states – which appears to contradict the definition of CMGS. However, this is not a problem, since we can always add an extra “pre”-initial state whose outgoing edges are labelled according to init guarded commands, and use it as the “real” initial state.

**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-EMPTYNESS. The main idea behind this algorithm is illustrated in Figure 6. The general flow of the implementation is as follows. Let  $\mathcal{G}_{\text{LTL}}$  be a game, modelled using SRML, with a set of players/modules  $\mathbb{N} = \{1, \dots, n\}$  and LTL goals  $\Gamma = \{\gamma_1, \dots, \gamma_n\}$ , one for each player. Using  $\mathcal{G}_{\text{LTL}}$  we construct an associated concurrent game with parity goals  $\mathcal{G}_{\text{PAR}}$  in order to shift reasoning on the set of Nash equilibria of  $\mathcal{G}_{\text{LTL}}$  into the set of Nash equilibria of  $\mathcal{G}_{\text{PAR}}$ . The basic idea of this construction is, firstly, to transform all LTL goals in  $\mathcal{G}_{\text{LTL}}$  into deterministic parity word (DPW) automata. To do this, we use LTL2BA tool [43, 44] to transform the formulae into nondeterministic Büchi word (NBW) automata. From NBWs, we construct the associated deterministic parity word (DPW) automata via construction described in

[33]. Secondly, to perform a product construction of the Kripke structure that represents  $\mathcal{G}_{\text{LTL}}$  with the collection of DPWs in which the set of Nash equilibria of the input game is preserved. With  $\mathcal{G}_{\text{PAR}}$  in our hands, we can then reason about Nash equilibria by solving a collection of parity games. To solve these parity games, we use PGSolver tool [45, 46]. EVE then iterates through all possible set of “winners”  $W \subseteq N$  (Algorithm 1 line 4) and computes a punishment region  $\text{Pun}_j(\mathcal{G}_{\text{PAR}})$  for each  $j \in L = N \setminus W$ , with which a reduced parity game  $\mathcal{G}_{\text{PAR}}^{-L} = \bigcap_{j \in L} \text{Pun}_j(\mathcal{G}_{\text{PAR}})$  is built. Notice that for each player  $j$ ,  $\text{Pun}_j(\mathcal{G}_{\text{PAR}})$  need only computed once and can be stored, thus resulting in a more efficient running time. Lastly, EVE checks whether there exists a path  $\rho$  in  $\mathcal{G}_{\text{PAR}}^{-L}$  that satisfies the goals of each  $i \in W$ . To do this, we translate  $\mathcal{G}_{\text{PAR}}^{-L}$  into a deterministic Streett automata, whose language is empty if and only if so is the set of Nash equilibria of  $\mathcal{G}_{\text{PAR}}$ . For E-NASH problem, we simply need to find a run in the witness returned when we check for NON-EMPTYNESS; this can be done via automata intersection<sup>9</sup>.

EVE was developed in Python and available online from [9]. EVE takes as input a concurrent and multi-agent system described in SRML code, with player goals and a property  $\varphi$  to be checked specified in LTL. For NON-EMPTYNESS, 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  $\varphi$  holds on *some (every)* Nash equilibrium of the system, and “NO” otherwise.

In the next subsection, we present some case studies to evaluate the performance of EVE. The case studies are based on distributed and concurrent systems that can naturally be modelled as multi-agent systems. We note, however, that such case studies bear no special relevance to multi-agent systems research. Instead, our only purpose is to use such case studies and multi-agent systems to evaluate EVE’s *performance*, rather than to solve problems of particular relevance in the AI or multi-agent systems literatures. Nevertheless, one could easily see that the case studies are based on systems that one can imagine to be found in many AI systems nowadays.

---

<sup>9</sup>For A-NASH is straightforward, since it is the dual of E-NASH.

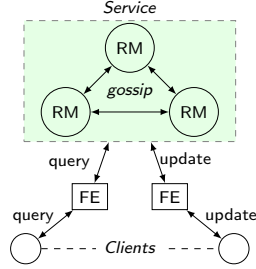


Figure 7: Gossip framework structure.

```

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

## 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 [7]), the second one is played on a graph, *e.g.*, as done in [47] 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. Furthermore, since PRALINE and MCMAS use different modelling languages – ISPL in the case of MCMAS – we need to translate the examples modelled in SRML into PRALINE's input language and ISPL. Given the high-level nature of SRML, the translation might introduce exponential blowup. However, we argue that this is not a problem from the comparison point of view, since the exponential blowup is also unavoidable when building Kripke structures from SRML games.

*Gossip protocols.* These are a class of networking and communication protocols that mimic the way social networks disseminate information. They have been used to solve problems in many large-scale distributed systems, such as *peer-to-peer* and *cloud* computing systems. Ladin *et al.* [48] developed a framework to provide high availability services via replication which is based on the gossip approach first introduced in [49, 50]. The main feature of this framework is the use of *replica managers* (RMs)

852 which exchange “gossip” messages periodically in order to keep the data updated. The  
 853 architecture of such an approach is shown in Figure 7.

854 We can model each RM as a module in SRML as follows: (1) When in *servicing*  
 855 *mode*, an RM can choose either to keep in servicing mode or to switch to gossiping  
 856 mode; (2) If it is in gossiping mode and there is at least another RM also in gossiping  
 857 mode<sup>10</sup>, since the information during gossip exchange is of (small) bounded size, it  
 858 goes back to servicing mode in the subsequent step. We then set the goal of each RM  
 859 to be able to gossip infinitely often. As shown in Figure 8, the module RM1 controls  
 860 a variable:  $s1$ . Its value being true signifies that RM1 is in servicing mode; other-  
 861 wise, it is in gossiping mode. Behaviour (1) is reflected in the first and second update  
 862 commands, while behaviour (2) is reflected in the third update command. The goal of  
 863 RM1 is specified with the LTL formula  $\mathbf{GF} \neg s1$ , which expresses that RM1’s goal is  
 864 to gossip infinitely often: “always” (G) “eventually” (F) gossip ( $\neg s1$ ).

865 Observe that with all RMs rationally pursuing their goals, they will adopt any  
 866 strategy which induces a run where each RM can gossip (with at least one other RM)  
 867 infinitely often. In fact, this kind of game-like modelling gives rise to a powerful  
 868 characteristic: on *all* runs that are sustained by a Nash equilibrium, the distributed  
 869 system is guaranteed to have two crucial *non-starvation/liveness* properties: RMs can  
 870 gossip infinitely often and clients can be served infinitely often. Indeed, these prop-  
 871 erties are verified in the experiments; with E-NASH: no Nash equilibrium sustains “all  
 872 RMs forever gossiping”; and with A-NASH: in all Nash equilibria at least one of the  
 873 RM is in servicing mode infinitely often. We also notice that each RM is modelled as  
 874 a non-deterministic open system: non-determinism is used in the first two updated  
 875 commands, as they have the same guard  $s1$  and therefore will be both enabled at the  
 876 same time; and the system is open since each module’s state space and choices depend  
 877 on the states of other modules, as reflected by the third updated command.

878 *Replica Control Protocol.* Consensus is a key issue in distributed computing and multi-  
 879 agent systems. An important application domain is in maintaining data consistency.

---

<sup>10</sup>The core of the protocol involves (at least) pairwise interactions periodically.

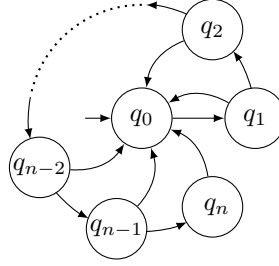


Figure 9: Gifford's protocol modelled as a game.

Gifford [51] proposed a quorum-based voting protocol to ensure data consistency by not allowing more than one processes to read/write a data item concurrently. To do this, each copy of a replicated item is assigned a vote.

We can model a (modified version of) Gifford's protocol as a game as follows. The set of players  $N = \{1, \dots, n\}$  in the game is arranged in a request queue represented by the sequence of states  $q_1, \dots, q_n$ , where  $q_i$  means that player  $i$  is requesting to read/write the data item. At state  $q_i$ , other players in  $N \setminus \{i\}$  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  $q_{i+1}$ <sup>11</sup>. The voting process then restarts from  $q_1$ . The protocol's structure is shown in Figure 9. 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 equilibrium in which the data is never updated; and, with A-NASH: on all Nash equilibria, for each player, its request will be granted infinitely often. Also, in this example, we define a module, called "Environment", which is used to represent the underlying concurrent game structure, shown in Figure 9, where the game is played.

<sup>11</sup>We assume arithmetic modulo  $(|N| + 1)$  in this example.

Table 1: Gossip Protocol experiment results.

P	S	E	EVE			PRALINE			MCMAS		
			$\nu$ (s)	$\epsilon$ (s)	$\alpha$ (s)	$\nu$ (s)	$\epsilon$ (s)	$\alpha$ (s)	$\nu$ (s)	$\epsilon$ (s)	$\alpha$ (s)
2	4	9	0.02	0.24	0.08	0.02	1.71	1.73	<b>0.01</b>	<b>0.01</b>	<b>0.01</b>
3	8	27	0.09	0.43	0.26	0.33	26.74	27.85	<b>0.02</b>	<b>0.06</b>	<b>0.06</b>
4	16	81	<b>0.42</b>	<b>3.51</b>	<b>1.41</b>	0.76	547.97	548.82	760.65	3257.56	3272.57
5	32	243	<b>2.30</b>	<b>35.80</b>	<b>25.77</b>	10.06	TO	TO	TO	TO	TO
6	64	729	<b>16.63</b>	<b>633.68</b>	<b>336.42</b>	255.02	TO	TO	TO	TO	TO
7	128	2187	<b>203.05</b>	TO	TO	5156.48	TO	TO	TO	TO	TO
8	256	6561	<b>4697.49</b>	TO	TO	TO	TO	TO	TO	TO	TO

Table 2: Replica control experiment results.

P	S	E	EVE			PRALINE			MCMAS		
			$\nu$ (s)	$\epsilon$ (s)	$\alpha$ (s)	$\nu$ (s)	$\epsilon$ (s)	$\alpha$ (s)	$\nu$ (s)	$\epsilon$ (s)	$\alpha$ (s)
2	3	8	0.04	0.11	0.10	0.05	0.64	0.74	<b>0.01</b>	<b>0.01</b>	<b>0.02</b>
3	4	20	0.11	1.53	0.22	0.12	4.96	5.46	<b>0.02</b>	<b>0.06</b>	<b>0.11</b>
4	5	48	<b>0.34</b>	<b>1.73</b>	<b>0.68</b>	0.56	65.50	67.45	1.99	4.15	11.28
5	6	112	<b>1.43</b>	<b>2.66</b>	<b>2.91</b>	6.86	1546.90	1554.80	1728.73	6590.53	TO
6	7	256	<b>5.87</b>	<b>13.69</b>	<b>16.03</b>	94.39	TO	TO	TO	TO	TO
7	8	576	<b>32.84</b>	<b>76.50</b>	<b>102.12</b>	2159.88	TO	TO	TO	TO	TO
8	9	1280	<b>166.60</b>	<b>485.99</b>	<b>746.55</b>	TO	TO	TO	TO	TO	TO

### 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<sup>12</sup> for solving NON-  
 EMPTINESS (“ $\nu$ ”), E-NASH (“ $\epsilon$ ”), and A-NASH (“ $\alpha$ ”). 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 [38]. Intuitively, the reduction is as follows:  
 given a game  $G$  and formula  $\varphi$ , we construct a new game  $H$  with two additional agents,  
 say  $n + 1$  and  $n + 2$ , with goals  $\gamma_{n+1} = \varphi \vee (p \leftrightarrow q)$  and  $\gamma_{n+2} = \varphi \vee \neg(p \leftrightarrow q)$ ,  
 where  $\Phi_{n+1} = \{p\}$  and  $\Phi_{n+2} = \{q\}$ ,  $p$  and  $q$  are fresh Boolean variables. This means  
 that it is the case  $\text{NE}(H) \neq \emptyset$  if and only if there exists a Nash equilibrium run in  $G$   
 satisfying  $\varphi$ .

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 PRA-  
 LINE performed better than MCMAS, both struggled (timed-out<sup>13</sup>) with inputs with  
 more than 100 edges, while EVE could handle up to 6000 edges (for NON-EMPTINESS).

#### 8.4. Experiment II

This experiment is taken from the motivating examples in [28]. Suppose the sys-  
 tems shown in Figure 10 and 11 represents a 3-player game, where each transition  
 is labelled by the actions  $x, y, z$  of player 1, 2, and 3, respectively, an asterisk  $*$  be-  
 ing a wildcard. The goals of the players can be represented by the LTL formulae  
 $\gamma_1 = \mathbf{F}p$ ,  $\gamma_2 = \mathbf{F}q$ , and  $\gamma_3 = \mathbf{G}\neg(p \vee q)$ . The system in Figure 10 has a Nash equi-  
 librium, whereas no (non-bisimulation-invariant strategies) Nash equilibria exists in  
 the (bisimilar) system in Figure 11.

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

<sup>12</sup>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.

<sup>13</sup>Time-out was fixed to be 7200 seconds.

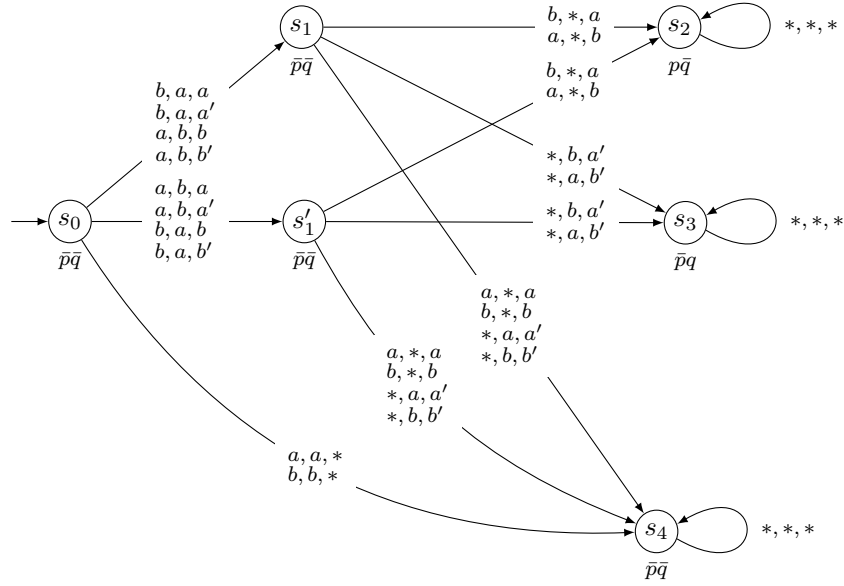


Figure 10: A 3-player game with Nash equilibrium.

933 states and edges; then, for each case, we report (i) the total running time<sup>14</sup> (“time”)   
 934 and (ii) whether the tools find any Nash equilibria (“NE”).

935 Table 3 shows the results of the experiments on the example in which the model   
 936 of strategies that depends only on the run (sequence of states) of the game (run-based   
 937 strategies [28]) cannot sustain any Nash equilibria, a model of strategies that is not   
 938 invariant under bisimilarity. Indeed, since MCMAS and PRALINE use this model of   
 939 strategies, both did not find any Nash equilibria in the game, as shown in Table 3.   
 940 EVE, which uses a model of strategies that not only depends on the run of the game   
 941 but also on the actions of players (computation-based [28]), found a Nash equilibrium   
 942 in the game. We can also see that EVE outperformed MCMAS on games with 14 or

<sup>14</sup>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.

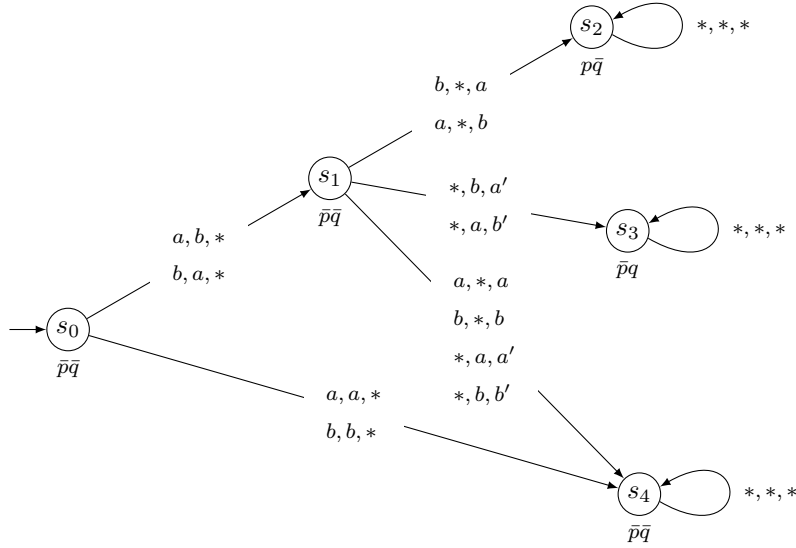


Figure 11: A 3-player game without (non-bisimulation-invariant strategies) Nash equilibria.

more states. In fact, MCMAS timed-out<sup>15</sup> 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.

<sup>15</sup>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	<b>0.04</b>	No	0.75	Yes	0.77	No
8	128	<b>0.24</b>	No	2.99	Yes	2.06	No
11	176	6.28	No	<b>3.86</b>	Yes	4.42	No
14	224	273.14	No	<b>7.46</b>	Yes	8.53	No
17	272	TO	–	<b>13.31</b>	Yes	15.33	No
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
50	800	TO	–	<b>655.80</b>	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	<b>0.02</b>	Yes	1.09	Yes	1.19	Yes
9	144	<b>0.77</b>	Yes	3.36	Yes	3.76	Yes
12	192	65.31	No	<b>7.45</b>	Yes	8.89	Yes
15	240	TO	–	<b>15.52</b>	Yes	17.72	Yes
18	288	TO	–	<b>30.06</b>	Yes	30.53	Yes
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
51	816	TO	–	<b>1314.47</b>	Yes	1563.79	Yes

### 955 8.5. Experiment III

This experiment is based on the example previously presented in Section 2. For this particular experiment, we assume that 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  $(n-1, n-1)$ . A number of obstacles are also placed (uniformly) randomly on the grid. We use a binary encoding to represent

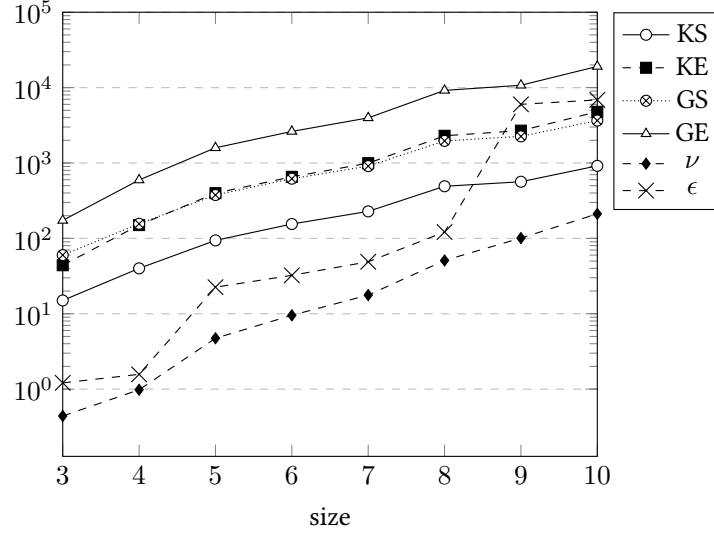


Figure 12: Plots from Table 5. Y-axis is in logarithmic scale.

the spatial information of the grid world which includes the grid coordinates, as well as the obstacles and the agents locations. For instance, to encode a position of an agent 1 in  $4 \times 4$  grid, we need 4 Boolean variables arranged as a tuple  $pos_1 = \langle x_0^1, x_1^1, y_0^1, y_1^1 \rangle$ . An instance of such a tuple  $pos_1 = \langle 0, 1, 1, 0 \rangle$  means that agent 1 is at  $(2, 1)$ . For each time step and  $i \in \{1, 2\}$ , the update guarded command set  $U_i$  is such a way that agent  $i$  can only move horizontally and vertically, 1 step at a time. Furthermore, the commands in  $U_i$  respect the legality of movement, *i.e.*, agent  $i$  cannot move out of bound or into an obstacle. The goal of each agent can be expressed by the LTL formulae

$$\gamma_1 = \mathbf{F} \left( \bigwedge_{i \in \{0, \dots, n-1\}} x_i^1 \wedge \bigwedge_{i \in \{0, \dots, n-1\}} y_i^1 \right)$$

and

$$\gamma_2 = \mathbf{F} \left( \bigwedge_{i \in \{0, \dots, n-1\}} \neg x_i^2 \wedge \bigwedge_{i \in \{0, \dots, n-1\}} \neg y_i^2 \right).$$

A safety specification (no more than one agent occupying the same position at the same time) can be expressed by the following LTL formula:

$$\varphi = \mathbf{G} \neg \left( \bigwedge_{i \in \{0, \dots, n-1\}} (x_i^1 \leftrightarrow x_i^2) \wedge \bigwedge_{i \in \{0, \dots, n-1\}} (y_i^1 \leftrightarrow y_i^2) \right).$$

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	$\nu$ (s)	$\epsilon$ (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)

956 The experiment was obtained on a PC with Intel i5-4690S CPU 3.20 GHz machine  
 957 with 8 GB of RAM running Linux kernel version 4.12.14-300.fc26.x86\_64. We varied  
 958 the size of the grid world (“size”) from  $3 \times 3$  to  $10 \times 10$ , each with a fixed number  
 959 of obstacles (“# Obs”), randomly distributed on the grid. We report the number of  
 960 Kripke states (“KS”), Kripke edges (“KE”),  $\mathcal{G}_{\text{PAR}}$  states (“GS”),  $\mathcal{G}_{\text{PAR}}$  edges (“GE”),  
 961 NON-EMPTYNESS execution time (“ $\nu$ ”), and E-NASH execution time (“ $\epsilon$ ”). We ran the  
 962 experiment for five replications, and report the average (*ave*), minimum (*min*), and

963 maximum (*max*) times from the replications. The results are reported in Table 5, with  
964 the following format: *ave(min, max)*.

965 From the experiment results, we see that EVE works well for NON-EMPTINESS up  
966 until size 10. From the plots in Figure 12, we can clearly see that the values of each  
967 variable, except for  $\epsilon$ , grow exponentially. For  $\epsilon$  (E-NASH), however, it seems to grow  
968 faster than the rest. Specifically, it is clearly visible in transitions between numbers  
969 that have different size of bit representation, *i.e.*, 4 to 5 and 8 to 9<sup>16</sup>. These jumps  
970 correspond to the time used to build deterministic parity automata on words from  
971 LTL properties to be checked in E-NASH, which is essentially, bit-for-bit comparisons  
972 between the position of agent 1 and 2.

973 From the experiments shown in this section it is also clear that the bottleneck in  
974 the performance is the translation of LTL goals and the high-level description of the  
975 game into the underlying parity game. Once an explicit parity game is constructed,  
976 then the performance improves radically. This result is perfectly consistent with what  
977 the theoretical complexity of the decision procedure predicts: our algorithm works  
978 in doubly-exponential time in the size of the goals of the players, while it is only  
979 singly-exponential in the size of the SRML specification. These two exponential-time  
980 reductions are in fact optimal, so there is no hope that they can be improved, at least  
981 in theory. On the other hand, the actual subroutine that finds a Nash equilibrium and  
982 computes players' strategies from the parity games representation of the problem is  
983 rather efficient in theory – but still not known to be in polynomial time using the best  
984 algorithms to solve parity games. Then, it is clear that a natural way to make rational  
985 verification a feasible problem, in theory, is to look at cases where goals and/or game  
986 representations are simpler. Such study is conducted in [52], where several positive  
987 results on the complexity of solving the rational verification problem are obtained.

---

<sup>16</sup>Since the grid coordinate index starts at 0, the “actual” transitions are 3 to 4 and 7 to 8.

## 988 9. Concluding Remarks and Related Work

989 This paper contains a complete study, from theory to implementation, of the tem-  
990 poral equilibrium analysis of multi-agent AI systems formally modelled as multi-  
991 player games. The two main contributions of the paper are: (1) a novel and optimal  
992 decision procedure, based on the solution of parity games, that can be used to solve  
993 both the rational verification and the automated synthesis problems for multi-player  
994 games; and (2) a complete implementation of the general game-theoretic modelling  
995 and reasoning framework – with full support of goals expressed as LTL formulae and  
996 high-level game descriptions in SRML – which is available online. Our work builds  
997 on several previous results in the computer science (synthesis and verification) and AI  
998 literatures (multi-agent systems). Relevant related literature will be discussed next.

999 **Equilibrium Analysis in Multi-Agent Systems.** Rational verification was pro-  
1000 posed as an complementary verification methodology to conventional methods, such  
1001 as model checking. A legitimate question is, then, when is rational verification an ap-  
1002 propriate verification approach? A possible answer is given next. The verification  
1003 problem [1], as conventionally formulated, is concerned with checking that some  
1004 property, usually defined using a modal or a temporal logic [53], holds on some or  
1005 on every computation run of a system. In a game-theoretic setting, this can be a  
1006 very strong requirement – and in some cases even inappropriate – since only some  
1007 computations of the system will arise (be sustained) as the result of agents in the sys-  
1008 tem choosing strategies in equilibrium, that is, due to strategic and rational play. It  
1009 was precisely this concern that motivated the rational verification approach [7, 8].  
1010 In rational verification, we ask if a given temporal property holds on some or every  
1011 computation run that can be sustained by agents choosing Nash equilibrium strate-  
1012 gies. Rational verification can be reduced to the NON-EMPTYNESS problem, as stated  
1013 in this paper; cf., [38]. As a consequence, along with the polynomial transformations  
1014 in [38], our results provide a complete framework (theory, algorithms, and imple-  
1015 mentation) for automated temporal equilibrium analysis, specifically, to do rational  
1016 synthesis and formal verification of logic-based multi-agent systems. The framework,  
1017 in particular, provides a concrete and algorithmic solution to the rational synthesis



1018 problem as studied in [14], where the Boolean case (iterated games where players  
 1019 control Boolean variables, whose valuations define sequences of states in the game,  
 1020 *i.e.*, the plays in the game) was given an interesting automata-theoretic solution via  
 1021 (an extension of) Strategy Logic [16].

1022 **Automata and logic.** In computer science, a common technique to reason about  
 1023 Nash equilibria in multi-player games is using alternating parity *automata on infinite*  
 1024 *trees* (APTs [18]). This approach is used to do rational synthesis [14, 54]; equilibrium  
 1025 checking and rational verification [8, 15, 7]; and model checking of logics for strategic  
 1026 reasoning capable to specify the existence of a Nash equilibrium in concurrent game  
 1027 structures [47], both in two-player games [16, 55] and in multi-player games [56, 12].  
 1028 In cases where players’ goals are simpler than general LTL formulae, *e.g.*, for reacha-  
 1029 bility or safety goals, alternating Büchi automata can be used instead [36]. *Our tech-*  
 1030 *nique is different from all these automata-based approaches, and in some cases more*  
 1031 *general*, as it can be used to handle either a more complex model of strategies or a  
 1032 more complex type of goals, and delivers an immediate procedure to synthesise indi-  
 1033 vidual strategies for players in the game, while being amenable to implementation.

1034 **Tools and algorithms.** In theory, the kind of equilibrium analysis that can be done  
 1035 using MCMAS [40, 57, 58] and PRALINE [39, 36] rely on the automata-based approach.  
 1036 However, the algorithms that are actually implemented have a different flavour. MC-  
 1037 MAS uses a procedure for SL which works as a *labelling algorithm* since it only consid-  
 1038 ers memoryless strategies [58]. On the other hand, PRALINE, which works for Büchi  
 1039 definable objectives, uses a procedure based on the “*suspect game*” [36]. Despite some  
 1040 similarities between our construction and the suspect game, introduced in [36], the  
 1041 two procedures are substantially different. Unlike our procedure, the suspect game is  
 1042 a standard two-player zero-sum turn-based game  $\mathcal{H}(\mathcal{G}, \pi)$ , constructed from a game  
 1043  $\mathcal{G}$  and a possible path  $\pi$ , in which one of the players (“Eve”) has a winning strategy  
 1044 if, and only if,  $\pi$  can be sustained by a Nash equilibrium in  $\mathcal{G}$ . The overall procedure  
 1045 in [36] relies on the construction of such a game, whose size (space complexity) is  
 1046 exponential in the number of agents [36, Section 4.3]. Instead, our procedure solves,  
 1047 independently, a collection of parity games that avoids an exponential use of space but

1048 may require to be executed exponentially many times. Key to the correctness of our  
 1049 approach is that we deal with parity conditions, which are prefix-independent, ensur-  
 1050 ing that punishment strategies do not depend on the history of the game. Regarding  
 1051 similarities, our procedure also checks for the existence of a path sustained by a Nash  
 1052 Equilibrium, but our algorithm does this for every subset  $W \subseteq N$  of agents, if needed.  
 1053 Doing this (*i.e.*, trading exponential space for exponential time), at every call of this  
 1054 subroutine, our algorithm avoids building an exponentially sized game, like  $\mathcal{H}$ . On the  
 1055 other hand, from a practical point of view, avoiding the construction of such an expo-  
 1056 nential sized game leads to better performance (running times), even in cases where  
 1057 no Nash equilibrium exists, when our subroutine is necessarily called exponentially  
 1058 many times. In addition to all of the above, neither the algorithm used for MCMAS nor  
 1059 the one used for PRALINE computes pure Nash equilibria in a bisimulation-invariant  
 1060 framework, as our procedure does. While MCMAS and PRALINE are the two closest  
 1061 tools to EVE, they are not the only available options to reason about games. For in-  
 1062 stance, PRISM-games [59], EAGLE [60], and UPPAAL [61] are other interesting tools  
 1063 to reason about games. PRISM-games allows one to do strategy synthesis for turn-  
 1064 based stochastic games as well as model checking for long-run, average, and ratio  
 1065 rewards properties. Only until very recently, PRISM-games had no support of equi-  
 1066 librium reasoning, but see [62]. EAGLE is a tool specifically designed to reason about  
 1067 pure Nash equilibria in multi-player games. EAGLE considers games where goals  
 1068 are given as CTL formulae and allows one to check if a given strategy profile is a  
 1069 Nash equilibrium of a given multi-agent system. This decision problem, called MEM-  
 1070 BERSHIP within the rational verification framework [8], is, theoretically, simpler than  
 1071 NON-EMPTYNESS: while the former can be solved in EXPTIME (for branching-time  
 1072 goals expressed using CTL formulae [13]), the latter is 2EXPTIME-complete for LTL  
 1073 goals, and even 2EXPTIME-hard for CTL goals and nondeterministic strategies [13].  
 1074 UPPAAL is another tool that can be used to analyse equilibrium behaviour in a sys-  
 1075 tem [63, 64]. However, UPPAAL differs from EVE in various critical ways: *e.g.*, it works  
 1076 in a quantitative setting, uses statistical model checking, and most importantly, com-  
 1077 putes approximate Nash equilibria of a game.

**The Role of Bisimilarity.** One crucial aspect of our approach to rational verification and synthesis is the role of *bisimilarity* [65, 31, 66, 67]. 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  $P_1, \dots, P_n$ . That is, that for every (temporal logic) property  $\varphi$  and every system component  $P'_i$  modelled as an agent in a multi-player game, if  $P'_i$  is bisimilar to  $P_i \in \{P_1, \dots, P_n\}$ , then  $\varphi$  is satisfied in equilibrium – that is, on a run induced by some Nash equilibrium of the game – by  $P_1, \dots, P_i, \dots, P_n$  if and only if it is also satisfied in equilibrium by  $P_1, \dots, P'_i, \dots, P_n$ , the system in which  $P_i$  is replaced by  $P'_i$ , that is, across all bisimilar systems to  $P_1, \dots, P_n$ . This property is called *invariance under bisimilarity*. Unfortunately, as shown in [34, 28], 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  $P_1, \dots, P_i, \dots, P_n$  may not, in general, be the same as building strategies for a bisimilar system  $P_1, \dots, P'_i, \dots, P_n$ , again, deterring any hope of being able to do modular reasoning on concurrent and multi-agent systems. These problems were first identified in [34] and further studied in [28]. However, no algorithmic solutions to these two problems were presented in either [34] or [28]. Specifically, in this paper, bisimilarity was exploited in two ways. Firstly, our construction of punishment strategies (used in the characterisation of Nash equilibrium given by Theorem 3) assumes that players have access to the history of choices that other players in the game have made. As shown in [28, 29], with a model of strategies where this is not the case, the preservation of Nash equilibria in the game, as well as of temporal logic properties in equilibrium, may not be guaranteed. Secondly, our implementation in

1109 EVE guarantees that any two games whose underlying CGSs are bisimilar, and there-  
 1110 fore should be regarded as observationally equivalent from a concurrency point of  
 1111 view, will produce the same answers to the rational verification and automated syn-  
 1112 thesis problems. It is also worth noting that even though bisimilarity is probably the  
 1113 most widely used behavioural equivalence in concurrency, in the context of multi-  
 1114 agent systems other relations may be preferred, for instance, equivalence relations  
 1115 that take a detailed account of the independent interactions and behaviour of indi-  
 1116 vidual components in a multi-agent system. In such a setting, “alternating” relations  
 1117 with natural ATL\* characterisations have been studied [68]. Alternating bisimulation  
 1118 is very similar to bisimilarity on labelled transition systems [65, 31], only that when  
 1119 defined on CGSs, instead of action profiles (directions) taken as possible transitions,  
 1120 one allows individual player’s actions, which must be matched in the bisimulation  
 1121 game. Because of this, it immediately follows that any alternating bisimulation as de-  
 1122 fined in [68] is also a bisimilarity as defined here. Despite having a different formal  
 1123 definition, a simple observation can be made: Nash equilibria are not preserved by  
 1124 the alternating (bisimulation) equivalence relations in [68] either, which discourages  
 1125 the use of these even stronger equivalence relations for multi-agent systems. In fact,  
 1126 as discussed in [69], the “right” notion of equivalence for games (which can be indi-  
 1127 rectly used as an observationally equivalence between multi-agent systems) and their  
 1128 game theoretic solution concepts is, undoubtedly, an important and interesting topic  
 1129 of debate, which deserves to be investigated further.

1130 **Some features of our framework.** Unlike other approaches to rational synthesis  
 1131 and temporal equilibrium analysis, *e.g.* [58, 36, 14, 7], we employ parity games [19],  
 1132 which are an intuitively *simple verification model* with an abundant associated set of  
 1133 algorithmic solutions [70]. In particular, strategies in our framework, as in [7], can  
 1134 depend on players’ actions, leading to a much richer game-theoretic setting where  
 1135 Nash equilibrium is invariant under bisimilarity [28, 29], a desirable property for con-  
 1136 current and reactive systems [65, 31, 66, 67]. Our reasoning and verification approach  
 1137 applies to multi-player games that are concurrent and synchronous, with perfect re-  
 1138 call and perfect information, and which can be represented in a high-level, succinct

manner using SRML [10]. In addition, 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 past players' actions. We do not consider mixed or non-deterministic 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 [17, 71].

**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 [72]. 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 [17, 73, 74, 75]. 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 [76, 77], as well as settings with more complex preference relations, for instance, using sets of goals given by linear, partial, or lexicographic orders as defined in [13, 14, 78], which would provide a strictly stronger modelling power.

*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) while working on this research at the University of Oxford. Part of this paper, focussing on the EVE system, presented at ATVA’18 [23].

## 1168 **References**

- 1169 [1] E. M. Clarke, O. Grumberg, D. A. Peled, Model Checking, MIT Press, Cambridge,  
1170 MA, USA, 2002.
- 1171 [2] M. Wooldridge, Introduction to Multiagent Systems, Wiley, Chichester, UK, 2001.
- 1172 [3] Y. Shoham, K. Leyton-Brown, Multiagent Systems: Algorithmic, Game-  
1173 Theoretic, and Logical Foundations, CUP, 2008.
- 1174 [4] M. J. Osborne, A. Rubinstein, A Course in Game Theory, MIT Press, 1994.
- 1175 [5] J. Y. Halpern, Beyond nash equilibrium: Solution concepts for the 21st century,  
1176 in: KR, AAAI Press, 2008, pp. 6–15.
- 1177 [6] I. Abraham, L. Alvisi, J. Y. Halpern, Distributed computing meets game theory:  
1178 combining insights from two fields, SIGACT News 42 (2) (2011) 69–76.
- 1179 [7] J. Gutierrez, P. Harrenstein, M. Wooldridge, From model checking to equilibrium  
1180 checking: Reactive modules for rational verification, Artificial Intelligence 248  
1181 (2017) 123–157.
- 1182 [8] M. Wooldridge, J. Gutierrez, P. Harrenstein, E. Marchioni, G. Perelli, A. Toumi,  
1183 Rational verification: From model checking to equilibrium checking, in: AAAI,  
1184 2016, pp. 4184–4191.
- 1185 [9] EVE: A tool for temporal equilibrium analysis, <https://github.com/eve-mas/eve-parity>,  
1186 accessed: 09-09-2019.
- 1187 [10] W. van der Hoek, A. Lomuscio, M. Wooldridge, On the complexity of practical  
1188 ATL model checking, in: AAMAS, ACM, 2006, pp. 201–208.
- 1189 [11] A. Pnueli, The temporal logic of programs, in: FOCS, IEEE, 1977, pp. 46–57.
- 1190 [12] F. Mogavero, A. Murano, G. Perelli, M. Y. Vardi, Reasoning about strategies: On  
1191 the model-checking problem, ACM Transactions on Computational Logic 15 (4)  
1192 (2014) 34:1–34:47.

- 1193 [13] J. Gutierrez, P. Harrenstein, M. Wooldridge, Reasoning about equilibria in game-  
1194 like concurrent systems, *Annals of Pure Applied Logic* 168 (2) (2017) 373–403.
- 1195 [14] D. Fisman, O. Kupferman, Y. Lustig, Rational synthesis, in: *TACAS*, Vol. 6015 of  
1196 LNCS, Springer, 2010, pp. 190–204.
- 1197 [15] J. Gutierrez, P. Harrenstein, M. Wooldridge, Iterated Boolean games, *Information*  
1198 *and Computation* 242 (2015) 53–79.
- 1199 [16] K. Chatterjee, T. A. Henzinger, N. Piterman, Strategy logic, *Information and*  
1200 *Computation* 208 (6) (2010) 677–693.
- 1201 [17] J. Gutierrez, G. Perelli, M. Wooldridge, Imperfect information in reactive mod-  
1202 ules games, *Information and Computation* 261 (Part) (2018) 650–675.
- 1203 [18] C. Löding, Basics on tree automata, in: *Modern Applications of Automata The-*  
1204 *ory*, Indian Institute of Science, Bangalore, India, 2012, pp. 79–110.
- 1205 [19] E. A. Emerson, C. S. Jutla, Tree automata, mu-calculus and determinacy, in:  
1206 *FOCS*, IEEE, 1991, pp. 368–377.
- 1207 [20] M. Jurdzinski, Deciding the winner in parity games is in  $UP \cap co-up$ , *Information*  
1208 *Processing Letters* 68 (3) (1998) 119–124.
- 1209 [21] C. S. Calude, S. Jain, B. Khoussainov, W. Li, F. Stephan, Deciding parity games in  
1210 quasipolynomial time, in: *STOC*, ACM, 2017, pp. 252–263.
- 1211 [22] O. Kupferman, Automata theory and model checking, *Handbook of TCS*.
- 1212 [23] J. Gutierrez, M. Najib, G. Perelli, M. Wooldridge, Eve: A tool for temporal equilib-  
1213 rium analysis, in: *ATVA*, Vol 11138 of LNCS, Springer, Cham, 2018, pp. 551–557.
- 1214 [24] R. Alur, T. A. Henzinger, Reactive modules, *Formal Methods in System Design*  
1215 15 (1) (1999) 7–48.
- 1216 [25] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, S. Tasiran,  
1217 *MOCHA: modularity in model checking*, in: *CAV*, Vol. 1427 of LNCS, Springer,  
1218 1998, pp. 521–525.

- 1219 [26] M. Z. Kwiatkowska, G. Norman, D. Parker, PRISM: probabilistic model checking  
1220 for performance and reliability analysis, SIGMETRICS Performance Evaluation  
1221 Review 36 (4) (2009) 40–45.
- 1222 [27] R. I. Brafman, C. Domshlak, From one to many: Planning for loosely coupled  
1223 multi-agent systems, in: Proceedings of the Eighteenth International Confer-  
1224 ence on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia,  
1225 September 14–18, 2008, 2008, pp. 28–35.
- 1226 [28] J. Gutierrez, P. Harrenstein, G. Perelli, M. Wooldridge, Nash equilibrium and  
1227 bisimulation invariance, in: CONCUR, Vol. 85 of LIPIcs, Schloss Dagstuhl, 2017,  
1228 pp. 17:1–17:16.
- 1229 [29] J. Gutierrez, P. Harrenstein, G. Perelli, M. J. Wooldridge, Nash equilibrium and  
1230 bisimulation invariance, Logical Methods in Computer Science 15 (3).
- 1231 [30] R. Milner, Communication and Concurrency, Prentice Hall, 1989.
- 1232 [31] M. Hennessy, R. Milner, Algebraic laws for nondeterminism and concurrency,  
1233 Journal of the ACM 32 (1) (1985) 137–161.
- 1234 [32] A. P. Sistla, M. Y. Vardi, P. Wolper, The complementation problem for Büchi au-  
1235 tomata with applications to temporal logic, Theoretical Computer Science 49  
1236 (1987) 217–237.
- 1237 [33] N. Piterman, From nondeterministic Büchi and Streett automata to deterministic  
1238 parity automata, Logical Methods in Computer Science 3 (3) (2007) 1–21.
- 1239 [34] J. Gutierrez, P. Harrenstein, M. Wooldridge, Expressiveness and complexity re-  
1240 sults for strategic reasoning, in: CONCUR, Vol. 42 of LIPIcs, Schloss Dagstuhl,  
1241 2015, pp. 268–282.
- 1242 [35] D. Perrin, J. Pin, Infinite Words., Pure and Applied Mathematics., Elsevier, 2004.
- 1243 [36] P. Bouyer, R. Brenguier, N. Markey, M. Ummels, Pure Nash equilibria in con-  
1244 current deterministic games, Logical Methods in Computer Science 11 (2) (2015)  
1245 1–72.



- [37] W. Zielonka, Infinite games on finitely coloured graphs with applications to automata on infinite trees, *Theoretical Computer Science* 200 (1-2) (1998) 135–183.
- [38] T. Gao, J. Gutierrez, M. Wooldridge, Iterated Boolean games for rational verification, in: *AAMAS, ACM*, 2017, pp. 705–713.
- [39] R. Brenguier, PRALINE: A tool for computing Nash equilibria in concurrent games, in: *CAV, Vol. 8044 of LNCS*, Springer, 2013, pp. 890–895.
- [40] 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.
- [41] 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.
- [42] 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.
- [43] P. Gastin, D. Oddoux, Fast LTL to Büchi automata translation, in: *CAV, Springer*, 2001, pp. 53–65.
- [44] LTL 2 BA: fast translation from LTL formulae to Büchi automata, <http://www.lsv.fr/~gastin/ltl2ba/>, accessed: 09-09-2019.
- [45] O. Friedmann, M. Lange, The pgsolver collection of parity game solvers – version 3 (2010).
- [46] PGSolver, <https://github.com/tcsprojects/pgsolver>, accessed: 09-09-2019.
- [47] R. Alur, T. A. Henzinger, O. Kupferman, Alternating-time temporal logic, *Journal of the ACM* 49 (5) (2002) 672–713.
- [48] R. Ladin, B. Liskov, L. Shriram, S. Ghemawat, Providing high availability using lazy replication, *ACM Transactions on Computer Systems* 10 (4) (1992) 360–391.

- 1272 [49] M. J. Fischer, A. Michael, Sacrificing serializability to attain high availability of  
1273 data in an unreliable network, in: PODS, ACM, New York, NY, USA, 1982, pp.  
1274 70–75.
- 1275 [50] G. T. Wu, A. J. Bernstein, Efficient solutions to the replicated log and dictionary  
1276 problems, in: PODC, ACM, New York, NY, USA, 1984, pp. 233–242.
- 1277 [51] D. K. Gifford, Weighted voting for replicated data, in: SOSP, ACM, New York,  
1278 NY, USA, 1979, pp. 150–162.
- 1279 [52] J. Gutierrez, M. Najib, G. Perelli, M. J. Wooldridge, On computational tractability  
1280 for rational verification, in: IJCAI, ijcai.org, 2019, pp. 329–335.
- 1281 [53] E. A. Emerson, Temporal and modal logic, in: Handbook of Theoretical Com-  
1282 puter Science, Volume B: Formal Models and Semantics (B), MIT Press, Cam-  
1283 bridge, MA, USA, 1990, pp. 995–1072.
- 1284 [54] O. Kupferman, G. Perelli, M. Y. Vardi, Synthesis with rational environments, An-  
1285 nals of Mathematics and Artificial Intelligence 78 (1) (2016) 3–20.
- 1286 [55] B. Finkbeiner, S. Schewe, Coordination logic, in: CSL, Vol. 6247 of LNCS,  
1287 Springer, 2010, pp. 305–319.
- 1288 [56] F. Laroussinie, N. Markey, Augmenting ATL with strategy contexts, Information  
1289 and Computation 245 (2015) 98–123.
- 1290 [57] P. Cermák, A. Lomuscio, A. Murano, Verifying and synthesising multi-agent sys-  
1291 tems against one-goal strategy logic specifications, in: AAI, AAAI Press, 2015,  
1292 pp. 2038–2044.
- 1293 [58] P. Cermák, A. Lomuscio, F. Mogavero, A. Murano, Practical verification of  
1294 multi-agent systems against SLK specifications, Information and Computation  
1295 261 (Part) (2018) 588–614.
- 1296 [59] M. Kwiatkowska, D. Parker, C. Wiltsche, Prism-games 2.0: A tool for multi-  
1297 objective strategy synthesis for stochastic games, in: TACAS, Vol. 9636 of LNCS,  
1298 Springer, 2016, pp. 560–566.

- 1299 [60] A. Toumi, J. Gutierrez, M. Wooldridge, A tool for the automated verification of  
1300 Nash equilibria in concurrent games, in: ICTAC, Vol. 9399 of LNCS, Springer,  
1301 2015, pp. 583–594.
- 1302 [61] A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, Uppaal SMC  
1303 tutorial, STTT 17 (4) (2015) 397–415.
- 1304 [62] M. Kwiatkowska, G. Norman, D. Parker, G. Santos, Equilibria-based probabilistic  
1305 model checking for concurrent stochastic games, in: FM, Vol. 11800 of LNCS,  
1306 Springer, 2019, pp. 298–315.
- 1307 [63] A. David, P. G. Jensen, K. G. Larsen, M. Mikucionis, J. H. Taankvist, Uppaal strat-  
1308 ego, in: TACAS, Vol. 9035 of LNCS, Springer, 2015, pp. 206–211.
- 1309 [64] P. E. Bulychev, A. David, K. G. Larsen, A. Legay, M. Mikucionis, Computing Nash  
1310 equilibrium in wireless ad hoc networks: A simulation-based approach, in: Pro-  
1311 ceedings Second International Workshop on Interactions, Games and Protocols,  
1312 IWIGP, Vol. 78 of EPTCS, 2012, pp. 1–14.
- 1313 [65] R. Milner, A Calculus of Communicating Systems, Vol. 92 of LNCS, Springer,  
1314 1980.
- 1315 [66] R. De Nicola, F. W. Vaandrager, Three logics for branching bisimulation, Journal  
1316 of the ACM 42 (2) (1995) 458–487.
- 1317 [67] R. J. van Glabbeek, W. P. Weijland, Branching time and abstraction in bisimula-  
1318 tion semantics, Journal of the ACM 43 (3) (1996) 555–600.
- 1319 [68] R. Alur, T. A. Henzinger, O. Kupferman, M. Y. Vardi, Alternating refinement re-  
1320 lations, in: CONCUR, Vol. 1466 of LNCS, Springer, 1998, pp. 163–178.
- 1321 [69] J. van Benthem, Extensive games as process models, Journal of Logic, Language  
1322 and Information 11 (3) (2002) 289–313.
- 1323 [70] O. Friedmann, M. Lange, Solving parity games in practice, in: ATVA, Vol. 5799  
1324 of LNCS, Springer, 2009, pp. 182–196.

- 1325 [71] M. Ummels, D. Wojtczak, The complexity of Nash equilibria in stochastic multi-  
1326 player games, *Logical Methods in Computer Science* 7 (3) (2011) 1–45.
- 1327 [72] T. Ågotnes, W. van der Hoek, M. Wooldridge, Reasoning about coalitional games,  
1328 *Artificial Intelligence* 173 (1) (2009) 45–79.
- 1329 [73] F. Belardinelli, A. Lomuscio, A. Murano, S. Rubin, Verification of multi-agent  
1330 systems with imperfect information and public actions, in: *AAMAS, ACM*, 2017,  
1331 pp. 1268–1276.
- 1332 [74] B. Aminof, F. Mogavero, A. Murano, Synthesis of hierarchical systems, *Science*  
1333 *of Computer Programming* 83 (2014) 56–79.
- 1334 [75] R. Berthon, B. Maubert, A. Murano, Decidability results for  $ATL^*$  with imperfect  
1335 information and perfect recall, in: *AAMAS, ACM*, 2017, pp. 1250–1258.
- 1336 [76] A. Herzig, E. Lorini, F. Maffre, F. Schwarzenruber, Epistemic Boolean games  
1337 based on a logic of visibility and control, in: *IJCAI, IJCAI/AAAI Press*, 2016, pp.  
1338 1116–1122.
- 1339 [77] F. Belardinelli, A. Lomuscio, Quantified epistemic logics for reasoning about  
1340 knowledge in multi-agent systems, *Artificial Intelligence* 173 (9–10) (2009) 982–  
1341 1013.
- 1342 [78] J. Gutierrez, A. Murano, G. Perelli, S. Rubin, M. J. Wooldridge, Nash equilibria in  
1343 concurrent games with lexicographic preferences, in: *IJCAI, ijcai.org*, 2017, pp.  
1344 1067–1073.