View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Archivio della ricerca- Università di Roma La Sapienza

Enforcing Equilibria in Multi-Agent Systems

Giuseppe Perelli
University of Leicester
giuseppe.perelli@leicester.ac.uk

ABSTRACT

We introduce and investigate *Normative Synthesis*: a new class of problems for the equilibrium verification that counters the absence of equilibria by purposely constraining multi-agent systems. We show that norms are powerful enough to ensure a positive answer to every instance of the equilibrium verification problem. Subsequently, we focus on two optimization versions, that aim at providing a solution in compliance with implementation costs. We show that the complexities of our procedures range between 2EXPTIME and 3EXPTIME, thus that the problems are no harder than the corresponding equilibrium verification ones.

KEYWORDS

Logics for Agents and Multi-Agent Systems; Synthesis of Agent-Based Systems; Verification Techniques for Multiagent Systems, including Model Checking

ACM Reference Format:
Giuseppe Perelli. 2019. Enforcing Equilibria in Multi-Agent Systems. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019*, IFAAMAS, 9 pages.

1 INTRODUCTION

Multi-Agent System Verification [50] has become one of the most important topics in the areas of both *Formal Methods* and *Artificial Intelligence* [51]. The general framework is intended to be a system constituted by more than one *component*, such as software, robots, or entities of any kind. Every agent is assumed to behave *rationally*, meaning that their action is strived to maximize a *payoff function*, which is assigned to it and depends on the outcome of the execution. This means that agents perform according to *strategies*: high-level plans that select the best actions according to the evolution and current knowledge in the system. From the formal point of view, a Multi-Agent System (MAS) is generally modelled by a composition of agents, each of them provided with a specification of its strategic ability and a temporal objective, whose interaction produces an outcome, generally understood as an infinite sequence of variable assignments – on top of which the temporal objectives are interpreted.

Such strategic interaction setting can be profitably analysed as a *game*, on which the classic game-theoretic questions are investigated. Recently, the formal methods and AI communities have shown interests to the question whether there exist (and how to *synthesise*) *equilibria* in such games [22, 24]. This gave rise to the notions of *equilibrium verification* and *rational synthesis*, that have become of particular importance. Equilibrium verification concerns the problem of checking whether some (E-NASH) or all (A-NASH) the Nash equilibria in the game are compatible with a temporal specification that is desired from the designers perspective. In rational verification, instead, the designers actively take part in the equilibrium formation process by directly controlling a system agent, aiming for an equilibrium among the remaining agents (usually denoted as environmental) that is compatible with the aforementioned desired temporal specification.

The setting can be instantiated in a variety of ways, each of them addressing different real-world scenarios, like *automated warehouses*, *self-driving cars*, *computer networks*, and the like, that are of interest to many research communities. For example, *Linear-Temporal Logic* (LTL [48]) is recognized as the canonical temporal goal specification language. Regarding the agents capability description, *Simple Reactive Module* [6] is being widely adopted both for theoretical [25] and practical purposes [7, 41].

So far, much work has been devoted to devise, improve, and implement theory and tools for the *synthesis of equilibria in MAS*. The problem has been addressed under a variety of contexts, by restricting or extending the underlying system structure [9, 24, 25], the information available to the agents [10, 13, 15, 30], their strategic ability [11, 34], as well as objective specification language [14, 26, 29]. In addition to this, logics for the strategic reasoning have been introduced and studied [5, 12, 16, 17, 43, 45, 46].

However, such equilibria are not always guaranteed to exist [4, Thm 1] and all the literature mentioned above on verification and synthesis of MAS is not capable of handling systems without equilibria, meaning that no efficient behaviour can be correctly synthesized for the agents in absence of an equilibrium. This means that many real-world scenarios resolving in MAS with no equilibria cannot benefit from the standard (theoretical and practical) tools developed for the analysis of equilibrium verification. In this paper, we address this case for the first time by introducing a new class of equilibrium verification problems, namely *Normative Synthesis*. We aim at implementing norms [1, 2, 32] into reactive modules games to manipulate agents' behaviours, in order to, either generate an equilibrium encompassing a the desired global behaviour of the system (enforcing E-NASH) or rule out those equilibria that are not compatible with such behaviour (enforcing A-NASH).

We prove that normative synthesis is a robust mechanism, that is, all the admissible instances can be enforced with a compatible equilibrium. This poses an optimization problem, that is solved assuming implementation costs of two kind: single cost, applied only once a norm restriction is employed; iterated cost, applied every time such restriction occurs.

All the results are provided with their computational complexity analysis. We prove that the E-NASH problems are 2EXPTIME-COMPLETE, while the A-NASH ones can be solved in 3EXPTIME. Due to the page limitations, some of the proof are omitted.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada. © 2019 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

2 FORMAL FRAMEWORK

Linear-Temporal Logic (LTL [48]) extends 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 of variables Φ as follows:

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

where $p \in \Phi$. We interpret formulae of LTL with respect to pairs (π, t), where $\pi \in (2^{\Phi})^{\omega}$ is an infinite path of evaluations and $t \in \mathbb{N}$ is a temporal index into π. The semantics of LTL formulae is as follows:

- $\pi, t \models p$ if $p \in \pi[t]$;
- $\pi, t \models \neg\varphi$ if it is not the case that $\pi, t \models \varphi$;
- $\pi, t \models \varphi_1 \vee \varphi_2$ if either $\pi, t \models \varphi_1$ or $\pi, t \models \varphi_2$;
- $\pi, t \models \mathsf{X}\varphi$ if $\pi, t+1 \models \varphi$;
- $\pi, t \models \varphi_1 \mathsf{U} \varphi_2$ if $\pi, t' \models \varphi_2$ for some $t' \geq t$ and $\pi, t'' \models \varphi_1$ for every $t \leq t'' < t'$.

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

A *Mealy* machine [44] is a tuple $M = \langle Q, q^0, I, O, \delta, \tau \rangle$ where: Q is a finite set of internal states; q^0 is the initial internal state; I is the input alphabet; O is the output alphabet; $\delta : Q \times I \to Q$ is the transition function; $\tau : Q \times I \to O$ is the output function.

For a given sequence of input symbols $\iota \in I^{\omega}$, there is a unique sequence of internal states $\rho \in Q^{\omega}$ such that $\rho_0 = q^0$ and, $\rho_{k+1} = \delta(\rho_k, \iota_k)$ for all $k \in \mathbb{N}$. Moreover, this run ρ, together with the input sequence ι, induce a unique sequence of output symbols $o \in O^{\omega}$ defined by $o_k = \tau(\rho_k, \iota_k)$, for all $k \in \mathbb{N}$, sometimes denoted $o = \tau(\rho, \iota)$.

Simple Reactive Modules [49] is a model specification language that is based on Reactive Modules [6] and has been used to describe multi-player games with LTL goals [25, 28]. In a Reactive Modules Game (RMG) one can specify constraints on the power that a player has over the variables it controls. In addition, one can specify multi-player games directly in a high-level description language (which can then be used as the input of a verification tool – Reactive Modules are used, *e.g.*, in MOCHA [7] and PRISM [41]), which is more convenient from a user point of view for modelling purposes.

The core elements of a reactive module are *guarded commands*, that are expressions over the set of Boolean variables Φ of the form φ `->` $x_1' := \psi_1; \cdots; x_k' := \psi_k$ where φ (the guard) is a propositional logic formula over Φ, each x_i is a controlled variable, and each ψ_i is a propositional logic formula over Φ. For a guarded command g, guard(g) and evl(g) denote the guard and the set of assignments of g, respectively. Thus, in the above rule, guard$(g) = \varphi$ and evl$(g) = x_1 := \psi_1; \cdots; x_k := \psi_k$ If no guarded command of a module is enabled, the values of all controlled variables, are left unchanged; the symbol `skip` will refer to the evl part of such command.

Formally, a reactive module, m, is defined as a triple $m = \langle \Phi_m, I_m, U_m \rangle$, where: $\Phi_m \subseteq \Phi$ is the (finite) set of variables controlled by m; I_m is a (finite) set of *initialisation* guarded commands, such that for all $g \in I_m$, we have $ctr(g) \subseteq \Phi_m$; and U_m is a (finite) set of *update* guarded commands, such that for all $g \in U_m$, we have $ctr(g) \subseteq \Phi_m$.

For every module $m = (\Phi_m, I_m, U_m)$, by $G_m = I_m \cup U_m$ we denote the set of its guarded commands, either initialisation or update. Whenever a module is of the indexed form m_i, we replace the indexing of its elements by simply reusing the index i. For example, the set of variables Φ_{m_i} is simply denoted by Φ_i.

For a given module $m = (\Phi_m, I_m, U_m)$ and a valuation $v \in 2^{\Phi}$, by $\text{enable}_m(v) = \{g \in G_m : v \models \text{guard}(g)\}$ we denote the set of guards that are enabled in v. A module m is *deterministic* if, for every valuation $v \in 2^{\Phi}$, it holds that $|\text{enable}_m(v)| = 1$, i.e., at every valuation, there is only one possible guarded command that can be executed. For a set M of modules, by $G_M = \bigcup_{m \in M} G_m$ we denote the set of all the guarded commands of some module in M.

Modules can be composed in an intersection manner as follows. For two modules $m_1 = (\Phi_1, I_1, U_1)$ and $m_2 = (\Phi_2, I_2, U_2)$ with $\Phi_1 \cap \Phi_2 = \emptyset$, the product module is $m_1 \otimes m_2 = (\Phi_1 \cup \Phi_2, I_1 \otimes I_2, U_1 \otimes U_2)$ where the $\otimes$-operator over sets of guards G_1 and G_2 takes every two guards $g_1 \in G_1$ and $g_2 \in G_2$ of the form φ_1 `->` $x_1^1 := \psi_1^1; \cdots; x_k^1 := \psi_{k_1}^1$ and φ_2 `->` $x_1^2 := \psi_1^2; \cdots; x_k^2 := \psi_{k_1}^2$, respectively, and returns the guard $g_1 \otimes g_2$ defined as $\varphi_1 \wedge \varphi_2$ `->` $x_1^1 := \psi_1^1; \cdots; x_k^1 := \psi_{k_1}^1; x_1^2 := \psi_1^2; \cdots; x_k^2 := \psi_{k_1}^2$.

As an example, consider the modules m_y and COPYCAT_x described in below.

```
module m_y controls {y}
  init
  [ ]⊤ -> y' := ⊤;
  [ ]⊤ -> y' := ⊥;
  update
  [ ]⊤ -> y' := ⊥;
  [ ]⊤ -> y' := ⊤;
```

```
module COPYCAT_x controls {x}
  init
  [ ]⊤ -> x' := ⊥
  update
  [ ]⊤ -> x' := y;
```

The module m_y does not put any constraint to the evaluation of variable y. Module COPYCAT_x sets the value of x to false on the first round, then it sets x to the same value of the variable y in the previous step. Observe that COPYCAT_x is deterministic and so there exists only one possible behaviour (later referred as strategy) for it.

The interaction of the two modules produces an execution in which, except for the first round, the value of x is always the same as the one of y taken in the previous iteration.

A *Reactive Module game* (RMG) is a tuple of the form $\mathcal{G} = \langle \text{Ag}, \Phi, m_0, \ldots, m_{|M|}, \gamma_0, \ldots, \gamma_n \rangle$ where $\text{Ag} = \{0, \ldots, n\}$ is a set of agents, Φ is a set of Boolean variables, $m_0, \ldots m_{|M|}$ is a list of modules such that $\Phi_0, \ldots \Phi_{|M|}$ forms a partition of Φ (so every variable in Φ is controlled by some module, and no variable is controlled by more than one module), $n < |M|$, module m_i is associated to agent i, and every module m_h with $h > n$ is deterministic. Finally γ_i is an LTL formula associated to agent i.

RMGs can be seen as an extension of iBGs [24] in which the strategic power of agents is not a-priori fixed but dynamically allocated according to the evolution of the game. Reactive modules can be used to enforce/prevent agents to adopt a desired/undesired behaviour.

The game is played over an infinite number of iterations as follows. At the beginning, all the variables are assumed to be set up to an initial evaluation hereafter understood as the empty evaluation $\emptyset \in 2^{\Phi}$ unless otherwise specified. Then, on the first step, every player i selects an initialisation guarded command $g_i^0 \in I_i$, this producing a transition from $\emptyset$ to the unique evaluation of variables v_0 that follows from executing the commands g_i's. From this point onward, at every step k of the execution, every agent selects an

update guarded command $g_i \in U_i$, which induces a transition from v_k to v_{k+1}.

For a given agent i, $\text{exec}_i : G_i \times 2^\Phi \to 2^{\Phi_i}$ is the function that determines the value of the Boolean variables at the right-hand side of a guarded command when such a guarded command is enabled by a valuation. Formally, exec_i is defined, for a guarded command $g = \varphi \text{ -> } x'_1 := \psi_1; \cdots ; x'_k := \psi_k$ and a valuation v such that $v \models \varphi$, as $\text{exec}_i(g, v) = (v \cup \{x_i : v \models \psi_i\}) \setminus \{x_i : v\neg \models \psi_i\}$.

Example 2.1 (File sharing network). In a file sharing network, two agents are exchanging data through the upload and download of bits. Players 1 and 2 control a variables y_1 and y_2, respectively, denoting that a bit of information from their local documents has been uploaded to the server. They are left free to upload their file in any moment of the execution, therefore they are paired with the modules m_{y_1} and m_{y_2}, respectively. The download phase, instead, is automatically managed in the system by means of the modules for COPYCAT_{x_1} and COPYCAT_{x_2} on variables x_1 and x_2, respectively. This makes the download to happen with a fixed one-step delay with respect to the upload.

Agent i in the network is interested in downloading the local documents of the other one infinitely often. This can be encoded into the LTL formula $\gamma_i = \mathsf{GF}x_{1-i}$, for $i = 1, 2$.

A *strategy* for player i, associated to a module $m_i = (\Phi_i, I_i, U_i)$ is a Mealy machine $\sigma_i = (S_i, s_i^0, 2^\Phi, G_i, \delta_i, \tau_i)$, with 2^Φ and G_i being the input and output alphabets, respectively, such that, for all $s \in S_i$ and $v \in 2^\Phi$, it holds that:

- $\delta_i(s, v) \neq s_i^0$, i.e., the strategy never goes back to the initial state;
- $\delta_i(s, v) \in I_i$ iff $s = s_i^0$, i.e., the strategy selects an initialisation command only when being on the initial state, and an update guarded command, otherwise;
- $\tau_i(s, v) \in \text{enable}_{m_i}(v)$, i.e., the selected guarded command must be enabled in the current valuation.

By Σ_i we denote the set of possible strategies for player i.

A *strategy profile* is a tuple $\vec{\sigma} = (\sigma_0, \ldots, \sigma_n)$ of strategies, one for each player. We also consider partial strategy profiles. For a given set of players $A \subseteq \text{Ag}$, we use the notation σ_A to denote a tuple of strategies, one for each player in A. Moreover, we use the notation σ_{-A} to denote a tuple of strategies, one for each player in $\text{Ag} \setminus A$. We also use σ_i instead of $\sigma_{\{i\}}$ and $\vec{\sigma}_{-i}$ instead of $\vec{\sigma}_{\text{Ag}\setminus\{i\}}$. For two partial strategy profiles $\vec{\sigma}_A$ and $\vec{\sigma}_B$, where $A \cap B = \emptyset$, by $(\vec{\sigma}_A, \vec{\sigma}_B)$ we denote the strategy profile obtained from associating the strategies in $\vec{\sigma}_A$ and $\vec{\sigma}_B$ to players in A and B, respectively.

Since Mealy machines are deterministic, each profile $\vec{\sigma}$ generates a unique play, denoted $\pi(\vec{\sigma})$, which consists of an infinite sequence of valuations, one for each round of the game. Moreover, as it has been observed in [23], such executions are *ultimately periodic*, i.e., of the form $p \cdot t^\omega$, with $p, t \in (2^\Phi)^*$. In this paper, we focus only on executions that are generated by strategies profiles of this form. Therefore, from now on, we will refer to the ultimately periodic executions simply as executions.

To better understand strategies, consider the file-sharing case described in Example 2.1. A very simple strategy for Player i would be given by a Mealy machine σ_i^{on} with a single internal state q_i^0 such that $\tau_i(q_i^0, v) = \{y_i\}$ for every evaluation v, that sends the upload signal at every iteration. Contrarily, the strategy σ_i^{off} with $\tau_i(q_i^0, v) = \emptyset$ makes the player adopting it to never upload. As another example, the strategy σ_i^{copy} with $\tau_i(q_i^0, v) = \{y_i\}$ iff $y_{1-i} \in v$ sends the upload signal only if the other agents has sent the upload signal as well in the previous iteration.

Each player i has a preference relation over plays $\pi \in (2^\Phi)^\omega$, which is determined by its goal γ_i. We say that π is preferred over π' by agent i, and write $\pi \succeq_i \pi'$, if and only if $\pi' \models \gamma_i$ implies that $\pi \models \gamma_i$. Using this notion of preference, one can introduce the concept of *Nash Equilibrium*. We say that $\vec{\sigma}$ is a Nash Equilibrium strategy profile if, for each agent i and a strategy $\sigma'_i \in \Sigma_i$, it holds that $\pi(\vec{\sigma}) \succeq_i \pi(\vec{\sigma}_{-i}, \sigma'_i)$. In addition, by $\text{NE}(\mathcal{G}) \subseteq \Sigma_0 \times \ldots \times \Sigma_n$ we denote the set of Nash Equilibria of the game $\mathcal{G}$ and by $\text{NE-SAT}(\mathcal{G}, \varphi)$ we denote the set of Nash Equilibria $\vec{\sigma}$ that satisfy φ, that is, such that $\pi(\vec{\sigma}) \models \varphi$. Moreover, for a strategy σ_0 for player 0, we say that σ_{-0} is a σ_0-fixed Nash Equilibrium if, for every agent $i \neq 0$ and strategy σ'_i, $\pi(\sigma_0, \vec{\sigma_{-0}}) \succeq_i \pi(\sigma_0, \vec{\sigma}_{-i}, \sigma'_i)$. By $\text{NE}^{\sigma_0}(\mathcal{G})$ we denote the set of σ_0-fixed Nash Equilibria in $\mathcal{G}$. By $\text{NE-SAT}_{\sigma_0}(\mathcal{G}, \varphi)$ we denote the set of σ_0-fixed Nash Equilibria that satisfy φ.

For example, in the file-sharing case depicted above, the profiles $(\sigma_1^{\text{on}}, \sigma_2^{\text{on}})$ and $(\sigma_1^{\text{off}}, \sigma_2^{\text{off}})$ are Nash equilibria, the first satisfies both agents' goals, the second satisfies none of them. On the other hand, the strategy profile $(\sigma_1^{\text{off}}, \sigma_2^{\text{copy}})$ is not a Nash Equilibrium, as agent 1 can deviate to σ_1^{on} to get his goal achieved.

A number of questions related to the *equilibrium analysis* of logic-based multi-player games have been investigated in the literature [25, 39, 51]. Here, we recall the Non-Emptiness, E-Nash and A-Nash problems.

Definition 2.2 (Equilibrium checking). For a given game $\mathcal{G}$, the Non-Emptiness problem is to establish whether $\text{NE}(\mathcal{G}) \neq \emptyset$. For a given LTL formula φ, the E-Nash problem is to establish whether $\text{NE-SAT}(\mathcal{G}, \varphi) \neq \emptyset$. Moreover, the A-Nash problem is to establish whether $\text{NE-SAT}(\mathcal{G}, \varphi) = \text{NE}(\mathcal{G})$.

In addition to this, we make use of the *Rational Synthesis* problems, defined as follows.

Definition 2.3 (Rational Synthesis). For a given game $\mathcal{G}$, the *weak Rational Synthesis* problem is to establish whether there exists a strategy σ_0 for Player 0 such that $\text{NE}^{\sigma_0}(\mathcal{G}, \gamma_0) \neq \emptyset$. Moreover, the *strong Rational Synthesis* problem is to establish whether there exists a strategy σ_0 for Player 0 such that $\text{NE}^{\sigma_0}(\mathcal{G}, \gamma_0) = \text{NE}^{\sigma_0}(\mathcal{G})$.

In the following, w.l.o.g., we sometimes deal with games on a number of agents ranging from 1 to n instead of 0 to n. In particular, we do this on games for which we investigate on the Non-Emptiness, E-Nash, and A-Nash problems.

Regarding Example 2.1, note that the Non-Emptiness problem has a positive answer, as the game admits a Nash Equilibrium. Now, assume we are interested in the global property $\varphi = \mathsf{G}\neg(x_1 \wedge x_2)$ which prevents the system to be overloaded by downloading the local documents at the same time. In this case, the E-Nash problem also has a positive answer. Indeed, consider the strategy $\sigma_1^{-\text{copy}}$ that behaves like σ_1^{copy} except for the fact that it starts by sending the signal of not uploading and then toggles indefinitely. Clearly the strategy profile $(\sigma_1^{-\text{copy}}, \sigma_2^{\text{copy}})$ satisfies φ and is also a Nash Equilibrium, as it also satisfies both agents goals. On the other hand

the A-Nash problem has a negative answer, as the Nash Equilibrium $(\sigma_1^{\text{on}}, \sigma_2^{\text{on}})$ does not satisfy φ.

3 NORMATIVE SYNTHESIS

For a given game $\mathcal{G}$ with M being the set of modules, and $D \subseteq G_M$ a set of *deactivating* commands, a (*dynamic*) D-*norm* is a Mealy machine $\mathcal{N}^D = \langle Q, q_0, 2^\Phi, 2^D, \delta, \eta \rangle$ with 2^Φ and 2^D being the input and output alphabets, respectively. For a D-norm, the output function is sometimes called *normative function*.

When it is clear from the context, we avoid using the symbol D and call a D-norm simply norm, and denote it as $\mathcal{N}$.

Intuitively, a norm $\mathcal{N}$ over $\mathcal{G}$ restricts the strategic power of the agents by preventing them to use some of the guarded commands from their modules.

For a given game $\mathcal{G}$ and a D-norm $\mathcal{N}^D$ over it, the *Normative Game*, denoted $G \dagger \mathcal{N}^D$, is a game in which the subset D of guarded commands for the agents can be activated/deactivated along the execution, meaning that agents can execute them only when they are left active by the norm. In this case, the execution is an infinite sequence $\pi \in (2^\Phi)^\omega$ of evaluation of the variables on the game paired with an infinite sequence $\zeta \in (2^D)^\omega$ of the subset of guarded commands that are currently deactivated by the norm. For this reason, the agents strategies must comply with the norm.

A *norm-compliant strategy* for agent i is a Mealy machine of the form $\sigma_i = (S_i, s_i^0, 2^\Phi \times 2^{G_M}, G_i, \delta_i, \tau_i)$ such that, for all $s \in S_i$ and $(v, C) \in 2^\Phi \times 2^{G_M}$, it holds that $\tau_i(s, (v, C)) \in G_i \setminus C$. By Σ_i^{norm} we denote the set of norm-compliant strategies for agent i.

In the case of a game with an implemented norm $\mathcal{G} \dagger \mathcal{N}$, the execution is generated not only by a norm-compliant strategy profile $\vec{\sigma}$, but also by the norm itself. We denote by $\pi(\mathcal{N}, \vec{\sigma}) \in (2^\Phi)^\omega$ such execution and $\zeta(\mathcal{N}, \vec{\sigma}) \in (2^D)^\omega$ the infinite sequence of the set of commands that are deactivated at every iteration.

Accordingly, a Nash Equilibrium in $\mathcal{G} \dagger \mathcal{N}$ is a total strategy profile $\vec{\sigma} \in \Sigma_1^{\text{norm}} \times \ldots \times \Sigma_n^{\text{norm}}$ such that, for every agent i and a norm-compliant strategy $\sigma_i' \in \Sigma_i^{\text{norm}}$, it holds that $\pi(\mathcal{N}, \vec{\sigma}) \succeq_i \pi(\mathcal{N}, \vec{\sigma}_{-i}, \sigma_i')$. In addition, by $\text{NE}(\mathcal{G} \dagger \mathcal{N}) \subseteq \Sigma_1^{\text{norm}} \times \ldots \times \Sigma_n^{\text{norm}}$ we denote the set of Nash Equilibria of the game $\mathcal{G} \dagger \mathcal{N}$. Equivalently, NE-Sat$(\mathcal{G} \dagger \mathcal{N}, \varphi)$ denotes the set of Nash Equilibria in $\mathcal{G} \dagger \mathcal{N}$ that satisfy φ.

We are now ready to state the *Norm Synthesis* problems.

Definition 3.1 (Norm Synthesis). For a given game $\mathcal{G}$, a set of deactivating commands D, and an LTL formula φ:

Norm-Synthesis Non-Emptiness (NSNE): is there a D-norm $\mathcal{N}$ such that $\text{NE}(\mathcal{G} \dagger \mathcal{N}) \neq \emptyset$?

Norm-Synthesis E-Nash (NSE-Nash): is there a norm $\mathcal{N}$ such that NE-Sat$(\mathcal{G} \dagger \mathcal{N}, \varphi) \neq \emptyset$?

Norm-Synthesis A-Nash (NSA-Nash): is there a norm $\mathcal{N}$ such that NE-Sat$(\mathcal{G} \dagger \mathcal{N}) = \text{NE}(\mathcal{G} \dagger \mathcal{N})$?

Before showing how to solve the norm synthesis problems, let us consider again the file-sharing case described in Example 2.1. Recall that the Non-Emptiness and E-Nash problems have a positive answer. Therefore, we do not need any norm to find a solution to these. However, regarding the A-Nash, we need to employ a norm to enforce a solution. For example, consider the norm $\mathcal{N}$ that deactivates the guarded command `[ ]⊤ -> y1' := ⊤` on the odd iterations and the guarded command `[ ]⊤ -> y2' := ⊤` on the even ones. This clearly makes every execution to satisfy the formula $\varphi = \mathsf{G}\neg(x_1 \wedge x_2)$. In particular, every Nash Equilibrium in $\mathcal{G} \dagger \mathcal{N}$ will satisfy φ, therefore the NSA-Nash has a positive answer.

Before solving the general norm synthesis problems, we focus on two special cases. Let us consider the $\emptyset$-norm $\mathcal{N}^\emptyset$, hereafter referred as the empty norm, producing an empty set of constraints at every iteration. It is clear that, for every possible strategy profile $\vec{\sigma}$ in $\mathcal{G}$, the empty norm produces the infinite sequence $\zeta = \emptyset^\omega$ of command deactivation, thus not providing any additional constraint to the agent's strategic power. We have the following.

Theorem 3.2. *For a given game $\mathcal{G}$, it holds that NE$(\mathcal{G})$ = NE$(\mathcal{G} \dagger \emptyset)$. Analogously, for a given game $\mathcal{G}$ and an LTL formula φ, it holds that NE-Sat$(\mathcal{G}, \varphi)$ = NE-Sat$(\mathcal{G} \dagger \emptyset, \varphi)$.*

On the other hand, let us consider every command to be deactivating, i.e., $D = G_M$. Then, for every possible execution π, it is not hard to define a norm $\mathcal{N}$ that enforces a single strategy profile $\vec{\sigma}$ and such that $\pi(\vec{\sigma}) = \pi$. We have the following

Lemma 3.3. *For a given game $\mathcal{G}$ and execution π in the game, there exists a G_M-norm $\mathcal{N}$ such that there exists only a norm-compliant strategy profile $\vec{\sigma}$ with $\pi(\vec{\sigma}) = \pi$.*

Proof sketch. Recall that π is ultimately periodic and that, at every step π_k of that and every agent i, there exists a guarded command g_k^i such that $\pi_{k+1} = \cup_{i \in \text{Ag}} \text{exec}_i(g_k^i, \pi_k)$. At this point, it is not hard to define a G_M-norm $\mathcal{N}$ that, at every time-step of the execution, deactivates all the guarded commands but the ones that have to be used to generate the next step of π. Furthermore, by construction, for every agent i, only the strategy σ_i used to generate π is available when $\mathcal{N}$ is implemented and so π is the only possible execution in $\mathcal{G} \dagger \mathcal{N}$. □

Now, for a given LTL formula φ, let us assume that there exists a strategy profile $\vec{\sigma}$ such that $\pi(\vec{\sigma}) \models \varphi$. Then, we can use the norm $\mathcal{N}^{G_M}$ that enforces $\vec{\sigma}$ to be the only possible strategy profile of the game $\mathcal{G} \dagger \mathcal{N}^{G_M}$. Clearly, $\vec{\sigma}$ is a Nash Equilibrium and it satisfies φ. Therefore, we have that NE-Sat$(\mathcal{G} \dagger \mathcal{N}^{G_M})$ = NE$(\mathcal{G} \dagger \mathcal{N}^{G_M}) \neq \emptyset$. This produces the following.

Lemma 3.4. *For a given game $\mathcal{G}$ and an LTL formula φ, the NSE-Nash problem when $D = G_M$ has a solution if and only if φ is realizable over $\mathcal{G}$.*

Thanks to these Lemmas, we have the two following results.

Theorem 3.5. *For every game $\mathcal{G}$, there exists a norm $\mathcal{N}^{G_M}$ such that NE$(\mathcal{G} \dagger \mathcal{N}^{G_M}) \neq \emptyset$.*

Theorem 3.6. *The NSE-Nash and NSA-Nash problems with $D = G_M$ are PSPACE-complete.*

We now focus on the general case of an arbitrary $D \subseteq 2^{G_M}$. We show that the problem can be suitably encoded in rational synthesis.

First, let us recall some useful definitions and notation. For $\pi : \mathbb{N}_0 \to 2^\Phi$ a run, we say that $\pi' : \mathbb{N}_0 \to 2^{\Phi'}$, with $\Phi \subseteq \Phi'$ is a 2-*fold inflation* of π if $\pi'[2t] = \pi[t]$ for every $t \geq 0$. For a set Ψ of propositional variables with $\Phi \subseteq \Psi$, also say that a run $\pi' : \mathbb{N}_0 \to 2^\Psi$ a 2-*fold inflation* of π if $\pi'[2t] \cap \Phi = \pi[t]$ for every $t \geq 0$. Moreover, for $r \in \Psi \setminus \Phi$, we say that a 2-fold inflation π' of π is

r-labelled if for all $t \geq 0$, $r \in \pi'[t]$ if and only if t is even, i.e. there is some $t' \in \mathbb{N}$ with $t = 2t'$. Thus, in a r-labelled, 2-fold inflation π' of π we have that $\pi'[t] \models r$ if and only if t is even.

Clearly, from a run $\pi' : \mathbb{N}_0 \to 2^{\Phi'}$, we can define the 2-fold deflation π over Φ to be the run $\pi : \mathbb{N}_0 \to 2^{\Phi}$ which satisfies that $\pi[t] = \pi'[2t] \cap \Phi$ for every $t \geq 0$. Note that, for a given run π', there is a unique 2-fold deflation π over Φ.

We now define a translation function tr^2 which maps LTL formulae φ over Φ to LTL formulae $\mathrm{tr}^2(\varphi)$ over $\Phi \cup \{r\}$, where $r \notin \Phi$ is a fresh variable.

- $\mathrm{tr}^2(x) = x$, for every $x \in \Phi$;
- $\mathrm{tr}^2(\neg\varphi) = \neg\mathrm{tr}^2(\varphi)$;
- $\mathrm{tr}^2(\varphi T \psi) = \mathrm{tr}^2(\varphi) T \mathrm{tr}^2(\psi)$;
- $\mathrm{tr}^2(\mathsf{X}\varphi) = \mathsf{XX}\mathrm{tr}^2(\varphi)$;
- $\mathrm{tr}^2(\varphi \mathsf{U} \psi) = (r \rightarrow \mathrm{tr}^2(\varphi)) \mathsf{U} (r \wedge \mathrm{tr}^2(\psi))$.

We have the following result.

Lemma 3.7 (Inflation [28]). *Let Φ and Φ' be two disjoint sets of propositional variables with $q \in \Phi'$, $\pi : \mathbb{N}_0 \to 2^{\Phi}$ a run, and $\pi' : \mathbb{N}_0 \to 2^{\Phi \cup \Phi'}$ a q-labelled, 2-fold inflation of π. Then, for all LTL formulae φ over Φ, it holds that $\pi \models \varphi$ if and only if $\pi' \models \mathrm{tr}^2(\varphi)$.*

The idea of the inflation lemma is that we can interleave the game execution with the decision made by the norm on which agents commands to deactivate.

Now, for a given game $\mathcal{G}$ and set D of deactivating commands, we define a game $\mathcal{G}^D_{\text{norm}}$ on which an extra agent, namely the *normative agent*, has the ability of activate/deactivate the other agents guarded commands. The set of strategies of the normative agent will correspond one-to-one to the set of possible D-norm that are applicable to $\mathcal{G}$. Thus, synthesizing a strategy for the normative agent in $\mathcal{G}^D_{\text{norm}}$ will correspond exactly to synthesizing a D-norm in $\mathcal{G}$.

```
module RUN controls {r}
  init
  [ ]⊤ -> r' := ⊥;
  update
  [ ]⊤ -> r' := ¬r;
```

Figure 1: The module RUN

We define $\mathcal{G}^D_{\text{norm}}$ by starting from its modules. First, consider the module in Figure 1. The module RUN starts by setting the value of variable r to false at the beginning, and then it flips its value at each step of its own execution. Note that the module RUN is deterministic. Therefore, it will keep flipping the value of r on any execution whatsoever.

Variable r is used to alternate the two phases of the executions. In the first phase, when r is false, the normative agent selects the set of guarded commands to be disabled for the rest of the agents. In the next phase, when r is true, the agents in the original game $\mathcal{G}$ execute a command that is enabled to update the truth-values of the variables in the game. The resulting outcome is an infinite play that interleaves the two phases, producing the execution of the norm on the odd iterations and the collective execution of the agents on the even iterations. Thanks to the results on 2-fold inflation of Lemma 3.7, we are able to correctly interpret the satisfaction value of the LTL goals in this interleaved execution.

To correctly encode this process, we need to define the modules apparatus of the normative agent. For every deactivating guarded command $g \in D$, define the module m_g as follows.

If g is an initial guard

```
module m_g controls {g}
  init
  [ ]⊤ -> g' := ⊤;
  [ ]⊤ -> g' := ⊥;
  update
  [ ]⊤ -> g' := ⊥;
```

If g is an update guard

```
module m_g controls {g}
  init
  [ ]⊤ -> g' := ⊥;
  update
  [ ]¬r -> g' := ⊤;
  [ ]¬r -> g' := ⊥;
```

Every module m_g controls a single variable g, named after the corresponding command. Intuitively, if g is an initial command, the corresponding module can decide whether to enable/disable the execution of such command at the beginning of the play, and it is forced to keep it disabled for the subsequent steps. On the other hand, if g is an update command, the corresponding module disables it on the first iteration of the play and then has full power on enabling/disabling g for all the odd iterations, i.e., the one labelled with $\neg r$ denoting the norm phases of the execution.

We show later that the product module $m_{\text{norm}} = \bigotimes_{g \in G_M} m_g$ determines exactly the strategic power of norms, that is, the set of norms for $\mathcal{G}$ corresponds to the set of strategies that are compatible with m_{norm}.

Now, we amend every module m_i of the original game $\mathcal{G}$ accounting the fact that it has to comply with possible deactivation of its commands. We define the module m'_i as reported in Figure 2. Since the execution starts with a norm phase, the module associated to the agents in the game is forced to execute the `skip` command at the beginning. At the second iteration, and all the even ones, i.e., on the agents phase, the module is allowed to execute a command g only if the corresponding variable (whose value has been set up by the norm in the previous iteration) is true and also guard(g) is satisfied.

```
module m'_i controls Φ_i
  init
skip
  update
  [ ]r ∧ g^i_1 ∧ guard(g^i_1) -> evl(g^i_1);
...
  [ ]r ∧ g^i_h ∧ guard(g^i_h) -> evl(g^i_h);
```

Figure 2: The module m'_i

The reader shall notice that, given the definition of the guard modules, the initial guards can be enabled only on the second iteration, while the update guards can be enabled only from the fourth iteration onwards. This complies with the fact that, thanks to Lemma 3.7, the original formulas are evaluated on the subsequence of paths that are extracted from even positions. For every guarded command g in a given module m_i, by g' we denote the corresponding guarded command in m'_i. Moreover, G'_i will denote the set of guarded commands in m'_i.

We are now ready to define the encoding game. For a given game $\mathcal{G} = \langle \mathrm{Ag}, \Phi, m_1, \ldots, m_M, \gamma_1, \ldots, \gamma_n \rangle$ and a set D of deactivating guarded commands, the D-normative game is defined as $\mathcal{G}^D_{\text{norm}} = \langle \{\text{norm}\} \cup \mathrm{Ag}, \Phi', m_{\text{norm}}, m'_1, \ldots, m_{|M'|}, \text{RUN}, \gamma_{\text{norm}}, \mathrm{tr}^2(\gamma_1), \ldots, \mathrm{tr}^2(\gamma_n) \rangle$, where $\Phi' = \Phi \cup D \cup \{r\}$ and γ_{norm} is an LTL formula provided for the normative agent. In addition, RUN is included in the list of deterministic modules.

The idea of $\mathcal{G}^D_{\text{norm}}$ is to encode the reasoning about the existence of D-norms for $\mathcal{G}$, in terms of the existence of a strategy in $\mathcal{G}^D_{\text{norm}}$ for the normative agent. In the following, we show how to bridge

D-norms for $\mathcal{G}$ and normative agents strategies, as well as strategies for agents that corresponds in both games.

Regarding norms, consider a function Γ that transforms a D-norm $\mathcal{N} = \langle Q, q_0, 2^{\Phi}, 2^{D}, \delta, \eta\rangle$ into a strategy $\sigma_{\text{norm}} = \Gamma(\mathcal{N}) = \langle Q, q_0, 2^{\Phi'}, 2^{D'}, \delta', \tau'\rangle$ for the normative agent norm in $\mathcal{G}^D_{\text{norm}}$, where $D' = \{g' : g \in D\}$ and, for every state q and valuation v, we have that $\delta'(q, v) = q$ if $r \in v$ and $\delta'(q, v) = \delta(q, v_{\restriction\Phi})$, otherwise, and $\tau'(q, v) = \texttt{skip}$, if $r \in v$, and $\tau'(q, v) = D' \setminus \eta(q, v_{\restriction\Phi})$ otherwise.

Intuitively, the strategy $\Gamma(\mathcal{N})$ emulates the operation of the norm $\mathcal{N}$ by setting to false all and only the variables related to the guards that are deactivated by $\mathcal{N}$ in the corresponding iteration. In addition to this, the strategy is forced to perform a skip operation whenever the turn is assigned to the other agents in $\mathcal{G}^D_{\text{norm}}$.

Conversely, for a strategy $\sigma_{\text{norm}} = \langle Q, q_0, 2^{\Phi'}, 2^{D'}, \delta', \tau'\rangle$ for player norm in $\mathcal{G}^D_{\text{norm}}$, consider the D-norm $\mathcal{N} = \Delta(\sigma_{\text{norm}}) = \langle Q \times 2^{D}, (q_0, \tau'(q_0, \emptyset)), 2^{\Phi}, 2^{D}, \delta, \eta\rangle$ where $\delta((q, c), v) = \delta'(q, v \cup (D \setminus c))$ and $\eta((q, c), v) = \tau'(q, v \cup (D \setminus c))$. In this case, starting from a strategy for the normative agent in $\mathcal{G}^D_{\text{norm}}$, we have defined a D-norm that emulates such strategy by deactivating those guards whose corresponding variable is set to false by the strategy σ_{norm}.

For a norm compliant strategy $\sigma_i = \langle S_i, s_i^0, 2^{\Phi} \times 2^{G_i}, G_i, \delta_i, \tau_i\rangle$ for agent i in the game $\mathcal{G}$, we apply a similar reasoning and define the strategy $\sigma_i' = \Gamma(\sigma_i) = \langle S_i, s_i^0, 2^{\Phi'}, G_i', \delta_i, \tau_i\rangle$ where, for every state $s \in S_i$ and valuation v, $\delta_i'(s, v) = s$ if $r \notin v$ and $\delta_i'(s, v) = \delta_i(s, v_{\restriction\Phi})$, otherwise, and $\tau_i'(s, v) = \texttt{skip}$, if $r \notin v$ and $\tau_i'(s, v) = \tau_i(s, (v_{\restriction\Phi}, v_{\restriction G'}))$, otherwise.

Conversely, for a strategy $\sigma_i' = \langle S_i, s_i^0, 2^{\Phi'}, G_i', \delta_i, \tau_i\rangle$ for player i in $\mathcal{G}^D_{\text{norm}}$, define the norm compliant strategy $\sigma_i = \Delta(\sigma_i') = \langle S_i, s_i^0, 2^{\Phi} \times 2^{G_i}, G_i, \delta_i, \tau_i\rangle$ where $\delta_i(s, (v, c)) = \delta_i'(q, v \cup c \cup \{r\})$ and $\tau_i(s, (v, c)) = \tau_i'(q, v \cup c \cup \{r\})$.

It is not hard to show that, for every norm $\mathcal{N}$ and normative strategy σ_{norm}, it holds that $\Delta(\Gamma(\mathcal{N})) = \mathcal{N}$ and $\Gamma(\Delta(\sigma_{\text{norm}})) = \sigma_{\text{norm}}$. The same holds for every strategy σ_i and σ_i' in $\mathcal{G}$ and $\mathcal{G}^D_{\text{norm}}$, respectively. Therefore, Γ and Δ are one the inverse of the other.

At this point, we have the following result.

Lemma 3.8. *Let $\mathcal{G}$ be a game and $\mathcal{G}^D_{\text{norm}}$ be the corresponding D-normative game for some set D of deactivating guarded commands. Then, the two following hold.*

1. *For every D-norm $\mathcal{N}$ and norm-compliant strategy profile $\vec{\sigma}$ in $\mathcal{G} \dagger \mathcal{N}$, we have that $\pi(\mathcal{N}, \vec{\sigma}) = \pi(\Gamma(\mathcal{N}, \vec{\sigma}))$[2];*
2. *For every strategy profile $\vec{\sigma}' \in \mathcal{G}^D_{\text{norm}}$, $\pi(\vec{\sigma}') = \pi(\Delta(\vec{\sigma}'))$.*

At this point, we can prove the following theorem.

Theorem 3.9. *For a game $\mathcal{G}$, a set of deactivating guarded commands D, and a formula φ, the following hold:*

1. *There exists a D-norm $\mathcal{N}$ such that $\text{NE}(\mathcal{G} \dagger \mathcal{N}) \neq \emptyset$ if, and only if $\text{NE}^{\Gamma(\mathcal{N})}(\mathcal{G}^D_{\text{norm}}, \top) \neq \emptyset$;*
2. *There exists a D-norm $\mathcal{N}$ such that $\text{NE-Sat}(\mathcal{G} \dagger \mathcal{N}, \varphi) \neq \emptyset$ if, and only if $\text{NE-Sat}^{\Gamma(\mathcal{N})}(\mathcal{G}^D_{\text{norm}}, \text{tr}^2(\varphi)) \neq \emptyset$;*
3. *There exists a D-norm $\mathcal{N}$ such that $\text{NE-Sat}(\mathcal{G} \dagger \mathcal{N}, \varphi) = \emptyset$ if, and only if $\text{NE-Sat}^{\Gamma(\mathcal{N})}(\mathcal{G}^D_{\text{norm}}, \varphi) = \text{NE}^{\Gamma(\mathcal{N})}(\mathcal{G}^D_{\text{norm}}, \text{tr}^2(\varphi))$;*

Proof. We show the proof of Item (1) only, as the ones for items (2) and (3) are similar.

Let us assume that there exists a D-norm $\mathcal{N}$ for $\mathcal{G}$ and a strategy profile $\vec{\sigma}$ such that $\vec{\sigma} \in \text{NE}(\mathcal{G} \dagger \mathcal{N})$. Then, it holds that $\Gamma(\mathcal{N}, \vec{\sigma}) \in \text{NE}^{\Gamma(\mathcal{N})}(\mathcal{G}^D_{\text{norm}}, \top)$. First, observe that $\top$ is satisfied no matter the play. So, we have to prove only that $\Gamma(\vec{\sigma})$ is a $\Gamma(\mathcal{N})$-fixed Nash Equilibrium. For an agent i, assume by contradiction that there exists a strategy σ_i' such that $\pi(\Gamma(\mathcal{N}, \vec{\sigma})) \not\models \text{tr}^2(\gamma_i)$ and $\pi((\Gamma(\mathcal{N}, \vec{\sigma}))_{-i}, \sigma_i') \models \text{tr}^2(\gamma_i)$. Thus, by means of Item (2) of Lemma 3.8, it holds that $\pi(\Delta(\Gamma(\mathcal{N}, \vec{\sigma}))) = \pi(\mathcal{N}, \vec{\sigma}) \not\models \gamma_i$ and $\pi(\Delta((\Gamma(\mathcal{N}, \vec{\sigma}))_{-i}), \Delta(\sigma_i')) = \pi(\mathcal{N}, \vec{\sigma})_{-i}, \Delta(\sigma_i')) \models \gamma_i$, and so $\Delta(\sigma_i')$ is a beneficial deviation for agent i from $\vec{\sigma}$, contradicting the fact that the latter is a Nash Equilibrium.

The other direction of the proof is symmetric and we omit it. □

The theorem proved above shows that we can solve the Norm Synthesis problems by means of a reduction to Rational Synthesis instances. In particular, the NSNE and NSE-Nash problems can be solved by means of a weak Rational Synthesis instance, whereas the NSA-Nash can be solved by means of a strong Rational Synthesis instance. Therefore, we have the following.

Corollary 3.10. *The NSNE and NSE-Nash problems are 2exptime-complete. The NSA-Nash problem can be solved in 3exptime.*

4 THE OPTIMIZATION CASE

In the previous section, we provided three results that can be summarized as follows. Theorem 3.2 shows that, in case no deactivating commands are available, the problem of norm synthesis resolves to the classic equilibrium verification. On the opposite side, Theorems 3.5 and 3.6 show that, when given the full power of deactivating any possible command, norms are powerful enough to either an equilibrium or an equilibrium satisfying a desired temporal property, provided the latter is realizable in the system. Finally, Theorem 3.9 solves the normative synthesis problem when the set of deactivating commands is arbitrarily fixed.

In real world scenarios, deactivating a command can come with a cost that is due to implementing the deactivation feature itself. So, from a designer's point of view, the question might be to minimize the number of deactivating commands in order to enforce an equilibrium in the game. Formally, we assume the game $\mathcal{G}$ being equipped with a cost function $\text{c} : G_M \to \mathbb{N}$, assigning a positive integer to every guarded command in the game. With an abuse of notation, by $\text{c}(D) = \sum_{g \in D} \text{c}(g)$ we denote the cost of a subset of commands. A game $\mathcal{G}$ equipped with a cost function c is called *cost game* and denoted by $(\mathcal{G}, \text{c})$. For a cost game $(\mathcal{G}, \text{c})$ we say that $D \subseteq G_M$ is *Non-Emptiness optimal* if there exists a D-norm $\mathcal{N}^D$ such that $\text{NE}(\mathcal{G} \dagger \mathcal{N}^D) \neq \emptyset$ and $\text{NE}(\mathcal{G} \dagger \mathcal{N}^{D'}) = \emptyset$ for every D'-norm $\mathcal{N}^{D'}$, such that $\text{c}(D') < \text{c}(D)$. Analogously, for a cost game $(\mathcal{G}, \text{c})$ and an LTL formula φ, we say that $D \subseteq G_M$ is *E-Nash optimal* with respect to φ if there exists a D-norm $\mathcal{N}^D$ such that $\text{NE-Sat}(\mathcal{G} \dagger \mathcal{N}^D, \varphi) \neq \emptyset$ and $\text{NE-Sat}(\mathcal{G} \dagger \mathcal{N}^{D'}, \varphi) = \emptyset$ for every D'-norm $\mathcal{N}^{D'}$, with D' such that $\text{c}(D') < \text{c}(D)$. Finally, we say that D is *A-Nash optimal* with respect to φ if there exists a D-norm $\mathcal{N}^D$ such that $\text{NE-Sat}(\mathcal{G} \dagger \mathcal{N}^D, \varphi) = \text{NE}(\mathcal{G} \dagger \mathcal{N}^D, \varphi)$ and $\text{NE-Sat}(\mathcal{G} \dagger \mathcal{N}^{D'}, \varphi) \neq \text{NE}(\mathcal{G} \dagger \mathcal{N}^{D'}, \varphi)$ for every D'-norm $\mathcal{N}^{D'}$, with D' such that $\text{c}(D') < \text{c}(D)$.
We can now define a new class of problems.

Definition 4.1 (Optimal Norm Synthesis). For a cost game $(\mathcal{G}, \text{c})$ and an LTL formula φ:

NSNE-Optimal: compute a set of deactivating guarded commands that is Non-emptiness optimal.
NSE-Nash-Optimal: compute a set of deactivating guarded commands that is E-Nash optimal with respect to φ.
NSA-Nash-Optimal: compute a set of deactivating guarded commands that is A-Nash optimal with respect to φ.

It is not hard to say that the complexity of solving these problems is the same as the corresponding decision versions analysed in the previous section.

Theorem 4.2. *The NSNE-Optimal and NSE-Nash-Optimal problems are 2exptime-complete. Moreover, the problem NSA-Nash-Optimal can be solved in 3exptime.*

Proof sketch. We show only the NSNE-Optimal problem, as the others are similar. First, we order the set of deactivating subsets according to their costs, that is, $D \leq_c D'$ if and only if $c(D) \leq c(D')$. At this point, starting from the less costly set D, we run the NSNE problem with D being the deactivating set. If the answer is negative, we proceed to the next set D' in terms of costs. If the answer is positive, we return D and the relative synthesized norm $\mathcal{N}^D$ as a solution. Note that, in order to return D, every less costly set D' must have been already analysed and labelled as unsuitable. Therefore, D is necessarily a minimal cost deactivating set. Moreover, note that Theorem 3.5 guarantees that at least one set D and a norm $\mathcal{N}^D$ will be returned.

To prove the lower bound, we reduce the Non-Emptiness problem introduced in [25]. For a game $\mathcal{G}$, set up $c(g) = 1$ for every guarded command in G_M. Then, trivially, $\mathcal{G}$ has a Nash Equilibrium if, and only if, the optimal deactivating set D is the empty set $\emptyset$. □

5 ENERGY NORMS

In this section we assume a norm to be always a G_M-norm, that is, a norm that is capable of deactivating any guarded command in the related game $\mathcal{G}$.

Let us now assume that deactivating a guarded command comes with a cost in terms of electric energy, and let the energy function $\mathrm{e} : G_M \to \mathbb{N}$ mapping every guard to the energy cost required to deactivate it.

A game $\mathcal{G}$ paired with an energy function e is called *energy game* and denoted $(\mathcal{G}, \mathrm{e})$. In order to function in an energy game, a norm is powered by an electric generator that produces c units of energy at every iteration, whose storage capacity is a certain value c_{max}. In an energy game, every step of the execution comes with an energy reserve that can be used by the norm to deactivate commands. The game starts at with an empty level of energy and, at every iteration k in the game, the energy reserve level l_k is updated by adding c units of energy and deducting the costs of deactivating the norms in the previous step, all capped at c_{max}. More formally, starting from an energy level l, the next one l' is given by $l' = \min\{c_{max}, l + c - (\sum_{g \in \eta(q,v)} \mathrm{e}(g))\}$, where q and v are the normative state and evaluation used to determine the set of deactivated guarded commands.

For a given energy game $(\mathcal{G}, \mathrm{e})$ and a maximum capacity c_{max}, an *energy norm* is a Mealy machine of the form $\mathcal{N} = \langle Q, q_0, 2^{\Phi} \times \{0, \ldots, c_{max}\}, 2^{G_M}, \delta, \eta \rangle$ such that, for all $q \in Q$ and $(v, c) \in 2^{\Phi} \times \{0, \ldots, c_{max}\}$, it holds that $\mathrm{e}(\eta(q, (v, c))) < c$. That is, the energy required to deactivate all the guarded commands in $\eta(q, (v, c))$ is strictly less than the current energy level c.

Observe that the implementation of an energy norm over an energy game always produces a sequence $\mathbf{l} = l_0, l_1, \ldots$, starting from $l_0 = 0$, such that $0 < l_k \leq c_{max}$ for all $k \in \mathbb{N}$, no matter which strategy profile $\vec{\sigma}$ is executed by the agents in $\mathcal{G}$.

We are now ready to define a set of energy game problems.

Definition 5.1 (Energy Norm Synthesis). For a given energy game $(\mathcal{G}, \mathrm{e})$ and an LTL formula:

Energy Norm-Synthesis Non-Emptiness (ENSNE): is there any energy norm $\mathcal{N}$ such that $\mathrm{NE}(\mathcal{G} \dagger \mathcal{N}) \neq \emptyset$?
Energy Norm-Synthesis E-Nash (ENSE-Nash): is there any energy norm $\mathcal{N}$ such that $\text{NE-Sat}(\mathcal{G} \dagger \mathcal{N}) \neq \emptyset$?
Energy Norm-Synthesis A-Nash (ENSA-Nash): is there any energy norm $\mathcal{N}$ such that $\text{NE-Sat}(\mathcal{G} \dagger \mathcal{N}) = \mathrm{NE}(\mathcal{G} \dagger \mathcal{N})$?

We now show the solution of the energy game problems. Similarly to the case of Section 3, we do this by exploiting the versatility of srml, that allow us to inject the energy reasoning into suitably defined modules.

First, let us consider a set of k boolean variables $d_0, \ldots, d_{k-1}$. By using the binary representation of natural numbers, we can pair one-to-one every truth assignment v_d of variables $d_0, \ldots, d_{k-1}$ with a number between 0 and $2^k - 1$, which is given by $\mathrm{num}(v_d) = \sum_{0 \leq j \leq k-1} 2^j \cdot v_d(d_j)$ [1]. Conversely, for every number $0 \leq c \leq 2^{k-1}$, by $\mathrm{bin}(c)$ we denote the unique truth assignment of variables $d_0, \ldots, d_{k-1}$ such that $c = \mathrm{num}(\mathrm{bin}(c))$.

The deterministic module represented in Figure 3 is in charge of generating the energy level sequence during the execution of the game. It does it by waiting for which guarded commands the norm deactivates (when the variable r is false) and then, in the next iteration, updating the digits representing the energy level accordingly. Notice that the update section in the definition of the module is a list of guarded commands, one per each evaluation of the form $\chi_{G'} = \bigwedge_{g' \in G'} g' \wedge \bigwedge_{g' \notin G'} \neg g'$, that updates the energy level by following $l' := l + c - \sum_{g \in \chi_{G'}} \mathrm{e}(g)$.

```
module ener controls {d_0, ... d_{k-1}}
  init
  [ ]skip
  update
  [ ]r ∧ χ_{G'}  ->  l' := l + c − Σ_{g∈χ_{G'}} e(g)
```

Figure 3: The module ener

We are now ready to define the encoding game. For a given game $\mathcal{G} = \langle \mathrm{Ag}, \Phi, m_1, \ldots m_M, \gamma_1, \ldots, \gamma_n \rangle$, an energy function e and a maximum capacity value c_{max}, consider the game $\mathcal{G}_{\mathrm{e}} = \langle \{\mathrm{norm}\} \cup \mathrm{Ag}, \Phi', m_{\mathrm{norm}}, m'_1, \ldots, m'_M, \textsc{run}, \textsc{ener}, \gamma_{\mathrm{norm}}, \mathrm{tr}^2(\gamma_1), \ldots, \mathrm{tr}^2(\gamma_n) \rangle$, where $\Phi' = \Phi \cup D \cup \{r\} \cup \{d_0, \ldots, d_{k-1}\}$ and γ_{norm} is an LTL formula provided for the normative agent. In addition, the modules run and ener are included in the list of deterministic modules.

Formally, the game $\mathcal{G}_{\mathrm{e}}$ is identical to the game $\mathcal{G}^{G_M}_{\mathrm{norm}}$ with the addition of the variables d_0, d_{k-1} and the module ener designated to manage variables. Therefore, we can still use the maps Γ and Δ and Lemma 3.8 to transfer the strategic reasoning from the energy game $(\mathcal{G}, \mathrm{e})$ to the game $\mathcal{G}_{\mathrm{e}}$ and vice versa. Moreover, being the energy level explicitly represented in the game by means of the

[1] In order to represent all the natural numbers between 0 and c_{max}, we assume that $c_{max} \leq 2^{k-1}$.

variables $d_0, \ldots, d_{k-1}$, we can express constraints about the energy via a suitable formula over these variables. In particular, we can express the fact that the energy value is always (strictly) positive along the execution with the LTL formula $E_{pos} = \mathtt{XG} \bigvee_{i=0}^{k-1} d_i$ [2], that forces at least one digit to be non-zero and so the represented number to be non-zero as well.

We have the following theorem, whose proof is similar to the one of Theorem 3.9.

Theorem 5.2. *For an energy game $(\mathcal{G}, \mathrm{e})$, a maximum capacity c_{max}, and a formula φ, the following hold:*

1. *There exists an energy norm $\mathcal{N}$ such that NE$(\mathcal{G} \dagger \mathcal{N}) \neq \emptyset$ if, and only if NE$^{\Gamma(\mathcal{N})}(\mathcal{G}_{\mathrm{e}}, E_{pos}) \neq \emptyset$;*
2. *There exists an energy norm $\mathcal{N}$ such that NE-SAT$(\mathcal{G} \dagger \mathcal{N}, \varphi) \neq \emptyset$ if, and only if NE-SAT$^{\Gamma(\mathcal{N})}(\mathcal{G}_{\mathrm{e}}, E_{pos} \wedge \mathrm{tr}^2(\varphi)) \neq \emptyset$;*
3. *There exists an energy norm $\mathcal{N}$ such that NE-SAT$(\mathcal{G} \dagger \mathcal{N}, \varphi) = \emptyset$ if, and only if NE-SAT$^{\Gamma(\mathcal{N})}(\mathcal{G}_{\mathrm{e}}, \varphi) =$ NE$^{\Gamma(\mathcal{N})}(\mathcal{G}_{\mathrm{e}}, E_{pos} \wedge \mathrm{tr}^2(\varphi))$;*

The above theorem shows that the energy norm synthesis problem can be solved by means of a reduction to weak and strong rational synthesis instances. Regarding the complexity, we need to carefully analyse it. As a matter of fact, the size of the module ener is exponential in the number of guarded commands in the original game $\mathcal{G}$. However, it has been shown in [25] that, for a module $m = (\Phi_m, I_m, U_m)$, there exists a Kripke structure $\mathcal{K}_m$ of size at most exponential in the number $|\Phi_m|$ of variables whose runs are exactly the ones compatible with m. Such Kripke structure can be regarded as part of the underlying game structure, on top of which the rational synthesis procedures can be solved in time polynomial with respect to its size. Thus, regarding the ENSNE and ENSE-Nash problems, the overall complexity is 2exptime with respect to the size of the formulas and exptime with respect to the size of the underlying Kripke structure, which is, in turn, of size at most exponential in the number of variables and produces a second 2exptime factor in the overall complexity. Regarding the ENSA-Nash, we have a 3exptime and a 2exptime factor on the size of formulas and number of variables, resulting in an overall 3exptime complexity. We have the following.

Corollary 5.3. *The problems of ENSNE and ENSE-Nash are 2exptime-complete. The problem of ENSA-Nash can be solved in 3exptime.*

6 CONCLUSION

Spurred by the absence of (serviceable) Nash equilibria in a considerable number of multi-agent system instances, we proposed a synthesis mechanism for the equilibrium formation. To this aim, we introduced a new class of synthesis problems, namely *Normative Synthesis*, to generate equilibria by dynamically enabling/disabling agents actions that might cause an equilibrium formation failure. If given full access on the deactivation of actions, this synthesis mechanism results to be powerful enough to always adjust the system in order to enforce desired equilibria.

This positive result allowed us to continue in this direction. We assumed that implementing norms comes with system redesigning costs, and addressed two optimization cases. In the first, a one-off cost for the deactivation of an action is applied and the requirement is to minimize the overall cost for the implementation of the norm. In the second, the deactivation demands for a continuous energy consumption and the norm is constrained to keep fulfilling an energy level requirement. For all these cases, the computational complexity of the problems is between 2exptime and 3exptime, thus not harder than the corresponding rational synthesis problems.

Starting from this, many future directions can be taken. First, one might consider other types of optimization function. For example, the overall cost of a norm implementation can be defined as the mean-payoff value of the cost sequence [52]. A recent work combining qualitative and mean-payoff objective might serve as the starting point for this investigation [26]. Alternatively, designers might be interested in synthesizing norms for either more involving game-theoretical solution concepts, like immune and resilient equilibria [31] or not harming agents' welfare, that is, minimizing the prices of Stability and Anarchy in a game [8, 38, 47]. Work on this direction recently appeared in [4]. We should also consider *Module Checking* [40], a setting in which (contrarily to Normative Synthesis) the constraints over system transitions are managed by an adversarial entity. This has been investigated also for multi-agent systems [35, 36] and a comparison could establish a direct connection between these two notions. Last but not least, starting from the already existing work in the area of multi-agent systems verification [7, 18, 19, 27, 41, 42], an implementation of normative synthesis solver should be considered.

Related work. The concept of repairing systems has been investigated in the literature. In [3], the authors explore the possibility of manipulating the agents objectives (expressed in terms of systems executions) in order to obtain a Nash equilibrium or improving the social welfare. Also in their case, manipulations come with a cost and an optimization problem is analysed. In [37], the authors consider the problem of fixing faults in a finite program as a game in which every successful repairing corresponds to a winning strategy in a suitably defined game.

The notion of energy constraint is also widely explored in the contexts of model-checking and synthesis. In [20], Chatterjee and Doyen introduce a variant of Parity games in which the even player's strategy has the additional requirement of keeping the energy level positive along the whole execution. More recently, ATL* with energy constraint has been introduced [21]. It is worth noticing that ATL* is not capable of expressing the existence of Nash equilbria in games and, more generally, rational synthesis instances [33]. Therefore, it would not have been possible to encase our reasoning in this formalism.

ACKNOWLEDGMENTS

The work is supported by project "d-SynMA" that is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 772459). The author would like to thank Michael Wooldridge, Julian Gutierrez, and Paul Harrenstein for very useful comments on the preliminary versions of this paper.

[2]Note that game starts with the energy level set to 0. Therefore, the formula requires the non-zero condition from the second iteration onwards.

REFERENCES

[1] T. Ågotnes, W. van der Hoek, J. A. Rodriguez-Aguilar, C. Sierra, and M. Wooldridge. 2007. On the Logic of Normative Systems. 1175–1180.

[2] T. Ågotnes, W. van der Hoek, and M. Wooldridge. 2007. Normative System Games. AAMAS'07, 129.

[3] S. Almagor, G. Avni, and O. Kupferman. 2015. Repairing Multi-Player Games. CONCUR'15, 325–339.

[4] S. Almagor, O. Kupferman, and G. Perelli. 2018. Synthesis of Controllable Nash Equilibria in Quantitative Objective Game. IJCAI'18, 35–41.

[5] R. Alur, T.A. Henzinger, and O. Kupferman. 2002. Alternating-Time Temporal Logic. *JACM* 49, 5 (2002), 672–713.

[6] R. Alur and T. A. Henzinger. 1999. Reactive Modules. *Formal Methods in System Design* 15, 1 (1999), 7–48.

[7] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. 1998. MOCHA: Modularity in Model Checking. CAV'98, 521–525.

[8] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. 2008. The Price of Stability for Network Design with Fair Cost Allocation. *SIAM J. Comput.* 38, 4 (2008), 1602–1623.

[9] F. Belardinelli, U. Grandi, A. Herzig, D. Longin, E. Lorini, A. Novaro, and L. Perrussel. 2017. Relaxing Exclusive Control in Boolean Games. In *TARK-17*. 43–56.

[10] F. Belardinelli and A. Lomuscio. 2017. Agent-based Abstractions for Verifying Alternating-time Temporal Logic with Imperfect Information. In *AAMAS'17*. 1259–1267.

[11] F. Belardinelli, A. Lomuscio, and V. Malvone. 2018. Approximating Perfect Recall When Model Checking Strategic Abilities. In *KR'18*. 435–444.

[12] F. Belardinelli, A. Lomuscio, A. Murano, and S. Rubin. 2017. Verification of Broadcasting Multi-Agent Systems against an Epistemic Strategy Logic. In *IJCAI'17*. 91–97.

[13] F. Belardinelli, A. Lomuscio, A. Murano, and S. Rubin. 2017. Verification of Multi-agent Systems with Imperfect Information and Public Actions. In *AAMAS'17*. 1268–1276.

[14] F. Belardinelli, A. Lomuscio, A. Murano, and S. Rubin. 2018. Alternating-time Temporal Logic on Finite Traces. In *IJCAI'18*. 77–83.

[15] F. Belardinelli, A. Lomuscio, A. Murano, and S. Rubin. 2018. Decidable Verification of Multi-agent Systems with Bounded Private Actions. In *AAMAS'18*. 1865–1867.

[16] R. Berthon, B. Maubert, and A. Murano. 2017. Decidability Results for ATL* with Imperfect Information and Perfect Recall. In *AAMAS'17*. 1250–1258.

[17] R. Berthon, B. Maubert, A. Murano, S. Rubin, and M.Y. Vardi. 2017. Strategy logic with imperfect information. In *LICS'17*. 1–12.

[18] P. Cermák, A. Lomuscio, F. Mogavero, and A. Murano. 2018. Practical Verification of Multi-Agent Systems against SLk Specifications. *Inf. Comput.* 261, Part (2018), 588–614.

[19] P. Cermák, A. Lomuscio, and A. Murano. 2015. Verifying and Synthesising Multi-Agent Systems against One-Goal Strategy Logic Specifications. In *AAAI'15*. 2038–2044.

[20] K. Chatterjee and L. Doyen. 2012. Energy Parity Games. *Theor. Comput. Sci.* 458 (2012), 49–60.

[21] D. Della Monica and A. Murano. 2018. Parity-energy ATL for Qualitative and Quantitative Reasoning in MAS. In *AAMAS'18*. 1441–1449.

[22] D. Fisman, O. Kupferman, and Y. Lustig. 2010. Rational Synthesis. In *TACAS'10 (LNCS)*, Vol. 6015. Springer, 190–204.

[23] J. Gutierrez, P. Harrenstein, G. Perelli, and M. Wooldridge. 2016. Expressiveness and Nash Equilibrium in Iterated Boolean Games.. In *AAMAS'16*. 707–715.

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

[25] J. Gutierrez, P. Harrenstein, and M. Wooldridge. 2017. From Model Checking to Equilibrium Checking: Reactive Modules for Rational Verification. *Artif. Intell.* 248 (2017), 123–157.

[26] J. Gutierrez, A. Murano, G. Perelli, S. Rubin, and M. Wooldridge. 2017. Nash Equilibria in Concurrent Games with Lexicographic Preferences. IJCAI'17, 1067–1073.

[27] J. Gutierrez, M. Najib, G. Perelli, and M. Wooldridge. 2018. EVE: A Tool for Temporal Equilibrium Analysis. In *ATVA-18 (Vol 11138 of LNCS)*. Springer, Cham, 551–557.

[28] J. Gutierrez, G. Perelli, and M. Wooldridge. 2016. Imperfect Information in Reactive Modules Games. In *KR'16*. 390–400.

[29] J. Gutierrez, G. Perelli, and M. Wooldridge. 2017. Iterated Games with LDL Goals over Finite Traces. In *AAMAS'17*. 696–704.

[30] J. Gutierrez, G. Perelli, and M. Wooldridge. 2018. Imperfect information in Reactive Modules games. *Inf. Comput.* 261, Part (2018), 650–675.

[31] J. Y. Halpern. 2008. Beyond Nash Equilibrium: Solution Concepts for the 21st Century. In *PODC'08*. 1–10.

[32] X. Huang, J. Ruan, Q. Chen, and K. Su. 2016. Normative Multiagent Systems: The Dynamic Generalization. IJCAI'16, 1123–1129.

[33] J. Gutierrez and P. Harrenstein and G. Perelli and M. Wooldridge. 2017. Nash Equilibrium and Bisimulation Invariance.. In *CONCUR'17*. 17:1–17:16.

[34] W. Jamroga, V. Malvone, and A. Murano. 2017. Reasoning about Natural Strategic Ability. In *AAMAS'17*. 714–722.

[35] W. Jamroga and A. Murano. 2014. On Module Checking and Strategies. In *AAMAS'14*. 701–708.

[36] W. Jamroga and A. Murano. 2015. Module Checking of Strategic Ability. In *AAMAS'15*. 227–235.

[37] B. Jobstmann, A. Griesmayer, and R. Bloem. 2005. Program Repair as a Game. In *CAV'05*. 226–238.

[38] E. Koutsoupias and C. Papadimitriou. 2009. Worst-case equilibria. *Computer Science Review* 3, 2 (2009), 65–69.

[39] O. Kupferman, G. Perelli, and M. Y. Vardi. 2016. Synthesis with Rational Environments. *Annals of Mathematics and Artificial Intelligence* 78, 1 (2016), 3–20.

[40] O. Kupferman and M. Y. Vardi. 1996. Module Checking.. In *CAV'96 (LNCS 1102)*. Springer-Verlag, 75–86.

[41] M. Z. Kwiatkowska, G. Norman, and D. Parker. 2009. PRISM: Probabilistic Model Checking for Performance and Reliability Analysis. *SIGMETRICS Performance Evaluation Review* 36, 4 (2009), 40–45.

[42] A. Lomuscio, H. Qu, and F. Raimondi. 2017. MCMAS: An Open-Source Model Checker for the Verification of Multi-Agent Systems. *STTT* 19, 1 (2017), 9–30.

[43] A.D.C. Lopes, F. Laroussinie, and N. Markey. 2010. ATL with Strategy Contexts: Expressiveness and Model Checking.. In *FSTTCS'10 (LIPIcs 8)*. Leibniz, 120–132.

[44] G. H. Mealy. 1955. A Method for Synthesizing Sequential Circuits. *Bell Labs Technical Journal* 34, 5 (1955), 1045–1079.

[45] Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. 2014. Reasoning About Strategies: On the Model-Checking Problem. *ACM Transaction on Computational Logic* 15, 4 (2014), 34:1–34:47.

[46] F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. 2017. Reasoning about Strategies: on the Satisfiability Problem. *Logical Methods in Computer Science* 13, 1 (2017).

[47] C. H. Papadimitriou. 2001. Algorithms, Games, and the Internet. In *STOC'01*. 749–753.

[48] A. Pnueli. 1977. The Temporal Logic of Programs. In *FOCS-77*. IEEE Computer Society, 46–57.

[49] W. van der Hoek, A. Lomuscio, and M. Wooldridge. 2006. On the Complexity of Practical ATL Model Checking.. In *AAMAS'06*. 201–208.

[50] M. Wooldridge. 2002. *Introduction to Multiagent Systems*. Wiley.

[51] M. Wooldridge, J. Gutierrez, P. Harrenstein, E. Marchioni, G. Perelli, and A. Toumi. 2016. Rational Verification: From Model Checking to Equilibrium Checking. AAAI'16, 4184–4191.

[52] U. Zwick and M. Paterson. 1996. The Complexity of Mean Payoff Games on Graphs. *Theor. Comput. Sci.* 158, 1&2 (1996), 343–359.