Iterated Boolean Games for Rational Verification

Tong Gao, Julian Gutierrez, Michael Wooldridge
Department of Computer Science
University of Oxford

ABSTRACT

Rational verification is the problem of understanding what temporal logic properties hold of a multi-agent system when agents are modelled as players in a game, each acting rationally in pursuit of personal preferences. More specifically, rational verification asks whether a given property, expressed as a temporal logic formula, is satisfied in a computation of the system that might be generated if agents within the system choose strategies for selecting actions that form a Nash equilibrium. We show that, when agents are modelled using the Simple Reactive Modules Language, a widely-used system modelling language for concurrent and multi-agent systems, this problem can be reduced to a simpler query: whether some iterated game—in which players have control over a finite set of Boolean variables and goals expressed as Linear Temporal Logic (LTL) formulae—has a Nash equilibrium. To better understand the complexity of solving this kind of verification problem in practice, we then study the two-player case for various types of LTL goals, present some experimental results, and describe a general technique to implement rational verification using MCMAS, a model checking tool for the verification of concurrent and multi-agent systems.

Keywords

Iterated Boolean Games, Nash Equilibria, Linear Temporal Logic, Formal Verification, MCMAS, Strategy Logic.

1. INTRODUCTION

This paper is concerned with the analysis of concurrent systems composed of multiple software agents, in which each agent is assumed to act strategically and rationally in pursuit of a personal goal, specified as a formula of temporal logic. Since agents are assumed to be acting strategically, game theory provides a very natural set of analytical concepts for such systems [23, 25]. If we interpret a concurrent system as a game in this way, then the outcomes of the game correspond to computations of the system, and game theoretic analysis will identify some computations as being the result of rational action. The main questions to be answered about such systems are therefore not simply "what computations might the system produce?", but rather, "what computations might the system produce if the constituent agents *act rationally*?" If we interpret acting rationally to mean choosing strategies for acting which form a Nash equilibrium, then this question amounts to asking "*which possible computations of the system will be produced in equilibrium*?" Further, if we use temporal logic as the language for expressing properties of our multi-agent system (as is standard in the verification community [4, 12]), then we can also interpret this question as "*which temporal logic formulae are satisfied by computations arising from the selection of strategies in equilibrium*?"

This general problem—understanding what temporal logic properties a system will satisfy, under the assumption that the agents within it act rationally—has been referred to as *rational verification* [31]. Rational verification may be contrasted with the classical view of verification, in which we simply ask whether a temporal logic property holds on some or all possible computations of a system. Expressed as a decision problem (which for technical reasons we refer to as E-NASH), this problem can be stated as follows:

> *Given*: A game G (representing a concurrent/multi-agent system), and temporal logic formula φ.
> *Question*: Is it the case that φ holds on a computation of G that could arise through the agents in G choosing strategies that form a Nash equilibrium?

The counterpart of this problem is A-NASH, which asks whether a temporal formula φ is satisfied on *all* computations that could arise as a result of agents choosing strategies that form a Nash equilibrium. A seemingly simpler question is NON-EMPTINESS, which asks whether a game has any Nash equilibria, a borderline measure of the correct (or desirable) behaviour of a multi-agent system, as a game without any Nash equilibria is inherently unstable.

Related questions have previously been considered by a number of researchers—see *e,g.*, [6, 7, 10, 15, 16, 24]. However, a common feature in this previous work is that the computational models used as the basis for analysis are highly abstract, and in particular are not directly based on real-world programming models or languages. In this paper, we consider multi-agent systems modelled with the *Simple Reactive Modules Language* (SRML), which was introduced to study the complexity of practical ATL model checking [29]. Indeed, SRML is a subset of Reactive Modules [1], a modelling specification language for concurrent and multi-agent systems that is widely used in practical model checking systems [3, 20].

We refer to an SRML model (defining the agents in a system, and the choices available to them) together with a temporal logic formula for each agent (defining the preferences of that agent) as an SRML *game*. SRML games seem to be a natural framework within which to frame questions relating to rational verification. However, when it comes to *solving* such games, that is, implementing algorithms for rational verification against SRML specifications, it would be desirable to deal with a simpler model. In this paper, we show that this is possible. In particular, we show that rational verification for SRML games can be reduced to the NON-EMPTINESS problem for Iterated Boolean Games (iBGs [18]), a model that is strictly

Appears in: *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, S~ao Paulo, Brazil.*

less powerful than SRML games. The remainder of the paper is structured as follows. We start by presenting the framework of Reactive Module Games, and our reduction to Iterated Boolean Games. We then present some experimental results, and a technique to implement rational verification using MCMAS [21].

Motivation and Related Work. Since the early 1990s, a substantial literature has emerged on the verification of multi-agent systems. Initially, this work focussed on the verification of practical reasoning agents—the belief-desire-intention architecture [27]. Later work considered the use of model checking techniques for the verification of multi-agent systems implemented in the BDI programming language AgentSpeak [5]. This strand of work was focussed on the verification of "rational balance" between mental states, rather than verifying strategic properties. Some years later, in the late 1990s, the ATL language was proposed, offering a markedly different perspective on the verification of multi-agent systems [2]. ATL was intended to support the verification of properties relating to the strategic ability of agents and coalitions of agents, and the development of the MOCHA model checker for ATL promoted the rapid adoption of this language [3]. A key issue that was investigated in depth was the relationship between strategic ability and the knowledge of agents [30]. MCMAS was the first model checking tool to implement epistemic ATL model checking [21]. While the view of multi-agent systems embodied by ATL promoted a *strategic* view of multi-agent systems, ATL was not intended for representing and reasoning about *preferences* or game theoretic solution concepts such as Nash equilibrium. One key limitation of ATL with respect to this problem is the inability in ATL to refer to strategies in the object language: Strategy Logic was proposed as a closely-related formalism to ATL, intended to overcome this limitation [9, 24]. The MCMAS model checking tool was later adapted to support Strategy Logic [8], opening up the possibility of model checking strategic properties within different models of interaction and computation. Indeed, since SRML specifications have a semantics given by concurrent game structures [2], the models of Strategy Logic formulae, and Strategy Logic can be used to reason about Nash equilibria, one can analyse SRML games and do rational verification in this way. Following this approach, rational verification is solved, indirectly, via the solution of a parity automaton; see, for instance, [15, 24]. Another instance of a model where strategic behaviour plays a critical role is the Iterated Boolean Games (iBG) framework. The iBGs framework was proposed as a model for reasoning about strategic properties of multi-agent systems in which players have goals expressed as temporal logic formulae [18]. In iBGs, each agent is assumed to control a set of Boolean variables, and has preferences defined by a temporal logic formula that the agent desires to see satisfied. Work on the iBGs framework introduced the three decision problems relating to rational verification that we referred to in the introduction: A-NASH, E-NASH, and NON-EMPTINESS, and established that these problems are 2EXPTIME-complete in cases where agent goals are expressed in Linear Temporal Logic. The iBGs model is closely related to work on *rational synthesis* [15]. However, it has been argued that the iBGs model does not represent a realistic model of concurrent and multi-agent systems. For this reason, some researchers proposed studying instead questions related to rational verification using more realistic system modelling languages: for example, Toumi *et al.* [28] developed a prototype tool for checking equilibria of multi-agent systems in which agents are modelled using SRML, and have preferences expressed as CTL formulae. Wooldridge *et al.* coined the term *rational verification* to refer to the problem of checking what temporal logic properties hold of a game-like concurrent/multi-agent system [31].

2. PRELIMINARIES

Models. Let Φ be a finite set of Boolean variables. A *valuation* for propositional logic is a set $v \subseteq \Phi$, with the interpretation that $p \in v$ means that p is true under valuation v, while $p \notin v$ means that p is false under v. Let $V(\Phi) = 2^{\Phi}$ be the set of all valuations for variables Φ; where Φ is clear, we omit reference to it and write V. We use *Kripke structures* to model the dynamics of our systems. A Kripke structure K over Φ is given by $K = \langle S, S_0, R, \pi \rangle$, where $S = \{s_0, \dots\}$ is a finite non-empty set of *states*, $S_0 \subseteq S$ is the set of *initial states*, $R \subseteq S \times S$ is a total *transition relation* on S, and $\pi : S \to V$ is a valuation function, assigning a valuation $\pi(s)$ to every $s \in S$. Where $K = \langle S, S_0, R, \pi \rangle$ is a Kripke structure over Φ, and $\Psi \subseteq \Phi$, we denote the *restriction of* K *to* Ψ by $K|_{\Psi}$, where $K|_{\Psi} = \langle S, S_0, R, \pi|_{\Psi} \rangle$ is the same as K except that $\pi|_{\Psi}$ is the valuation function defined as follows: $\pi|_{\Psi}(s) = \pi(s) \cap \Psi$.

Runs. A *run of* K is a sequence $\rho = s_0, s_1, s_2, \dots$ where for all $t \in \mathbb{N}$ we have $(s_t, s_{t+1}) \in R$. Using square brackets around parameters referring to time points, we let $\rho[t]$ denote the state assigned to time point t by run ρ. We say ρ is an s-run if $\rho[0] = s$. A run ρ of K where $\rho[0] \in S_0$ is referred to as an *initial* run. Let $runs(K, s)$ be the set of s-runs of K, and let $runs(K)$ be the set of initial runs of K. Notice that a run $\rho \in runs(K)$ induces an infinite sequence $\boldsymbol{\rho} \in V^{\omega}$ of propositional valuations, viz., $\boldsymbol{\rho} = \pi(\rho[0]), \pi(\rho[1]), \pi(\rho[2]), \dots$. We denote by $\mathbf{runs}(K)$ the set of these sequences. Given $\Psi \subseteq \Phi$ and a run $\boldsymbol{\rho} \colon \mathbb{N} \to V(\Phi)$, we denote the restriction of $\boldsymbol{\rho}$ to Ψ by $\boldsymbol{\rho}|_{\Psi}$, *i.e.*, $\boldsymbol{\rho}|_{\Psi}[t] = \boldsymbol{\rho}[t] \cap \Psi$ for each $t \in \mathbb{N}$.

Linear Temporal Logic (LTL). LTL extends propositional logic with two operators, $\mathbf{X}$ ("next") and $\mathbf{U}$ ("until"), that can be used to express properties of runs, *e.g.*, of a Kripke structure. The syntax of LTL is defined w.r.t. a set Φ of Boolean variables as follows:

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

where $p \in \Phi$. The remaining classical logic operators are defined in the standard way; also, we write $\mathbf{F}\varphi = \top \, \mathbf{U} \, \varphi$ and $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$, for "eventually" and "always" respectively. We interpret formulae of LTL with respect to pairs (ρ, t), where ρ is a run of a Kripke structure $K = \langle S, S_0, R, \pi \rangle$ and $t \in \mathbb{N}$ is a temporal index into ρ:

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

If $(\rho, 0) \models \varphi$, we write $\rho \models \varphi$ and say that ρ *satisfies* φ. An LTL formula φ is *satisfiable* if there is a run satisfying φ. A Kripke structure K satisfies φ if $\rho \models \varphi$ for all initial runs ρ of K. Finally, with $|\varphi|$ we denote the size of φ, given by its number of subformulae.

Simple Reactive Modules and Multi-Player Games. We consider concurrent and multi-agent systems described using the Simple Reactive Modules Language (SRML), a language introduced in [29] to study the complexity of practical ATL model checking. Agents in Reactive Modules are known as *modules*, which consist of:

(i) an *interface*, which defines the module's name, the set of Boolean variables under the *control* of the module; and

(ii) a number of *guarded commands*, which define the choices available to the module at every state.

Guarded commands are of two kinds: those used for *initialising* the variables under the module's control (**init** guarded commands), and those for *updating* these variables subsequently (**update** guarded commands). A guarded command has two parts: a condition part (the "guard") and an action part, which defines how to update the value of (some of) the variables under the control of a module. The intuitive reading of a guarded command $\varphi \rightsquigarrow \alpha$ is "if the condition φ is satisfied, then *one of the choices available to the module is to execute the action* α". We note that the truth of the guard φ does not mean that α *will* be executed: only that such a command is *enabled* for execution—that is, that it *may be chosen* for execution.

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

$$\varphi \rightsquigarrow x_1' := \psi_1; \cdots ; x_k' := \psi_k$$

where φ (the guard) is a propositional formula over Φ, each x_i is a controlled variable, and each ψ_i is a propositional logic formula over Φ. Let $guard(g)$ denote the guard of g. Thus, in the above rule, $guard(g) = \varphi$. We require that no variable appears on the left hand side of two assignment statements in the same guarded command. We say that $x_1, \ldots, x_k$ are the *controlled variables* of g, and denote this set by $ctr(g)$. If no guarded command of a module is enabled, the values of all variables in $ctr(g)$ are left unchanged; in SRML notation, if needed, **skip** will refer to this particular case.

Formally, an SRML module, m_i, is defined as a tuple $m_i = \langle \Phi_i, I_i, U_i \rangle$, where: $\Phi_i \subseteq \Phi$ is the (finite) set of variables controlled by m_i; I_i is a (finite) set of *initialisation* guarded commands, such that for all $g \in I_i$, we have $ctr(g) \subseteq \Phi_i$; and U_i is a (finite) set of *update* guarded commands, such that for all $g \in U_i$, we have $ctr(g) \subseteq \Phi_i$. An SRML *arena* is defined to be an $(n+2)$-tuple

$$A = \langle N, \Phi, m_1, \ldots, m_n \rangle$$

where $N = \{1, \ldots, n\}$ is a set of agents, Φ is a finite and non-empty set of Boolean variables, and for each agent $i \in N$, $m_i = \langle \Phi_i, I_i, U_i \rangle$ is an SRML module over Φ that defines the choices available to agent i. It is required that $\{\Phi_1, \ldots, \Phi_n\}$ forms a partition of the set of variables Φ (so every variable in Φ is controlled by some module, and no variable is controlled by more than one module).

The behaviour/semantics of an SRML arena is obtained by executing guarded commands, one for each module, in a synchronous and concurrent way. The execution of an SRML arena proceeds in rounds, where in each round every module $m_i = \langle \Phi_i, I_i, U_i \rangle$ produces a valuation v_i for the variables in Φ_i on the basis of a current valuation v. For each SRML arena A, the execution of guarded commands induces a unique Kripke structure, denoted by K_A, which formally defines the semantics of A. A particular construction is given in [29]. Based on K_A, one can define the sets of runs allowed in A, namely, those associated with the Kripke structure K_A. Similarly, for each SRML module m, the execution of guarded commands induces a unique Kripke structure, denoted by m_A, which formally defines the semantics of m. Sometimes, we will be interested in the size of arenas and modules. We say that the size of an arena $A = \langle N, \Phi, m_1, \ldots, m_n \rangle$, denoted by $|A|$ is $|m_1| + \ldots + |m_n|$, where the size of a module $m_i = \langle \Phi_i, I_i, U_i \rangle$, denoted by $|m_i|$, is $|\Phi_i| + |I_i| + |U_i|$. Also, we will use LTL characterisations of the runs of arenas A and modules m. Such LTL formulae, denoted by $TH(A)$ and $TH(m)$, respectively, are polynomial in the sizes of A and m, and satisfy that, for every run $\boldsymbol{\rho}$, we have $\boldsymbol{\rho} \in \mathbf{runs}(K_A)$ if and only if $\boldsymbol{\rho} \models TH(A)$, and similarly for modules.

Using SRML we can model multi-agent systems represented as multi-player games. That model of games has two components, described next. The first component is an *arena*, which defines the players, the variables they control, and the choices available to them in every game state. The second component defines the *preferences* of all players: every player i is associated with a *goal* γ_i, which will be an LTL formula. The idea is that players desire to see their goal satisfied by the outcome of the game. Formally, an SRML *game* is given by a structure $G = \langle A, \gamma_1, \ldots, \gamma_n \rangle$ where $A = \langle N, \Phi, m_1, \ldots, m_n \rangle$ is an arena with player set N, Boolean variable set Φ, and m_i an SRML module defining the choices available to each player i; moreover, for each $i \in N$, the LTL formula γ_i represents the *goal* that i aims to satisfy. On this basis, the size of a game, $|G|$, is given by $|A| + |\gamma_1| + \ldots + |\gamma_n|$, where $|\gamma_i|$ is the size of γ_i.

Games are played by each player i selecting a *strategy* σ that will define how to make choices over time. Given an SRML arena $A = \langle N, \Phi, m_1, \ldots, m_n \rangle$, a *strategy* for module $m_i = \langle \Phi_i, \mathrm{Vis}_i, I_i, U_i \rangle$ is a structure $\sigma = (Q_i, q_i^0, \delta_i, \tau_i)$, where Q_i is a finite and non-empty set of *states*, $q_i^0 \in Q_i$ is the *initial* state, $\delta_i : Q_i \times V(\Phi) \to Q_i$ is a *transition function*, and $\tau_i : Q_i \to V_i$ is an *output function*. Let Σ_i be the set of strategies for m_i. A strategy σ can be represented by an SRML module (of polynomial size in $|\sigma|$) with variable set $\Phi_i \cup Q_i$. We write m_σ for such a module specification.

Once every player i has selected a strategy σ, a *strategy profile* $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ results and the game has an *outcome*. Because strategies are deterministic, the outcome of a game with SRML arena $A = \langle N, \Phi, m_1, \ldots, m_n \rangle$ is the unique run over Φ induced by $\vec{\sigma}$, which we denote by $\boldsymbol{\rho}(\vec{\sigma})$, that is, the infinite run

$$\bigcup_{i \in N} \tau_i(q_i^0), \bigcup_{i \in N} \tau_i(\delta_i(q_i^0, \bigcup_{i \in N} \tau_i(q_i^0))), \ldots ,$$

which is the unique infinite run associated with the SRML arena $A_{\vec{\sigma}} = \langle N, \Phi \cup \bigcup_{i \in N} Q_i, m_{\sigma_1}, \ldots, m_{\sigma_n} \rangle$ restricted to valuations with respect to Φ, *i.e.*, the Kripke structure $K_{A_{\vec{\sigma}}}|_\Phi$. Since the outcome of a game determines whether each player's goal is or is not satisfied, we can now define a preference relation $\succsim_i$ over outcomes for each player i with goal γ_i. For strategy profiles $\vec{\sigma}$ and $\vec{\sigma}'$, we have

$$\boldsymbol{\rho}(\vec{\sigma}) \succsim_i \boldsymbol{\rho}(\vec{\sigma}') \quad \text{if and only if} \quad \boldsymbol{\rho}(\vec{\sigma}') \models \gamma_i \text{ implies } \boldsymbol{\rho}(\vec{\sigma}) \models \gamma_i.$$

On this basis, we also define the concept of Nash equilibrium [25]: given a game $G = (A, \gamma_1, \ldots, \gamma_n)$, a strategy profile $\vec{\sigma}$ is a *Nash equilibrium* of G if for all players i and all strategies σ', we have

$$\boldsymbol{\rho}(\vec{\sigma}) \succsim_i \boldsymbol{\rho}((\vec{\sigma}_{-i}, \sigma_i')),$$

where $(\vec{\sigma}_{-i}, \sigma_i')$ denotes $(\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$, the strategy profile where the strategy of player i in $\vec{\sigma}$ is replaced by σ_i'. Hereafter, let $NE(G)$ be the set of Nash equilibria of G.

Rational Verification. Once a multi-agent system has been modelled as a multi-player game, from a game-theoretic point of view, a number of queries about the correctness of the system naturally arise. These questions—which represent the game-theoretic counterparts of some core decision problems in automated formal verification from logical specifications—are defined as follows:

> *Given*: SRML game G, LTL formula φ.
> E-NASH: Does $\exists \vec{\sigma} \in NE(G)$. $\boldsymbol{\rho}(\vec{\sigma}) \models \varphi$ hold?
>
> *Given*: SRML game G, LTL formula φ.
> A-NASH: Does $\forall \vec{\sigma} \in NE(G)$. $\boldsymbol{\rho}(\vec{\sigma}) \models \varphi$ hold?
>
> *Given*: SRML game G.
> NON-EMPTINESS: Is it the case that $NE(G) \neq \emptyset$?

These three problems, specifically stated for SRML games, form the core of what it is known as *rational verification* [31]. In what follows, we will show that, for Reactive Modules games, these questions can *all* be reduced to a single decision problem in a much simpler model: namely, to NON-EMPTINESS for iBGs [18].

```
module $rbme_i$ controls $p_i, q_i$
  init
  :: $\top \rightsquigarrow p_i' := \bot, q_i' :=$ B
  update
  :: $(q_j \wedge \neg p_j) \vee p_i \rightsquigarrow p_i' := \top, q_i' := \top$
  :: $(q_j \wedge \neg p_j) \vee p_i \rightsquigarrow p_i' := \bot, q_i' := \top$
  :: $\neg p_i \wedge q_i \rightsquigarrow q_i' := \bot$
```

Figure 1: A mutual exclusion algorithm with 3 processes/agents: $rbme_i$, with $i \in \{0, 1, 2\}$. In this SRML specification, we have $\mathtt{B} = \top$, if $i = 0$, and $\mathtt{B} = \bot$ otherwise. Moreover, $j = (i+2) \bmod 3$.

Example. In order to illustrate, and better understand, how to model multi-agent systems using SRML, while following a game-theoretic approach, we now show an example, using the concrete syntax of SRML, in which three agents are engineered so that some desirable computational properties are guaranteed under rational behaviour.

The multi-agent system we use in this example implements a mutual exclusion algorithm (MEA). MEAs play a key role in distributed systems. They ensure that in concurrent settings two processes cannot enter a 'critical region' (CR) at the same time. Well-known examples of MEAs are, *e.g.*, Peterson's algorithm or Lamport's bakery algorithm [11]. This kind of algorithm may readily be defined in SRML. The modules in Figure 1 specify an SRML arena

$$A^{rbme} = \langle \{0, 1, 2\}, \{p_0, p_1, p_2, q_0, q_1, q_2\}, rbme_0, rbme_1, rbme_2 \rangle,$$

which implements a three-agent distributed ring-based mutual exclusion algorithm (cf., [11], pp. 474–5). The algorithm easily extends to settings with any finite number of agents. The key idea of a ring-based algorithm is that the agents are organised in a cycle along which a 'token' is passed—similar to a distributed implementation of a round-robin scheduler. Possession of the token signifies exclusive permission to enter the CR and an agent can enter the CR only if such an agent is in possession of the token.

Thus, in A^{rbme} each agent i controls two variables, p_i and q_i. Variable p_i being true means that agent i enters the CR, whereas q_i indicates that agent i has the token. In the initial state, no player is in the CR and agent 0 has the token. Once an agent i is in the CR, *i.e.*, if p_i is true, she can stay there indefinitely. On exiting the CR, *i.e.*, if $\neg p_i \wedge q_i$ holds, agent i immediately passes the token to agent $i + 1$, assuming arithmetic modulo 3 throughout the example.

The behaviour of the system, although concurrent and distributed, is rather clear from the SRML specification. In particular, observe that, on all runs of A^{rbme}, at every time, at most one agent can enter the CR—that is, if $p_i = \top$ then $(\neg p_{i+1} \wedge \neg p_{i+2})$ holds. On possession of the token, each agent can decide whether to enter the CR or not. However, to ensure the correct operation of the system, as a protocol designer, we let each agent i have as goal the LTL formula $\gamma_i = \mathbf{GF}\neg q_i$ so that each agent is designed to *rationally* follow a starvation-free protocol (a desirable property in distributed systems which ensures that no process is perpetually denied from having access to shared resources), as explained next.

Clearly, the system allows for many different runs, some of which are undesirable, *e.g.*, one in which an agent enters the CR and remains there indefinitely—this behaviour would violate the desired starvation-free property. However, only some special runs of the system can arise when playing a Nash equilibrium—*i.e.*, only some runs can be rationally sustained by all agents. A powerful property of this SRML specification is that on *all* runs that are sustained by Nash equilibria, the system is guaranteed to be starvation-free! Thus, both *safety* and *liveness* for the system can be uniformly engineered following a simple game-theoretic approach using SRML.

3. REDUCTIONS

The framework of SRML games makes it possible to model multi-agent systems in a relatively natural and high level way. However, this flexibility comes at a price: implementing solvers for SRML games—*i.e.*, tools that automatically do rational verification—is not a straightforward task. In fact, until recently, only a proof-of-concept implementation of a decision problem within the rational verification paradigm had been developed [28, 31], although with respect to goals given by CTL formulae.[1]

Because of this, it would be desirable to be able to do rational verification in a framework simpler than SRML games. In this section we show how to do this. In particular, we show how to reduce the three key decision problems about rational verification in SRML games into a single decision problem on a simpler model of games: the NON-EMPTINESS problem for iterated Boolean Games [18].

3.1 Iterated Boolean Games

Iterated Boolean games (iBGs [18]) are similar to SRML games, but strictly less expressive[2] and with a more succinct representation. Formally, an iBG is a tuple

$$H = \langle N, \Phi, \Phi_1, \ldots, \Phi_n, \gamma_1, \ldots, \gamma_n \rangle$$

where, as in SRML games, $N = \{1, \ldots, n\}$ is a set of agents, Φ is a finite and non-empty set of Boolean variables, and for each agent $i \in N$, Φ_i is the set of Boolean variables controlled by i, which defines the choices available to agent i, namely, the set of valuations $v_i \subseteq V(\Phi_i)$. We also require that $\Phi_1, \ldots, \Phi_n$ forms a partition of Φ, and let each γ_i be the LTL goal of agent i, and strategies be defined as strategies in SRML games, that is, as structures of the form $\sigma_i = (Q_i, q_i^0, \delta_i, \tau_i)$, for each agent i in the game.

Then, in terms of expressive power, the only difference between iBGs and SRML games is that whereas in an iBG each agent i, at each game state, agent i can always choose to play any valuation v_i in $V(\Phi_i)$, in an SRML games the choices of every agent i can be restricted to be a strict subset of Φ_i. On the other hand, even though every iBG H can be represented as an SRML game G, this may be possible only via an unavoidable exponential blow-up in the model representation: the number of guarded commands of G may be required to be exponentially bigger than the number of Boolean variables in H. To see this, consider the following example.

Example. Let $H = \langle \{1\}, \{x, y\}, \{x, y\}, \gamma_1 \rangle$ be a one-player iBG in which agent i has LTL goal γ_1. The SRML game G that represents H is given by $G = \langle \{1\}, \{x, y\}, m_1, \gamma_1 \rangle$, in which the *smallest* SRML module m_1 that can model the behaviour of agent 1 is as follows:

```
module $m_1$ controls $\{x, y\}$
  init
  :: $\top \rightsquigarrow x' := \top;\ y' := \top$
  :: $\top \rightsquigarrow x' := \top;\ y' := \bot$
  :: $\top \rightsquigarrow x' := \bot;\ y' := \top$
  :: $\top \rightsquigarrow x' := \bot;\ y' := \bot$
  update
  :: $\top \rightsquigarrow x' := \top;\ y' := \top$
  :: $\top \rightsquigarrow x' := \top;\ y' := \bot$
  :: $\top \rightsquigarrow x' := \bot;\ y' := \top$
  :: $\top \rightsquigarrow x' := \bot;\ y' := \bot$
```

Thus, clearly the size of G is in general exponentially bigger than the size of H, which is simply given by $|N| + |\Phi| + |\gamma_1| + \ldots + |\gamma_n|$.

[1] In this paper, we consider LTL goals instead of CTL goals as this is the way that, currently, SRML games and iBGs are defined.

[2] For instance, the example in Figure 1 cannot be modelled in iBGs.

3.2 From SRML Games to iBGs

The first reduction is from SRML games to iBGs. In particular, we show that NON-EMPTINESS for SRML games can be reduced to NON-EMPTINESS for iBGs, that is, to the following problem:

> *Given*: iBG H.
> NON-EMPTINESS: Is it the case that $NE(H) \neq \emptyset$?

Formally, we have the following result.

THEOREM 1. *Let $G = \langle N, \Phi, m_1, \ldots, m_n \rangle$ be an* SRML *game. There is an iBG H of size polynomial in G such that*

$$NE(G) \neq \emptyset \quad \textit{if and only if} \quad NE(H) \neq \emptyset$$

PROOF. Let $H = \langle N', \Phi', (\Phi_j)_{j \in N'}, (\gamma_j)_{j \in N'} \rangle$ be the iBG where:

- $N' = N \cup \{n+1, n+2\}$;
- $\Phi' = \Phi \cup \{p, q\}$, with p and q two fresh Boolean variables;
- $\Phi_j = \Phi_i$, if $i = j$; $\Phi_{n+1} = \{p\}$; $\Phi_{n+2} = \{q\}$;
- $\gamma_j = TH(m_i) \wedge \gamma_i$, if $i = j$;
- $\gamma_{n+1} = TH(G) \vee (p \leftrightarrow q)$;
- $\gamma_{n+2} = TH(G) \vee \neg(p \leftrightarrow q)$;

such that $TH(G) = \bigwedge_{i \in N} TH(m_i)$. That the size of H is polynomial in the size of G is obvious from the above construction.

Now, we prove the rest of the statement by showing both implications. For the left-to-right direction, first suppose that $\vec{\sigma} \in NE(G)$, and let every agent j in H, with $i = j$, use a strategy σ_j^* that plays consistently with $\sigma_i = (Q_i, q_i^0, \delta_i, \tau_i)$ in $\vec{\sigma}$. Formally, such a strategy $\sigma_j^* = (Q_j, q_j^0, \delta_j, \tau_j)$ is defined as follows: $Q_j = Q_i$; $q_j^0 = q_i^0$; $\tau_j = \tau_i$; and $\delta_j(q, v \cup v') = \delta_i(q, v)$, for every $v \in V(\Phi)$ and $v' \in V(\{p, q\})$. Moreover, let agents $n+1$ and $n+2$ use any available strategy, say σ_{n+1} and σ_{n+2}. Let $\vec{\sigma}_H$ be the strategy profile $(\sigma_1^*, \ldots, \sigma_n^*, \sigma_{n+1}, \sigma_{n+2})$. Firstly, observe that agents $n+1$ and $n+2$ get their goals satisfied in H since

$$\boldsymbol{\rho}(\vec{\sigma}_H) \models \gamma_{n+1}$$

and

$$\boldsymbol{\rho}(\vec{\sigma}_H) \models \gamma_{n+2}$$

—because $TH(G)$ is satisfied.

Moreover, for every $i \in N$, if agent i gets its goal γ_i achieved in G, then agent j, with $i = j$, will also get its goal achieved in H, since

$$\boldsymbol{\rho}(\vec{\sigma}_H) \models TH(m_i) \wedge \gamma_i\ .$$

In addition, agents who do not get their goal achieved in G cannot beneficially deviate in H. Suppose σ_j' is a deviation from σ_j^* in H (in a way, a deviation from σ_i of $\vec{\sigma}$). Then, necessarily, we have

$$\boldsymbol{\rho}((\vec{\sigma}_H)_{-j}, \sigma_j') \not\models TH(m_i) \wedge \gamma_i\ .$$

If, on the one hand,

$$\boldsymbol{\rho}((\vec{\sigma}_H)_{-j}, \sigma_j') \models TH(m_i)$$

then

$$\boldsymbol{\rho}((\vec{\sigma}_H)_{-j}, \sigma_j') \not\models \gamma_i$$

because $\vec{\sigma} \in NE(G)$, and neither any $TH(m_i)$ nor any γ_i depends on p or q. If, on the other hand,

$$\boldsymbol{\rho}((\vec{\sigma}_H)_{-j}, \sigma_j') \models \gamma_i$$

then

$$\boldsymbol{\rho}((\vec{\sigma}_H)_{-j}, \sigma_j') \not\models TH(m_i)$$

as otherwise σ_j' would determine a beneficial deviation in G for agent i, with $i = j$, which is impossible since $\vec{\sigma} \in NE(G)$. Then, either way we have

$$\boldsymbol{\rho}((\vec{\sigma}_H)_{-j}, \sigma_j') \not\models TH(m_i) \wedge \gamma_i$$

for every agent j and alternative strategy σ_j'. Thus, we can conclude that $\vec{\sigma}_H \in NE(H)$, and therefore that $NE(H) \neq \emptyset$, as required.

For the right-to-left direction, suppose that $\vec{\sigma} \in NE(H)$. Since $\vec{\sigma}$ is a Nash equilibrium then $\boldsymbol{\rho}(\vec{\sigma}) \models \gamma_{n+1}$ and $\boldsymbol{\rho}(\vec{\sigma}) \models \gamma_{n+2}$. This means that every agent j in H plays consistently with the choices available in G, since

$$TH(G) = \bigwedge_{i \in N} TH(m_i)$$

must be satisfied.

Then, let every agent i in G use a strategy σ_i^* that plays consistently with $\sigma_j = (Q_j, q_j^0, \delta_j, \tau_j)$ of $\vec{\sigma}$, for every $i = j$, which is possible only because $TH(G)$ is satisfied. Formally, such a strategy $\sigma_i^* = (Q_i, q_i^0, \delta_i, \tau_i)$ is defined as follows: $Q_i = Q_j$; $q_i^0 = q_j^0$; $\tau_i = \tau_j$; and $\delta_i(q, v) = \delta_i(q, v \cup v')$, for every $v \in V(\Phi)$ and any $v' \in V(\{p, q\})$. Let $\vec{\sigma}_G$ be the strategy profile $(\sigma_1^*, \ldots, \sigma_n^*)$ constructed in this way. Therefore, if

$$\boldsymbol{\rho}(\vec{\sigma}) \models TH(m_i) \wedge \gamma_i$$

in H, then

$$\boldsymbol{\rho}(\vec{\sigma}_G) \models \gamma_i$$

in G. Moreover, if

$$\boldsymbol{\rho}(\vec{\sigma}) \not\models TH(m_i) \wedge \gamma_i$$

in H, then

$$\boldsymbol{\rho}(\vec{\sigma}_G) \not\models \gamma_i$$

in G, because $TH(m_i)$ must be satisfied in H. In addition, every agent i in G who does not get its goal γ_i satisfied cannot beneficially deviate. For a contradiction, suppose that such an agent and strategy exit, *i.e.*, suppose that there is an agent i and strategy σ_i' such that

$$\boldsymbol{\rho}(((\vec{\sigma}_G)_{-i}, \sigma_i')) \models \gamma_i$$

with $\boldsymbol{\rho}(\vec{\sigma}_G) \not\models \gamma_i$ in G.

Then, we know that there is a strategy $\sigma_j' = \sigma_j^*$ for agent j, with $j = i$, defined as in the left-to-right part of this proof, in this case constructed from the strategy profile $((\vec{\sigma}_G)_{-i}, \sigma_i')$, such that

$$\boldsymbol{\rho}((\vec{\sigma}_{-j}, \sigma_j')) \models TH(m_i) \wedge \gamma_i$$

in H, which is impossible because $\vec{\sigma} \in NE(H)$. □

Note that Theorem 1 does not state that iBGs are as expressive as SRML games. We know that this is not the case since the class of Kripke structures (arenas) that can be defined by SRML games is strictly bigger than the class of Kripke structures defined by iBGs. What Theorem 1 shows is simply that checking whether an SRML game G has a Nash equilibrium can be reduced to checking whether some iBG constructed from G also has a Nash equilibrium—and that this can be done at the same computational complexity cost since the constructed iBG is polynomial in the size of G. It is known, and not hard to see, that the other direction also holds (*i.e.,* that the NON-EMPTINESS problem for iBGs can be reduced to NON-EMPTINESS for SRML games), but at the cost of an exponential blow-up in the worst case scenario, as illustrated in the example given before.

3.3 From E-Nash to Non-Emptiness

In this section we show a second reduction, namely, that E-NASH can be reduced to NON-EMPTINESS, regardless of whether the question is asked for SRML games or iBGs. It should be noted that a reduction in the other direction is trivial, and already known [18]: NON-EMPTINESS is E-NASH in case $\varphi = \top$. Obviously, such a reduction takes constant time, as it does the one presented next.

THEOREM 2. *Let G be a game (either an iBG or an* SRML GAME*) and* φ *be an LTL formula. There is a game H (an iBG or an* SRML *game, correspondingly), of constant size in G, such that*

$$NE(H) \neq \emptyset \quad \textit{if and only if} \quad \exists \vec{\sigma} \in NE(G) \; . \; \boldsymbol{\rho}(\vec{\sigma}) \models \varphi$$

PROOF. Since the statement, and proof, uniformly applies to both iBGs and SRML games, let us call G simply a game of an iBG or an SRML game. Now, let H be the game G which in addition has two more 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 $\Phi_{n+2} = \{q\}$ for two fresh Boolean variables p and q. Also, for a given strategy profile $\vec{\sigma}$, say in G/H, let $\vec{\sigma}'$ be the strategy profile in H/G that can be constructed from $\vec{\sigma}$ just as described in the proof of Theorem 1, that is, the strategy profile that adds/removes two agents to/from $\vec{\sigma}$. Now, we prove the statement by showing both implications.

For the right-to-left direction, suppose that, for some $\vec{\sigma}$ in G, we have $\vec{\sigma} \in NE(G)$ and $\boldsymbol{\rho}(\vec{\sigma}) \models \varphi$. Then,

$$\boldsymbol{\rho}(\vec{\sigma}') \models \gamma_{n+1}$$

and

$$\boldsymbol{\rho}(\vec{\sigma}') \models \gamma_{n+2}$$

for all strategies σ_{n+1} and σ_{n+2} in H. Obviously, if $\vec{\sigma} \in NE(G)$ then $\vec{\sigma}' \in NE(H)$ because no player can beneficially deviate and $n+1$ and $n+2$ will get their goals satisfied since, in particular, φ must be satisfied.

Now, for the left-to-right direction, suppose that $\vec{\sigma} \in NE(H)$, for some $\vec{\sigma}$ in H. Because of players $n+1$ and $n+2$, we know that

$$\boldsymbol{\rho}(\vec{\sigma}) \models \varphi$$

in H. Then, it follows that

$$\boldsymbol{\rho}(\vec{\sigma}') \models \varphi$$

in G. Agents who get their goal achieved in H will necessarily get their goal achieved in G. Moreover, agents who do not get their goal achieved in H do not have a beneficial deviation in G. If they did such a beneficial deviation would also be possible in H, which is impossible since $\vec{\sigma} \in NE(H)$. Therefore, $\vec{\sigma}' \in NE(G)$ and $\vec{\sigma}' \models \varphi$ in G, as required. Finally, that H is of constant size in the size of the game G immediately follows from the construction of H. □

3.4 From A-Nash to Non-Emptiness

The final reduction, namely from A-NASH for SRML games to NON-EMPTINESS for iBGs, goes via E-NASH using Theorems 1 and 2. This reduction is simple, and relies on the fact that A-NASH and E-NASH are (logically) dual problems. Let G be a game and φ an LTL formula. To solve A-NASH with respect to G and φ, we can simply ask if E-NASH for G and $\neg\varphi$ can be answered negatively. If it does, then, for all $\vec{\sigma}$ in $NE(G)$ we have $\boldsymbol{\rho}(\vec{\sigma}) \models \varphi$, either because $NE(G)$ is empty or because every $\vec{\sigma}$ in $NE(G)$ satisfies φ. Clearly, this reduction does not involve any exponential blowups, as in previous cases, and therefore it follows that all questions about rational verification for SRML games can be reduced to NON-EMPTINESS for iBGs. We now present some experimental results.

4. IBGS IN PRACTICE

As shown in the previous section, rational verification for multiagent systems modelled as SRML games can be solved by solving the NON-EMPTINESS problem for iBGs. Since all reductions presented before are at most polynomial in the size of the given SRML games, one can do this without paying any (theoretical) computational complexity cost. This is true only for games with two or more agents: whereas rational verification is 2EXPTIME-complete in the general case, it can be solved in PSPACE for games with only one agent. Then, at least in theory, rational verification is as hard for two-player games as it is for n-player games, with $n \geq 2$. However, it is only reasonable to expect that in practice it may be easier to solve two-player games than to solve n-player games, if $n > 2$.

Because of this reason we will, in this section, first explore the setting that we believe provides an *experimental lower bound* to rational verification: solving the NON-EMPTINESS problem for iBGs with only two agents. Clearly, with respect to the number of agents in a multi-agent system, no problem about rational verification would be in practice easier to solve than this case. In particular, we will study how to solve in practice the NON-EMPTINESS for two-player iBGs of imperfect recall and perfect information.[3]

4.1 Non-Emptiness via Synthesis

In [17] it was shown that NON-EMPTINESS for two-player iBGs can be solved through the solution of a number of synthesis problems for CTL* formulae [13]. Using this technique we can implement a solver for NON-EMPTINESS with the use of any tool or library for CTL* synthesis. In order to obtain the experimental results presented in this paper, we have made used of a recent implementation for MCMAS [21] of an algorithm for model checking Strategy Logic (SL) specifications [8, 9]. In particular, using SL we can express the CTL* problems that need to be solved in order to find a solution for NON-EMPTINESS in two-player iBGs. These CTL* problems, whose solution we have implemented in MCMAS, are determined by Theorem 3, given below. In order to be able to formally state the theorem, we need to introduce some definitions first.

A synthesis problem can be seen as a two-player zero-sum game in which one player, the *Verifier*, tries to show that a given formula φ (over a set of Boolean variables Φ) can be satisfied, whereas the other player, the *Falsifier*, tries to show that this is not the case. The choices of the players are given by Φ. Whereas the Verifier controls the variables in $\Phi_1 \subseteq \Phi$, the Falsifier controls $\Phi_2 = \Phi \setminus \Phi_1$. In case φ is a temporal logic formula, the game proceeds in rounds, by each player making a choice given by $v_i \in V(\Phi_i)$, with $i \in \{1, 2\}$—just as in an iBG. The infinite sequence of joint choices/valuations built in this way determines whether φ can or cannot be synthesised: if Verifier has a winning strategy in this game, then φ can be synthesised; if Falsifier has such a winning strategy, then φ cannot be synthesised. Since these games are determined, a winning strategy always exists. We write SYNTH$(\varphi(\Phi),\Phi_1)$ for a synthesis problem, where φ is a formula over Φ that we want to analyse and Φ_1 is the set of variables controlled by Verifier.

The logic we use to solve NON-EMPTINESS via synthesis is CTL*, an extension of LTL with an existential ("$\exists\varphi$") and a universal ("$\forall\varphi$") quantifier. The former, $\exists\varphi$, expresses that *"there is a path where* φ *holds"* and the latter, ("$\forall\varphi$"), that *"in every path* φ *holds"*. Details on the formal semantics of CTL* can be found in [12, 13]. With these definitions in place, we can now state Theorem 3.

[3] It is known that whereas two-player SRML games with perfect information are as hard as n-player SRML games with perfect information, with $n \geq 2$, this is not the case for games with imperfect information. In such a case while the two-player case is decidable, and in 2EXPTIME, the n-player case is undecidable for every $n > 2$.

Theorem 3 (from [17]). *Let* $G = \langle \{1,2\}, \Phi, \Phi_1, \Phi_2, \gamma_1, \gamma_2 \rangle$ *be a two-player iBG. Then, NE*$(G) \neq \emptyset$ *if and only if either*

- Synth$(\gamma_1 \wedge \gamma_1, \Phi)$*; or*
- Synth$(\neg\gamma_2, \Phi_1)$ *and* Synth$(\neg\gamma_1, \Phi_2)$*; or*
- Synth$(\exists\gamma_1 \wedge \forall\neg\gamma_2, \Phi_1)$ *or* Synth$(\exists\gamma_2 \wedge \forall\neg\gamma_1, \Phi_2)$.

As shown by Theorem 3 (for proof details see [17, 19]), Non-Emptiness for iBGs with two players can be reduced to checking a number of synthesis problems. Our implementation in MCMAS of Theorem 3 uses SL, a logic to reason about strategic behaviour in multi-player games. SL also extends LTL, in this case with three new operators: an existential strategy quantifier $\langle\langle x \rangle\rangle$, a universal strategy quantifier $[[x]]$, and an agent binding operator (i, x). Intuitively, these three new operators can be read as *"there is a strategy x"*, *"for every strategy x"*, and *"let agent i use the strategy associated with x"*, respectively. Details on the formal semantics of SL can be found, *e.g.*, in [9]. With the game-theoretic interpretation of a synthesis problem given before, we can use SL, along with MCMAS, to express and solve the synthesis problems considered in Theorem 3.

4.2 Experimental Results using MCMAS

It should be immediately apparent that rational verification must be expected to be a very demanding problem from a practical point of view: the 2EXPTIME LTL synthesis problem is a simple special case, which can be seen as a two-agent turn-based zero-sum perfect-information scenario [26]. Thus, to check the practical complexity of automatically solving the problems we have investigated in this paper, we use (LTL) goal specifications of four types: safety, reachability, stutter-invariant, and general specifications. Each of these types can be syntactically characterised in LTL as follows.

Safety specifications, which are used to ensure that a system never visits an invalid state, are characterised by the **G**-*fragment of LTL*, the sublogic of LTL where **G** is the only temporal operator. *Reachability Specifications*, which are used to ensure that a program/system eventually visits a desired state (for instance, to check for program termination), are characterised by the **F**-*fragment of LTL*, the sublogic of LTL where **F** is the only temporal operator. *Stutter-invariant specifications*, which combine any Boolean combinations of both safety and reachability goals and disallow the use of **X**, are characterised by the (G,F)-*fragment of LTL*; this type of specifications are used for modular reasoning and system refinement since they do not take into account repetitions of states in execution runs of a system. Finally, we also consider *general LTL specifications*, where **U** and **X** are not restricted in any way.

The performance of our implementation of two-player iBGs in MCMAS mainly depends on two factors: the size of the Kripke structure induced by the game, and the complexity of the goals of the players. Our experiments show that, overall, regardless of the type of specifications used, MCMAS computes a solution for Non-Emptiness in less than 10 minutes for systems with a few dozens of states. Small differences, in the order of a few seconds, can be seen (increasingly in execution times) when moving from safety to reachability and from reachability to stutter-invariant specifications. For general LTL specifications, the execution time starts growing more rapidly with respect to stutter-invariant properties. Moreover, only in cases where the only modal operator used in system specifications is "next" (**X**), MCMAS is able to handle bigger systems in less than 10 minutes. For instance, in this case, the systems that can be handled may have up to 10 thousand states. This clearly shows a performance below desirable for industrial-size systems, which can have thousands of states, but it is still an acceptable execution time to check systems where only up to a hundred states are generated.

Remark:. At this point it is important to note that checking the existence of Nash equilibria in a game with LTL goals is very difficult for a human, even for systems with no more than 10 states as in such a case there can be up to 10! (~3.6 million) different memoryless strategy profiles in the game, should a manual or a brute-force algorithm were used to solve the problem.

4.3 General iBGs in MCMAS

In the previous section we studied how complex is to solve Non-Emptiness for two-player iBGs using MCMAS, where the goal was to find experimental lower bounds for the problem. However, in theory, MCMAS, along with SL, can be used to solve the Non-Emptiness problem for (imperfect-recall) iBGs with any number of variables, agents, and types of LTL goals—and therefore any rational verification problem studied before. In this section, we present how to do this by presenting a polynomial translation from iBGs to ISPL, the specification language used by MCMAS.

ISPL is a language to model *interpreted systems* [14]. A system in ISPL consists of a set of *agents*, together with a designated agent *Environment*. It also specifies a set of *initial states* and a set of *temporal logic goals* (which can be given in SL) that can be checked against the system. Moreover, an agent has at least five parts: a *name*, a set of *controlled variables*, a set of *actions*, a *protocol*, which indicates the actions that an agent can take at a given state, and an *evolution* part, which indicates how to update the values of the controlled variables depending on the actions that can be taken by the agent; thus, an agent selecting an action has the effect of updating the values of the variables that such an agent controls. Agents interact with one another, as well as with the agent Environment, as in an iBG: at each time step, every agent chooses an action, which defines a valuation for the variables it controls. Temporal logic goals are then interpreted in the possible execution runs of the multi-agent system defined in this way.

Example. A trivial, but exponential, translation of an iBG into ISPL can be implemented as follows. Take, for instance, the agent (of the iBG) associated with the SRML module, m_1, presented in Section 3.1. Such an agent is translated into ISPL as shown below:

```
Agent m1
  Vars:
     x:boolean; y:boolean;
  end Vars
  Actions = {ac00,ac01,ac10,ac11};
  Protocol:
    Other : {ac00,ac01,ac10,ac11};
  end Protocol
  Evolution:
    (x=false) and (y=false) if Action=ac00;
    (x=false) and (y=true) if Action=ac01;
    (x=true) and (y=false) if Action=ac10;
    (x=true) and (y=true) if Action=ac11;
  end Evolution
end Agent
```

Figure 2: ISPL representation of an agent in an iBG. In this example, the agent controls two Boolean variables, namely x and y.

While simple, the ISPL representation of agents illustrated in Figure 2 clearly shows an unavoidable exponential blow-up, if a trivial translation is used. To remedy this problem, we propose a different translation, polynomial in the size of the input iBG, in which we again follow a logic-based approach: in this case, we use the power of SL to avoid the undesirable exponential blow-up.

```
Agent varx
  Vars:
     x:boolean;
  end Vars
  Actions = {ac0,ac1};
  Protocol:
    Other : {ac0,ac1};
  end Protocol
  Evolution:
    x=false if Action=ac0;
    x=true if Action=ac1;
  end Evolution
end Agent
```

Figure 3: ISPL model of an agent that controls a single variable.

A Polynomial Translation. A translation from iBGs into ISPL—and therefore from SRML games into ISPL—that does not result in an exponential blow-up can be defined as follows. First define, for every variable in the given iBG, an agent in ISPL. Therefore, if the given iBG has n Boolean variables, then the resulting ISPL specification has $n + 1$ agents (one per Boolean variable plus the environment agent). Then, such an agent, for a Boolean variable x is as shown in Figure 3. Secondly, use the power of SL to specify that a set of ISPL agents corresponds to a single agent in the iBG given as input. This can be done, *e.g.*, in the following way. An SL expression that characterises the existence of a Nash equilibria is:

$$\begin{aligned}\varphi_{NE} \quad = \quad & \langle\langle st1\rangle\rangle \ldots \langle\langle stn\rangle\rangle(1, st1)\ldots(n, stn) \\ & \textstyle\bigwedge_{i\in\{1,\ldots,n\}}(\neg\gamma_i \to [[altsti]](i, altsti)\neg\gamma_i)\ .\end{aligned}$$

Informally, φ_{NE} expresses that there is a strategy profile $(st1, \ldots, stn)$ such that, for every agent i, if i does not have its goal γ_i achieved by the unique run $\boldsymbol{\rho}((st1, \ldots, stn))$ induced by such a strategy profile, and decides to use any alternative strategy, say *altsti*, then agent i does not get its goal achieved in such a case either, that is, γ_i is not satisfied by the run $\boldsymbol{\rho}((st1, \ldots, altsti, \ldots, stn))$ either.

We can now use the power of SL to specify cooperation between agents, so that, if an agent, say i, in an iBG controls a set of Boolean variables $\Phi_i = \{x, y, \ldots, z\}$, then the ISPL agents corresponding to this iBG player i (that is, the ISPL agents `varx, vary, ..., varz` using the notation in Figure 3) can both cooperate and jointly deviate to achieve γ_i. In order to present an SL expression that characterises the kind of behaviour just described, let us use the following notation: given an iBG $H = \langle N, \Phi\ (\Phi)_{i\in N}, (\gamma_i)_{i\in N}\rangle$, the corresponding agents in ISPL will have exactly the same name as the Boolean variable they represent. For instance, variable $x \in \Phi$ has an associated ISPL agent `x`. Having this in mind, the desired SL expression that expresses the existence of Nash equilibria (NON-EMPTINESS) in the resulting ISPL system can be written as follows:

$$\varphi'_{NE} = \langle\langle stx\rangle\rangle \ldots \langle\langle stz\rangle\rangle(x, stx)\ldots(z, stz) \bigwedge_{i\in N}(\neg\gamma_i \to \psi_i)$$

where

$$\psi_i = [[astl]] \ldots [[astm]](l, astl)\ldots(m, astm)\neg\gamma_i$$

such that $\Phi_i = \{l, \ldots, m\}$.

Thus, using this SL specification, we can effectively translate any iBG into an ISPL specification of polynomial size. This, in turn, allows one to represent SRML games as polynomially sized ISPL systems/specifications so that rational verification for SRML games (with imperfect recall) can be automated using MCMAS.

5. CONCLUSION & FUTURE WORK

Model Checking vs. Rational Verification. Since the introduction of temporal logics, such as LTL or CTL [12], formal verification has been a very active area of research, which has led to the development of an impressive number of tools and techniques. Perhaps, the most successful technique within formal verification is model checking, which nowadays has got mature enough to handle systems of industrial size. Rational verification, on the other hand, is still in its infancy: the main ideas and techniques underlying it are under development, while current tool support can only handle systems of small size. But rational verification is expected to be a computationally harder problem, and this can easily be seen from its game-theoretic presentation. While the most basic questions about traditional formal verification, including model checking, can be solved by reductions to two-player zero-sum games, in rational verification all essential questions consider n-player non-zero-sum games, which at least in practice are usually harder to solve. Rational verification is also different from model checking in the kinds of properties that each technique tries to check: while model checking is interested in correctness with respect to *all* possible behaviours of a system, rational verification is interested only in behaviours that can be *sustained by a Nash equilibrium*, when a system is modelled as a game. This, in particular, adds a new ingredient to the verification problem, as it is now necessary to take into account players' preferences with respect to the possible runs of the system.

SRML games vs. iBGs. In this paper we have shown how to reduce the main questions about the equilibrium analysis of SRML games into iBGs, and how to automatically solve them in practice using the MCMAS model checking tool. Then, one may wonder why not to consider iBGs in the first place instead. There are at least two main reasons for this. Firstly, iBGs are strictly less powerful than SRML games.[4] While in an SRML game one can impose constraints on the actions that an agent can take at any given time, in iBGs this is not possible: at any time step the choices available to an agent are exactly *all* the possible valuations of the variables that such an agent controls. From a *modelling* point of view this may be an undesirable aspect of iBGs. Secondly, SRML games are defined directly based on a specification language used by a number of *practical* model checking tools, *e.g.*, MOCHA [3] or PRISM [20], which already provide quite a large number of reasoning techniques for SRML specifications—although their toolkits of reasoning techniques do not currently include algorithms for rational verification.

Future Work. On the one hand, from a theoretical point of view, it would be interesting to know whether some, or all, of our ideas may transfer to other settings, such as games with quantitative payoffs, imperfect information, or different preference relations. A first step could be to consider more general preferences relations, for instance, as done in Lukasiewicz games [22]. On the other hand, from a practical point of view, it would also be desirable to check if rational verification can be implemented more efficiently using other model checking systems, or if alternative algorithms for the NON-EMPTINESS problem can be found—which may not rely on the solution of synthesis problems or on the use of SL. Alternatively, one could consider SL directly over SRML games or iBGs only.

Acknowledgements. We thank the financial support of the ERC Advanced Investigator grant 291528 ("RACE") at Oxford.

[4] Formally, the class of Kripke structures induced by iBGs is strictly smaller than the class of Kripke structures defined by SRML arenas.

REFERENCES

[1] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(11):7–48, July 1999.

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

[3] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Taşiran. Mocha: Modularity in model checking. In *CAV 1998: Tenth International Conference on Computer-aided Verification, (LNCS Volume 1427)*, pages 521–525. Springer-Verlag: Berlin, Germany, 1998.

[4] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press: Cambridge, MA, 2008.

[5] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking agentspeak. In *Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, Columbia University, NY, USA, July 2003.

[6] P. Bouyer, R. Brenguier, N. Markey, and M. Ummels. Pure nash equilibria in concurrent games. *Logical Methods in Computer Science*, 2015.

[7] N. Bulling, W. Jamroga, and J. Dix. Reasoning about temporal properties of rational play. *Annals of Mathematics and Artificial Intelligence*, 53(1–4):51–114, 2008.

[8] P. Cermák, A. Lomuscio, F. Mogavero, and A. Murano. MCMAS-SLK: A model checker for the verification of strategy logic specifications. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, pages 525–532, 2014.

[9] K. Chatterjee, T. Henzinger, and N. Piterman. Strategy logic. *Information and Computation*, 208(6):677–693, June 2010.

[10] K. Chatterjee and T. A. Henzinger. A survey of stochastic omega-regular games. *Journal Of Computer And System Sciences*, 78:394–413, 2012.

[11] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison Wesley, 2005.

[12] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, pages 996–1072. Elsevier, 1990.

[13] E. A. Emerson and J. Y. Halpern. 'Sometimes' and 'not never' revisited: on branching time versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[14] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. The MIT Press: Cambridge, MA, 1995.

[15] D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 190–204. Springer, 2010.

[16] J. Gutierrez, P. Harrenstein, and M. Wooldridge. Reasoning about equilibria in game-like concurrent systems. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 408–417. AAAI Press, 2014.

[17] J. Gutierrez, P. Harrenstein, and M. Wooldridge. Expresiveness and complexity results for strategic reasoning. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, pages 268–282, 2015.

[18] J. Gutierrez, P. Harrenstein, and M. Wooldridge. Iterated boolean games. *Information and Computation*, 242:53–79, 2015.

[19] J. Gutierrez, G. Perelli, and M. Wooldridge. Imperfect information in reactive modules games. In *Principles of Knowledge Representation and Reasoning (KR)*, pages 390–400. AAAI Press, 2016.

[20] M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009.

[21] A. Lomuscio and F. Raimondi. MCMAS: a tool for verifying multi-agent systems. In *Proceedings of The Twelfth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-2006)*. Springer-Verlag: Berlin, Germany, 2006.

[22] E. Marchioni and M. Wooldridge. Lukasiewicz games: A logic-based approach to quantitative strategic interactions. *ACM Transactions on Computational Logic*, 16(4):33:1–33:44, 2015.

[23] M. Maschler, E. Solan, and S. Zamir. *Game Theory*. Cambridge University Press: Cambridge, England, 2013.

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

[25] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press: Cambridge, MA, 1994.

[26] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *International Colloquium on Automata, Languages, and Programs (ICALP)*, volume 372 of *LNCS*, pages 652–671. Springer, 1989.

[27] A. S. Rao and M. P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 318–324, Chambéry, France, 1993.

[28] A. Toumi, J. Gutierrez, and M. Wooldridge. A tool for the automated verification of Nash equilibria in concurrent games. In *Proceedings of the Twelfth International Colloquium on Theoretical Aspects of Computing (ICTAC 2015)*, Cali, Colombia, 2015.

[29] W. van der Hoek, A. Lomuscio, and M. Wooldridge. On the complexity of practical ATL model checking. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 201–208. ACM, 2006.

[30] W. van der Hoek and M. Wooldridge. Time, knowledge, and cooperation: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.

[31] M. Wooldridge, J. Gutierrez, P. Harrenstein, E. Marchioni, G. Perelli, and A. Toumi. Rational verification: From model checking to equilibrium checking. In *Thirtieth AAAI Conference on Artificial Intelligence (AAAI-2016)*, 2016.