Applied Intelligence manuscript No.
(will be inserted by the editor)

Rational Verification: A Progress Report

Alessandro Abate · Julian Gutierrez · Lewis Hammond · Paul Harrenstein · Marta Kwiatkowska · Muhammad Najib · Giuseppe Perelli · Thomas Steeples · Michael Wooldridge

Received: DD Month YEAR / Accepted: DD Month YEAR

Wooldridge, Gutierrez, Harrenstein, and Perelli acknowledge the support of the ERC under grant 291528 ("RACE"). Wooldridge and Harrenstein further acknowledge the support of the Alan Turing Institute, London. Kwiatkowska acknowledges the support of the ERC under the grant 834115 ("FUN2MODEL") and the EPSRC Programme Grant on Mobile Autonomy (EP/M019918/1). Abate acknowledges the HICLASS project (113213), a partnership between the Aerospace Technology Institute (ATI), Department for Business, Energy & Industrial Strategy (BEIS) and Innovate UK. Hammond acknowledges the support of an EPSRC Doctoral Training Partnership studentship (Reference: 2218880). Harrenstein was furthermore supported by the ERC under grant 639945 ("ACCORD"). Perelli was furthermore supported by the ERC under grant 834228 ("WHITEMECH") and the EU ICT-48 2020 project TAILOR (No. 952215). Steeples gratefully acknowledges the support of the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems EP/L015897/1 and the Ian Palmer Memorial Scholarship. Najib acknowledges the support of the ERC European Union's Horizon 2020 research and innovation programme (grant 759969).

A. Abate
Department of Computer Science, University of Oxford. E-mail: aabate@cs.ox.ac.uk

J. Gutierrez
Faculty of Information Technology, Monash University. E-mail: julian.gutierrez@monash.edu

L. Hammond
Department of Computer Science / Future of Humanity Institute, University of Oxford. E-mail: lewis.hammond@cs.ox.ac.uk

P. Harrenstein
Department of Computer Science, University of Oxford. E-mail: paul.harrenstein@cs.ox.ac.uk

M. Kwiatkowska
Department of Computer Science, University of Oxford. E-mail: marta@cs.ox.ac.uk

M. Najib
Department of Computer Science, University of Kaiserslautern. E-mail: najib@cs.uni-kl.de

G. Perelli
Department of Computer, Control, and Management Engineering, Sapienza University of Rome. E-mail: perelli@diag.uniroma1.it

T. Steeples
Department of Computer Science, University of Oxford. E-mail: thomas.steeples@cs.ox.ac.uk

M. Wooldridge
Department of Computer Science, University of Oxford. E-mail: mjw@cs.ox.ac.uk

Abstract We provide a survey of the state of the art of *rational verification*: the problem of checking whether a given temporal logic formula φ is satisfied in some or all game theoretic equilibrium computations of a multi-agent system – that is, whether the system will exhibit the behaviour φ represents under the assumption that agents within the system act rationally in pursuit of their preferences. After motivating and introducing the overall framework of rational verification, we discuss key results obtained in the past few years as well as relevant related work in logic, AI, and Computer Science.

Keywords automated verification · game theory · multi-agent systems · model checking · automated synthesis

1 Introduction

The deployment of AI technologies in a wide range of application areas over the past decade has brought the problem of *verifying* such systems into sharp focus. Verification is one of the most important and widely-studied problems in computer science [14]. Verification is the problem of checking program correctness: the key decision problem relating to verification is that of establishing whether or not a given system P satisfies a given specification φ. The most successful contemporary approach to formal verification is model checking, in which an abstract, finite state model of the system of interest is represented as a Kripke structure (a labelled transition system), and the specification is represented as a temporal logic formula, the models of which are intended to correspond to "correct" behaviours of the system [30]. The verification process then reduces to establishing whether the specification formula is satisfied in the given Kripke structure, a process that can be efficiently automated in many settings of interest [27, 9].

In the present paper, we will be concerned with *multi-agent systems* [72, 81]. Software agents were originally proposed in the late 1980s, but it is only over the past decade that the software agent paradigm has been widely adopted. At the time of writing, software agents are ubiquitous: we have software agents in our phone (*e.g.*, Siri), processing requests online, automatically trading in global markets, controlling complex navigation systems (*e.g.*, those in self-driving cars), and even carrying out tasks on our behalf at home (*e.g.*, Alexa). Typically, these agents do not work in isolation: they may interact with humans or with other software agents. The field of multi-agent systems is concerned with understanding and engineering systems that have these characteristics.

Since agents are typically "owned" by different principals, there is no requirement or assumption that the preferences delegated to different agents are aligned in any way. It may be that their preferences are compatible, but it may equally be that preferences are in opposition. Game theory provides a natural and widely-adopted framework through which to understand systems with these properties, where participants pursue their preferences rationally and strategically [59], and this observation has prompted a huge body of research over the past decade, attempting to apply and adapt game theoretic techniques to the analysis of multi-agent systems [62, 72].

We are concerned with the question of how we should think about the issues of correctness and verification in multi-agent systems. We argue that in a multi-agent setting, it is appropriate to ask what behaviours the system will exhibit *under the assumption that agents act rationally in pursuit of their preferences*. We advance the paradigm of *rational verification* for multi-agent systems, as a counterpart to classical verification. Rational verification is concerned with establishing whether a given temporal logic formula φ is satisfied in some or all game theoretic equilibria of a multi-agent system – that is, whether the system will exhibit the behaviour represented by φ under the assumption that agents within the system act rationally in pursuit of their preferences/goals.

We begin by motivating our approach, describing in detail the issue of correctness and verification, and the hugely successful model checking paradigm for verification. We then discuss the question of what correctness means in the setting of multi-agent systems, and this leads us to introduce the paradigm of rational verification and equilibrium checking. We then survey a range of semantic models for rational verification, summarising the key complexity results known for these models, and then examine three key tools for rational verification. We conclude by surveying some active areas of current research.

2 Setting the Scene

The aim of this section is to explain how the concept of rational verification has emerged from various research trends in computer science and artificial intelligence, and how it differs from the conventional conception of verification.

Correctness and Formal Verification: The *correctness problem* has been one of the most widely studied problems in computer science over the past fifty years, and remains a topic of fundamental concern to the present day [14]. Broadly speaking, the correctness problem is concerned with checking that computer systems behave as their designer intends. Probably the most important problem studied within the correctness domain is that of *formal verification*. Formal verification is the problem of checking that a given computer program or system P is correct with respect to a given formal (i.e., mathematical) specification φ. We understand φ as a description of system behaviours that the designer judges to be acceptable – a program that guarantees to generate a behaviour as described in φ is deemed to correctly implement the specification φ.

A key insight, due to Amir Pnueli, is that *temporal logic* provides a suitable framework within with which to express formal specifications of reactive system behaviour [65]. Pnueli proposed Linear Temporal Logic (LTL) for expressing desirable properties of computations. LTL extends classical logic with tense operators $\mathbf{X}$ (“in the next state…”), $\mathbf{F}$ (“eventually…”), $\mathbf{G}$ (“always…”), and $\mathbf{U}$ (“…until…”) [30]. For example, the requirement that a system never enters a “crash” state can naturally be expressed in LTL by a formula $\mathbf{G}\neg \mathit{crash}$, where $\neg \mathit{crash}$ denotes the complement (negation) of the set of “crash” states (namely states associated with a label *crash*). If we let $[\![P]\!]$ denote the set of all possible computations that may be produced by the program P, and let $[\![\varphi]\!]$ denote the set of state sequences that satisfy the LTL formula

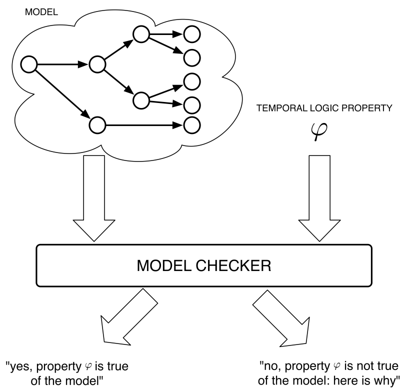

Fig. 1 Model checking. A model checker takes as input a model, representing a finite state abstraction of a system, together with a claim about the system behaviour, expressed in temporal logic. It then determines whether or not the claim is true of the model or not; most practical model checkers will provide a counter example if not.

φ, then verification of LTL properties reduces to the problem of checking whether $[\![P]\!] \subseteq [\![\varphi]\!]$. Another key temporal formalism is Computation Tree Logic (CTL), which modifies LTL by prefixing path formulae (which depend on temporal operators) with *path quantifiers* **A** ("on all paths...") and **E** ("on some path...") [30]. While LTL is suited to reasoning about runs or computational histories, CTL is suited to reasoning about states of transition systems that encode possible system behaviours.

Model Checking: The most successful approach to verification using temporal logic specifications is *model checking* [27]. Model checking starts from the idea that the behaviour of a finite state program P can be represented as a Kripke structure, or transition system K_P. Now, Kripke structures can be interpreted as models for temporal logic. So, checking whether P satisfies an LTL property φ reduces to the problem of checking whether φ is satisfied on paths through K_P. Checking a CTL specification φ is even simpler: the Kripke structure K_P is a CTL model, so we simply need to check whether $K_P \models \varphi$, which boils down to performing reachability analysis over the states of K_P. These checks can be efficiently automated for many cases of interest. In the case of CTL, for example, checking whether $K_P \models \varphi$ can be solved in time $O(|K_P| \cdot |\varphi|)$ [26, 30]; for LTL, the problem is more complex (PSPACE-complete [30]), but using automata theoretic techniques it can be solved in time $O(|K_P| \cdot 2^{|\varphi|})$ [79], the latter result indicating that such an approach is feasible for small specifications. Since the model checking paradigm was first proposed in 1981, huge progress has been made on extending the range of systems amenable to verification by model checking, and to extending the range of properties that might be checked [27].

Multi-agent systems: We now turn the class of systems that we will be concerned with in the present paper. The field of *multi-agent systems* is concerned with the theory and practice of systems containing multiple interacting semi-autonomous AI software components known as *agents* [81, 72]. Multi-agent systems are generally understood as distinct from conventional distributed or concurrent systems in several respects, but the most important distinction for our purposes is that different agents are assumed to be operating on behalf of different external principals, who delegate their preferences or goals to their agent. Because different agents are "owned" by different principals, there is no assumption that agents will have preferences that are aligned with each other.

Correctness in Multi-Agent Systems: Now, consider the following question:

> *How should we interpret correctness and formal verification in the context of multi-agent systems*?

In an uninteresting sense, this question is easily answered: We can certainly think of a multi-agent system as nothing more than a collection of interacting non-deterministic computer programs, with non-determinism representing the idea that agents have choices available to them; we can express such a system using any readily available model checking framework, which would then allow us to start reasoning about the possible computational behaviours that the system might in principle exhibit. But while such an analysis is entirely legitimate, and might well yield important insights, it is nevertheless missing a very big part of the story that is relevant in order to understand a multi-agent system. This is because *it ignores the fact that agents are assumed to pursue their preferences rationally and strategically*. Thus, certain system behaviours that might be possible *in principle* will never arise *in practice* because they could not arise from rational choices by agents within the system.

To take a specific example, consider eBay, the online auction house. When users create an auction on eBay, they must specify a deadline for bidding in the auction. This deadline, coupled with the strategic concerns of bidders, leads to a behaviour known as sniping [68]. Roughly, sniping is where bidders try to wait for the last possible moment to submit bids. Sniping is strategic behaviour, used by participants to try to get the best outcome possible. If we do not take into account preferences and strategic behaviour when designing a system like eBay, then we will not be able to predict or understand behaviours like sniping.

The classical formulation of correctness does not naturally match the multi-agent system setting because there can be no single specification φ, against which the correctness of a multi-agent system is judged. Instead, *each agent within such a system carries its own specification*: an agent is judged to be correct if it acts rationally to achieve its delegated preferences or goals. So, what should replace the classical notion of correctness and verification in the context of multi-agent systems? We posit that *rational verification* and *equilibrium checking* provide a suitable framework.

Rational Verification and Equilibrium Checking: As many other researchers [62, 72] we believe that *game theory* provides an appropriate formal framework for the analysis of multi-agent systems. Originating within economics, game theory is essentially the theory of strategic interaction between self-interested entities [59]. While the mathematical framework of game theory was not developed specifically to study computational settings, it nevertheless seems that the toolkit of analytical concepts it provides can be adapted and applied to multi-agent settings. A game in the sense of game theory is usually understood as an abstract mathematical model of a situation in which self-interested players must make decisions. A game specifies the decision-makers in the game – the "players" and the choices available to these players (their *strategies*). For every combination of possible choices by players, the game also specifies what outcome will result, and each player has their own preferences over possible outcomes.

A key concern in game theory is to try to understand what the outcomes of a game can or should be, under the assumption that the players within it act rationally. To this end, a number of *solution concepts* have been proposed, of which *Nash equilibrium* is the most prominent. A Nash equilibrium is a collection of choices, one for each participant in the game, such that no player can benefit by unilaterally deviating from this combination of choices. Nash equilibria seem like reasonable candidates for the outcome of a game because to move away from a Nash equilibrium would result in some player being worse off – which would clearly not be rational. In general, it could be the case that a given game has no Nash equilibrium, or multiple Nash equilibria. Now, it should be easy to see how this general setup maps to the multi-agent systems setting: players map to the agents within the system, and each player's preferences are as defined in their delegated goals; the choices available to each player correspond to the possible courses of action that may be taken by each agent in the system. Outcomes will correspond to the computations or runs of the system, and agents will have preferences over these runs; they act to try and bring about their most preferred runs.

With this in mind, we believe it is natural to think of the following problem as a counterpart to model checking and classical verification. We are given a multi-agent system, and a temporal logic formula φ representing a property of interest. We then ask *whether φ would be satisfied in some run that would arise from a Nash equilibrium collection of choices by agents within the system*. We call this *equilibrium checking*, and refer to the general paradigm as *rational verification*.

3 Models for Rational Verification

3.1 An Abstract Model

Let us make our discussion a little more formal with some suggestive notation (we present some concrete models in later sections). Let $P_1,\dots,P_n$ be the agents within a multi-agent system. For now, we do not impose any specific model for agents P_i: we will simply assume that agents are non-deterministic reactive programs. Non-determinism captures the idea that agents have choices available to them, while reactivity implies that agents are non-terminating. The framework we describe below can easily be applied to any number of computational models, including, for example, concurrent games [5], event structures [80], interpreted systems [32], or multi-agent planning systems [15].

A *strategy* for an agent P_i is a rule that defines how the agent makes choices over time. Each possible strategy for an agent P_i defines one way that the agent can resolve its non-determinism. We can think of a strategy as a function from the history of the system to date to the choices available to the agent in the present moment. We denote the possible strategies available to agent P_i by $\Sigma(P_i)$. The basic task of an agent P_i is to select an element of $\Sigma(P_i)$ – we will see later that agents select strategies in an attempt to bring about their preferences. When each agent P_i has selected a strategy, we have a profile of strategies $\vec{\sigma} = (\sigma_1, \dots, \sigma_n)$, one for each agent. This profile of strategies will collectively define the behaviour of the overall system. For now, we will assume that strategies are themselves deterministic, and that a collection of strategies therefore induces a unique run of the system, which we denote by $\rho(\sigma_1, \dots, \sigma_n)$. The set $R(P_1, \dots, P_n)$ of all possible runs of $P_1, \dots, P_n$ is:

$$R(P_1, \dots, P_n) = \{\rho(\vec{\sigma}) : \vec{\sigma} \in \Sigma(P_1) \times \cdots \times \Sigma(P_n)\}.$$

Where the strategies that lead to a run do not need to be named, we will denote elements of $R(P_1, \dots, P_n)$ by ρ, ρ', etc. Returning to our earlier discussion, we typically use Linear Temporal Logic as a language for expressing properties of runs: we will write $\rho \models \varphi$ to mean that run ρ satisfies temporal formula φ.

Before proceeding, we state a version of the conventional model checking problem for our setting:

MODEL CHECKING:
Given: System $P_1, \dots, P_n$; temporal formula φ.
Question: Is it the case that $\exists \vec{\sigma} \in \Sigma(P_1) \times \cdots \times \Sigma(P_n) : \rho(\vec{\sigma}) \models \varphi$?

This decision problem amounts to asking whether $\exists \rho \in R(P_1, \dots, P_n)$ such that $\rho \models \varphi$, that is, *whether there is any possible computation of the system that satisfies* φ, that is, *whether the system could* in principle *exhibit the behaviour* φ.

Preferences: So far, we have said nothing about the idea that agents act rationally in pursuit of delegated preferences. We assume that *agents have preferences over runs of the system*. Thus, given two possible runs $\rho_1, \rho_2 \in R(P_1, \dots, P_n)$, it may be that P_i prefers ρ_1 over ρ_2, or that it prefers ρ_2 over ρ_1, or that it is indifferent between the two. We represent preferences by assigning to each player P_i a relation $\succeq_i \subseteq R(P_1, \dots, P_n) \times R(P_1, \dots, P_n)$, requiring that this relation is complete, reflexive, and transitive. Thus $\rho_1 \succeq_i \rho_2$ means that P_i prefers ρ_1 at least as much as ρ_2. We denote the irreflexive sub-relation of $\succeq_i$ by $\succ_i$, so $\rho_1 \succ_i \rho_2$ means that P_i *strictly* prefers ρ_1 over ρ_2. Indifference (where we have both $\rho_1 \succeq_i \rho_2$ and $\rho_2 \succeq_i \rho_1$) is denoted by $\rho_1 \sim_i \rho_2$. We refer to a structure $M = (P_1, \dots, P_n, \succeq_1, \dots, \succeq_n)$ as a multi-agent system.

Alert readers will have noted that, if runs are infinite, then so are preference relations over such runs. This raises the issue of finite and succinct representations of runs. Several approaches to this issue have been suggested. The most obvious is to assign each agent P_i a temporal logic formula γ_i representing its *goal*. The idea is that P_i prefers all runs that satisfy γ_i over all those that do not, is indifferent between all runs that satisfy γ_i, and is similarly indifferent between runs that do not satisfy γ_i. Formally, the preference relation $\succeq_i$ corresponding to a goal γ_i is defined as follows:

$$\rho_1 \succeq_i \rho_2 \quad \text{iff} \quad \rho_2 \models \gamma_i \text{ implies } \rho_1 \models \gamma_i.$$

We discuss alternative preference models in section 5.2.

Nash equilibrium: With this definition, we can now define the standard game theoretic concept of Nash equilibrium for our setting. Let $M = (P_1, \dots, P_n, \succeq_1, \dots, \succeq_n)$ be a multi-agent system, and let $\vec{\sigma} = (\sigma_1, \dots, \sigma_i, \dots, \sigma_n)$ be a strategy profile. Then we

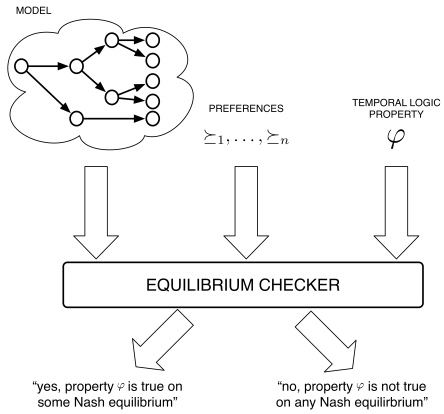

Fig. 2 Equilibrium checking. The key difference to model checking is that we also take as input the preferences of each of the system components, and the key question asked is whether or not the temporal property φ holds on some/all equilibria of the system.

say $\vec{\sigma}$ is a Nash equilibrium of M if for all players P_i and for all strategies $\sigma_i' \in \Sigma(P_i)$, we have:

$$\rho(\vec{\sigma}) \succeq_i \rho(\sigma_1, \ldots, \sigma_i', \ldots, \sigma_n).$$

Let $NE(M)$ denote the set of all Nash equilibria of M. Of course, many other solution concepts have been proposed in the game theory literature [59] – to keep things simple, in this paper we will restrict our attention to Nash equilibrium.

Equilibrium Checking: We are now in a position to introduce equilibrium checking, and the associated key decision problems. The basic idea of equilibrium checking is that, instead of asking whether a given temporal formula φ is satisfied on some possible run of the system, we instead ask whether it is satisfied on some run corresponding to a Nash equilibrium of the system. Informally, we can understand this as asking whether φ could be made true as the result of rational choices by agents within the system. This idea is captured in the following decision problem (see Figure 2):

> <u>E-NASH:</u>
> *Given*: Multi-agent system M; temporal formula φ.
> *Question*: Is it the case that $\exists \vec{\sigma} \in NE(M) : \rho(\vec{\sigma}) \models \varphi$?

The obvious counterpart of this decision problem is A-NASH, which asks whether a temporal formula φ is satisfied on *all* Nash equilibrium outcomes.

> <u>A-NASH:</u>
> *Given*: Multi-agent system M; temporal formula φ.
> *Question*: Is it the case that $\forall \vec{\sigma} \in NE(M) : \rho(\vec{\sigma}) \models \varphi$?

A higher-level question is simply whether a system has *any* Nash equilibria:

> <u>NON-EMPTINESS:</u>
> *Given*: Multi-agent system M.
> *Question*: Is it the case that $NE(M) \neq \emptyset$?

A system without any Nash equilibria is inherently *unstable*: whatever collection of choices we might consider for the agents within it, some player would have preferred to make an alternative choice. Notice that an efficient algorithm for solving E-NASH would imply an efficient algorithm for NON-EMPTINESS.

Finally, we might consider the question of verifying whether a given strategy profile represents a Nash equilibrium:

IS-NE:
Given: Multi-agent system M, strategy profile $\vec{\sigma}$
Question: Is it the case that $\vec{\sigma} \in NE(M)$?

Recall that, mathematically, strategies are functions that take as input the history of the system to date, and give as output a choice for the agent in question. Since the computations generated by multi-agent systems will be infinitary objects, to study this decision problem we will need a finite representation for strategies. A common approach is to use *finite state machines with outputs* (e.g., Moore machines).

3.2 Iterated Boolean Games

A simple and elegant concrete computational model that we have found useful to explore questions surrounding rational verification is the framework of *iterated Boolean games* (iBGs) [37]. In an iBG, each agent P_i is defined by associating it with a finite, non-empty set of Boolean variables Φ_i, and preferences for P_i are specified with an LTL formula γ_i. It is assumed that each propositional variable is associated with a single agent. The choices available to P_i at any given point in the game then represent the set of all possible assignments of truth or falsity to the variables under the control of P_i. An iBG is "played" over an infinite sequence of rounds; in each round every player independently selects a valuation for their variables, and the infinite run traced out in this way thus defines an LTL model, which will either satisfy or fail to satisfy each player's goal. In iBGs, strategies are represented as finite state machines with output (Moore machines). This may seem like a limitation, but in fact it is not: in the setting of iBGs, finite state machine strategies are all that is required.

Let us now turn to the decision problems that we identified above, and consider their complexity in the iBG case. Before we state the complexity of these problems, it is worth recalling a special case of iBGs, which was first studied in the 1980s by Pnueli and Rosner [66]. A *LTL synthesis problem* is a setting defined by two players, often denoted A and E, two disjoint sets of propositional variables, Φ_E and Φ_A, and an LTL formula defined over the variables $\Phi_E \cup \Phi_A$. The setting is interpreted as a game in the following way: the play continues for an infinite sequence of rounds, where in each round the players simultaneously choose a valuation for their respective variable set. In this way, the play traces out a word in $(\Phi_E \cup \Phi_A)^{\omega}$, and this word can be understood as an LTL valuation. Player E wins if this valuation satisfies φ, and loses otherwise. The LTL synthesis problem is then as follows:

LTL SYNTHESIS:
Given: Variables Φ_E and Φ_A, and LTL formula φ.
Question: Can E force a win in the game induced by Φ_E, Φ_A, φ? That is, does

there exists a strategy σ_E for E such that for all strategies σ_A for A, we have $\rho(\sigma_E, \sigma_A) \models \varphi$?

The LTL synthesis problem was introduced to study the problem of software settings in which we want to know whether a particular software component (represented by E in this case) can ensure that an overall system objective (φ) is satisfied in the presence of arbitrary, or adversarial input from the software environment (A). In game theoretic terms, LTL synthesis is a two player, strictly competitive win-lose game, and it can be seen as a special case of iBGs: we can model LTL synthesis in an iBG by assigning player E the goal φ and A the goal $\neg\varphi$. Now, the central result proved by Pnueli and Rosner was this:

Theorem 1 ([66]) *The LTL synthesis problem is 2EXPTIME-complete.*

Observe that this is an extremely negative result, considerably worse than (for example) the PSPACE-complete LTL model checking problem [73]. The high complexity derives from the fact that the LTL synthesis problem requires quantifying over strategies for satisfying LTL formulae: checking Nash equilibrium properties of iBGs requires similar quantification, and it should therefore come as no surprise that iBGs inherit the high complexity of LTL synthesis.

Theorem 2 ([37]) *For iBGs,* IS-NE *is PSPACE-complete (and hence no easier or harder than model checking or satisfiability for LTL). In contrast,* NON-EMPTINESS*,* E-NASH*, and* A-NASH *are all 2EXPTIME-complete.*

It is not hard to see the close relationship between these problems and LTL synthesis. For example, we can immediately see that A-NASH is 2EXPTIME hard from the following reduction: given an instance $(\Phi_E, \Phi_A, \varphi_E)$ of LTL synthesis, construct an iBG with players $\{E, A\}$, and propositional control sets as in the LTL synthesis instance, with goals for the players being φ_E and $\neg\varphi_E$ respectively. Then ask whether φ_E is satisfied on all Nash equilibrium runs of the game. It is straightforward to see that E has a winning strategy for φ_E if and only if φ_E is satisfied on all Nash equilibrium computations.

Although it may seem rather abstract, the iBG framework is quite general, and more widely applicable than it might at first appear. For example, frameworks in which agent programs P_i can be axiomatized in LTL can be expressed in iBGs – see [36] for details.

One fascinating aspect of the development of the theory for iBGs is that, when understanding the equilibrium properties of iBGs, we can make use of the *Nash folk theorems* – classic results in game theory which relate to the equilibrium properties that can be sustained in iterated games [59]. It is remarkable that a proof technique developed in the 1950s to study an abstract class of games turns out to be directly applicable to the verification of AI systems 70 years later: see [37] for details.

3.3 Concurrent Game Structures

Concurrent Game Structures are a widely-used model for concurrent and multi-agent systems [5]. In this model, say M, typically presented in its deterministic form, there are N players who, at each state s, make an independent choice a_i, with $i \in N$, which jointly define an action profile $\vec{a} = (a_1, \ldots, a_{|N|})$ that uniquely determines the next state s', that is, a unique transition $(s, \vec{a}, s')$ in M. Formally, a Concurrent Game Structure is given by a tuple:

$$M = (N, S, s^0, (A_i)_{i \in N}, \delta),$$

where, N and S are finite, non-empty sets of agents and system states, respectively, where $s^0 \in S$ is an initial state; A_i is a set of actions available to agent i, for each i; $\delta : S \times A_1 \times \cdots \times A_{|N|} \to S$ is a transition function. Concurrent games are played as follows. The game begins in state s^0, and each player $i \in N$ simultaneously picks an action $a_i^0 \in A_i$. The game then transitions to a new state, $s^1 = \delta(s^0, a_1^0, \ldots, a_{|N|}^0)$, and this process repeats. Thus, the n^{th} state transitioned to is $s^n = \delta(s^{n-1}, a_1^{n-1}, \ldots, a_{|N|}^{n-1})$. Since the transition function is deterministic, a play of a game will be an infinite sequence of states, denoted by π. Such a sequence of states is called a *run*.

Thus, to play a game, agents use strategies, which are formally defined as functions from sequences of states to next states. Because Concurrent Game Structures are deterministic, a profile of strategies for all agents $\vec{f} = (f_1, \ldots, f_{|N|})$ determines a unique run in M, denoted by $\pi(\vec{f})$. Assuming that agents have a preference relation $\geq_i$, with $i \in N$, over the set of runs in M, one can immediately define further game-theoretic concepts, such as the stable outcomes, runs, or profiles of a game. For instance, in case of Nash equilibrium, we say that a strategy profile $\vec{f} = (f_1, \ldots, f_{|N|})$ is a Nash equilibrium if, for each agent i and every strategy f_i' of i we have:

$$\pi(\vec{f}) \geq_i \pi(f_1, \ldots, f_i', \ldots, f_{|N|}),$$

that is, agent i does not prefer the run induced by $(f_1, \ldots, f_i', \ldots, f_{|N|})$ over the run induced by $\vec{f} = (f_1, \ldots, f_i, \ldots, f_{|N|})$, which we call a Nash equilibrium run.

3.4 Reactive Module Games

While concurrent games provide a natural semantic framework for multi-agent systems, they are not directly appropriate as a modelling framework to be used by people. For this, the framework of *Reactive Module Games* is more appropriate [39]. Within this framework, concurrent games are modelled using the *Simple Reactive Modules Language* (SRML) [77], a simplified version of the *Reactive Modules* language that is widely used within the model checking community [3].

The basic idea is that each system component (agent/player) in SRML is represented as a *module*, which consists of an *interface* that defines the name of the module and lists a non-empty set of Boolean variables controlled by the module, and a set of *guarded commands*, which define the choices available to the module at each state. There are two kinds of guarded commands: **init**, used for initialising the variables, and **update**, used for updating variables subsequently.

A guarded command has two parts: a "condition" part (the "guard") and an "action" part. The "guard" determines whether a guarded command can be executed or

module *toggle* **controls** x
  init
  $:: \top \rightsquigarrow x' := \top;$
  $:: \top \rightsquigarrow x' := \bot;$
  update
  $:: \neg x \rightsquigarrow x' := \top;$
  $:: x \rightsquigarrow x' := \bot;$

Fig. 3 Example of module toggle in SRML.

not given the current state, while the "action" part defines how to update the value of (some of) the variables controlled by a corresponding module. Intuitively, $\varphi \rightsquigarrow \alpha$ can be read as "if the condition φ is satisfied, then *one* of the choices available to the module is to execute α". Note that the value of φ being true does not guarantee the execution of α, but only that it is *enabled* for execution, and thus *may be chosen*. If no guarded command of a module is enabled in some state, then that module has no choice and the values of the variables controlled by it remain unchanged in the next state. More formally, a guarded command g over a set of variables Φ is an expression

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

where the guard φ is a propositional logic formula over Φ, each x_i is a member of Φ and ψ_i is a propositional logic formula over Φ. It is required that no variable x_i appears on the left hand side of more than one assignment statements in the same guarded command, hence no issue on the (potentially) conflicting updates arises.

Here is a concrete example of a guarded command:

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

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

Formally, an SRML module m_i is defined as a triple $m_i = (\Phi_i, I_i, U_i)$, where $\Phi_i \subseteq \Phi$ is the finite set of Boolean variables controlled by m_i, I_i a finite set of **init** guarded commands, and U_i a finite set of **update** guarded commands. As in iBGs, it is required that variables are controlled by exactly one agent.

Figure 3 shows a module named *toggle* that controls a single Boolean variable, named x. There are two **init** guarded commands and two **update** guarded commands. The **init** guarded commands define two choices for the initialisation of variable x: true or false. The first **update** guarded command says that if x has the value of true, then the corresponding choice is to assign it to false, while the second command says that if x has the value of false, then it can be assigned to true. Intuitively, the module would choose (in a non-deterministic manner) an initial value for x, and then on subsequent rounds toggles this value. In this particular example, the **init** commands are non-deterministic, while the **update** commands are deterministic. We refer to [39] for further details on the semantics of SRML. In particular, in Figure 12 of [39], we detail how to build a Kripke structure that models the behaviour of an SRML system.

Module definitions allow us to represent the possible actions of individual agents, and the effects of their actions, but do not represent preferences. In RMGs, preferences are captured by associating each module with a goal, which is specified as a temporal logic formula. Given this, a reactive module game is given by a structure $G = (N, m_1, \ldots, m_n, \gamma_1, \ldots, \gamma_n)$, where $N = \{1, \ldots, n\}$ is the set of agents, m_i is the module defining the choices available to agent i, as explained above, and γ_i is the goal of player i. In [39], two possibilities were considered for the language of goals γ_i: LTL and CTL. In the case of LTL, strategies σ_i for individual players are essentially the same as in iBGs: deterministic finite state machines with output. At each round of the game, a strategy σ_i chooses one of the enabled guarded commands to be executed. Because all strategies are deterministic, upon execution the collective strategies of all players will trace out a unique run, which will either satisfy or not satisfy each players goal, as in the case of iBGs. In the case of CTL, however, player strategies are non-deterministic: instead of selecting a single guarded command for execution, a strategy selects a set of guarded commands. The result of executing such strategies yields a tree structure, which will then either satisfy or fail to satisfy the CTL goals of players.

When it comes to the complexity of decision problems relating to RMGs, we find the following:

Theorem 3 ([39])

- *For LTL RMGs,* IS-NE *is PSPACE-complete, while* E-NASH *and* A-NASH *are both 2EXPTIME-complete.*
- *For CTL RMGs,* IS-NE *is EXPTIME-complete, while* E-NASH *and* A-NASH *are both 2EXPTIME-hard.*

The key conclusion relating to these results is that, despite the naturalness and expressive power of RMGs, computationally they are no more complex than iBGs. The high complexity of the key decision problems relating to RMGs indicates that naive algorithms to solve them will be hopelessly impractical: specialised techniques are required. In section 4.1, we will describe such techniques, and a system implemented based upon them.

3.5 Markov Games

Markov Games, also known as Concurrent Stochastic Games (sometimes simply Stochastic Games), are a popular representation of (simultaneous) multi-agent decision-making scenarios with stochastic dynamics. In this latter respect they differ from Concurrent Games, as discussed above, in which environments are assumed to be deterministic. They naturally generalise both Markov Decision Processes (a Markov Game with one player) and Iterated Normal-Form Games (a Markov Game with one state). Such games proceed at each time-step, from some state s, by each agent P_i using their strategy σ_i to select an action a_i, leading to a joint action $\vec{a} = (a_1, \ldots, a_n)$.

The next state s' is then drawn from the conditional probability distribution given by a Markovian transition function $T(s' \mid s, \vec{a})$. The strategy profile $\vec{\sigma}$ and the transition dynamics thus define a Markov Chain over the states S of the game, leading to a distribution $\Pr_{\vec{\sigma}}(\rho)$ over runs $\rho = s_0 s_1 s_2 \ldots$ through the state space.

On top of this underlying game structure one may then define different forms of objective for each of the agents. Common examples include the expected cumulative discounted reward:

$$\mathbb{E}_{\vec{\sigma}}[\sum_{t=0}^{\infty} \beta^t r^i_{t+1} \mid s_0 = s] I(s)$$

and the expected mean-payoff reward:

$$\lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{\vec{\sigma}}[\sum_{t=0}^{T} r^i_{t+1} \mid s_0 = s] I(s).$$

Here, $\beta \in [0,1)$ is a discount factor, $r^i_{t+1} \in \mathbb{R}$ is the reward given to agent i at time $t+1$, and $I(s)$ is an initial state distribution. Alternatively, for any set of runs $R' \subseteq R(P_1, \ldots, P_n)$ we may define an indicator random variable $X_{R'}$ such that $X_{R'}(\rho) = 1$ if $\rho \in R'$ and $X_{R'}(\rho) = 0$ otherwise. A player's reward can then be defined as the expected value $\mathbb{E}_{\vec{\sigma}}[X_{R'}]$ of this variable. For example, we could consider the probability of satisfying a temporal logic formula γ_i by defining R' as containing all and only those runs ρ such that $\rho \models \gamma_i$.

The introduction of stochastic dynamics also introduces different 'ways of winning' when we have Boolean objectives that are either satisfied or not by a particular path [28]. For example, a player may win by satisfying their goal γ_i *surely* (with certainty), *almost surely* (with probability one), *limit surely* (with probability greater than $1 - \varepsilon$ for every $\varepsilon > 0$), *boundedly* (with probability bounded away from one), *positively* (with positive probability), or *existentially* (possibly). Aside from these qualitative conditions, players may be interested in simply maximising the probability that their goal γ_i is achieved. Such a perspective can also be carried over to the problem of rational verification, in which we may be interested in the sure, almost sure, or limit sure satisfaction of a property φ, or simply in the probability that φ is satisfied.

4 Tools

While synthesis problems (such as the LTL synthesis problem, introduced by Pnueli and Rosner and discussed above) have been increasingly studied within the verification community, rational verification has come to prominence only in the past few years. As such, relatively few software tools exist for this problem. Below, we briefly survey some of the most widely used.

4.1 EVE: The Equilibrum Verification Environment

As we noted above, the high complexity of rational verification for RMGs (see above) indicates that naive algorithms for this purpose will be doomed to failure, even for systems of very moderate size. It follows that any practical system will require sophisticated algorithmic techniques. The *Equilibrium Verification Environment* is a system based on such techniques [43, 45].

The basic approach embodied by EVE involves reducing rational verification to a collection of parity games [31], which are widely used for synthesis and verification problems. A parity game is a two-player zero-sum turn-based game given by a labelled finite graph $H = (V_0, V_1, E, \alpha)$ such that $V = V_0 \cup V_1$ is a set of states partitioned into Player 0 (V_0) and Player 1 (V_1) states, respectively, $E \subseteq V \times V$ is a set of edges/transitions, and $\alpha : V \to \mathbb{N}$ is a labelling priority function. Player 0 wins if the smallest priority that occurs infinitely often in the infinite play is even. Otherwise, player 1 wins. It is known that solving a parity game (checking which player has a winning strategy) is in NP $\cap$ coNP [49], and can be solved in quasi-polynomial time [17] [1].

The algorithm underpinning EVE uses parity games in the following way. It takes as input an RMG M and builds a parity game H whose sets of states and transitions are doubly exponential in the size of the input but with priority function only exponential in the size of the input game. Using a deterministic Streett automaton on infinite words (DSW) [50], we then solve the parity game, leading to a decision procedure that is, overall, in 2EXPTIME, and, therefore, given the hardness results we mentioned above, essentially optimal. The EVE system can: (i) solve the E-NASH and A-NASH problems for the given RMG; and (ii) synthesise individual player strategies in the game.

Experimental results show that EVE performs favourably compared with other existing tools that support rational verification.

4.2 PRISM-games

A separate though closely related thread of research into the verification of multi-agent systems has emerged from the probabilistic model-checking community. The most prominent example of this in recent years is the expansion of PRISM [52], a popular tool for probabilistic model-checking, to handle first Turn-Based [11] and now Concurrent Stochastic Games (Markov Games) [53, 54]. Earlier work was limited to non-cooperative turn-based or zero-sum concurrent settings. Later efforts considering cooperative, concurrent games were initially restricted to those with only two coalitions, but this restriction has been partially lifted in the most recent instantiation of the work, which supports model-checking of arbitrary numbers of coalitions in the special case of stopping games – those in which eventually, with probability one, the outcome of each player's objective becomes fixed [54]. We note further that the current version of the tool also supports the use of Probabilistic Timed Automata in verifying Turn-Based Markov Games with real-valued clocks [55].

In PRISM-games, specifications are expressed in rPATL, probabilistic ATL (a generalisation of CTL that uses an extra quantifier $\langle\langle A \rangle\rangle \varphi$ for reasoning about properties φ that that be ensured by some subset A of the agents [5]) with rewards [24]. The logic

[1] Despite more than 30 years of research, and promising practical performance for algorithms to solve them, it remains unknown whether parity games can be solved in polynomial time.

is then further extended in order to be able to reason about equilibria in the game (in particular, subgame-perfect social-welfare optimal Nash equilibria). For example, this allows one to answer not only queries such as $\langle\langle P_1 \rangle\rangle_{max\geq 0.5}(\Pr[\psi])$ – is it the case that P_1 can ensure that ψ holds with at least probability a half? – but also queries such as $\langle\langle P_1 : P_2 \rangle\rangle_{max\geq 2}(\Pr[\psi] + \Pr[\chi])$ – is it the case that P_1 and P_2 can coordinate to ensure that both of their respective goals, ψ and χ, hold with probability one? – where ψ and χ are LTL formulae and similarly for expected rewards. More information can be found in [54]. An alternative specification formalism that can express equilibria concepts is Probabilistic Strategy Logic [8], but it has no associated implementation.

From a technical standpoint, PRISM-games also makes use of the Reactive Modules language with individual players represented by a set of modules which may then choose an enabled command at each time-step. On top of this users can include reward structures that produce real-valued rewards given a state and joint action as input, and define temporal logic properties expressed in the (extended version of) rPATL. For zero-sum properties PRISM-games relies on using value iteration to approximate values for all states of the game, and then solves a linear program for each state in order to compute a minimax strategy. For equilibria-based properties, a combination of backwards induction and value iteration are used, which is exact for finite-horizon and approximate for infinite-horizon properties, together with a sub-procedure for computing optimal Nash equilibria in n-player Normal-Form Games that makes use of SMT and non-linear optimisation engines.

4.3 MCMAS

MCMAS [56] adopts interpreted systems [32] as the formal language to represent systems comprised of multiple entities. In MCMAS, interpreted systems are extended to incorporate game theoretic notions such as those provided by ATL modalities [57]. The formalisation used to model systems in MCMAS can be thought of as a "bottom-up" approach, where the global state is defined as a tuple of the local states of the agents. In this setting, global states are given as the composition of local states of the agents and environment. MCMAS uses a dedicated programming language called *Interpreted Systems Programming Language* (ISPL) to describe the specification of IS.

There are different extensions of MCMAS that handle different specification logics. However, one particular extension that supports specification language expressive enough to reason about Nash equilibrium is MCMAS-SLK [19]. The tool's specification language is Strategy Logic with Knowledge (SLK) [18], an extension of the previously introduced Strategy Logic (SL) [60, 61]. Due to the undecidability result of the model-checking problem of multi-agent systems under perfect recall and incomplete information [4], the tool adopts imperfect recall semantics.

The problem NON-EMPTINESS can be solved using MCMAS by specifying the existence of Nash equilibrium with SLK. Let $N = \{1, \dots, n\}$ be the set of players in a game, Var be the set of strategy variables, and Γ be the set of goals of players in the game. Using SLK, we can express the existence of Nash equilibrium with the formula φ_{NE}:

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

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

Intuitively, formula φ_{NE} can be explained as follows: for each player i with its chosen strategy x_i in the game, if the goal of Player i cannot be achieved using strategy x_i then for every "alternative" strategy y_i, the goal of Player i cannot be achieved. This means that, players who do not get their goals achieved cannot benefit from unilaterally changing their strategies. Thus, if φ_{NE} is true in the given game, then there exists a Nash equilibrium in the game. The other problems of rational verification, namely E-NASH and A-NASH, can be reduced to NON-EMPTINESS [36].

5 Challenges

In this section, we provide a brief discussion of some current and future research challenges for rational verification.

5.1 Tackling Complexity

Perhaps the most obvious challenge in making rational verification an industrial-strength reality is that of the high computational complexity of the basic decision problems. Whilst LTL formulae are expressive and natural [78], and moreover, widely used in industry [21, 25, 69, 70], the 2EXPTIME-completeness results leave our problems grossly intractable. As such, it is important for us to consider other languages which strike a balance of complexity and expressiveness - how can we capture the richness of multi-agent systems, whilst still being able to reason about them effectively?

Perhaps the most obvious thing to try is to consider fragments of LTL. Various restrictions of LTL are very well studied [7, 74] and the decision problems relating to them are much more computationally amenable. In [37], the authors consider games where all the players have propositional safety goals – that is, LTL goals of the form $\mathbf{G}\varphi$, where φ is some propositional formula. In this setting, the E-NASH problem is PSPACE-complete. Additionally, in [44], the authors consider GR(1) [12] goals and specifications. Here, the E-NASH problem is PSPACE-complete with GR(1) goals and LTL specifications, and lies in FPT (fixed parameter tractable) [29] when both the goals and the specifications are in GR(1).

In addition to considering restricted languages for goals and temporal queries, a number of other directions suggest themselves as possible ways in which to reduce complexity, although we emphasise that we have no concrete results with these directions at this time. The first possibility is to consider ways in which games can be decomposed into smaller games, while preserving the relevant game theoretic properties. Similar techniques have been studied within the model checking community (see, e.g., [6]). Another possibility, also inspired by work within model checking, is to consider abstracting games to their smallest bisimulation-equivalent form. Care must be taken in this case, however, because we need to ensure that the precise form of bisimulation to be used must preserve Nash equilibria across bisimulation-equivalent models, and naive attempts to define bisimulation, which preserve temporal logic properties under model checking, do not necessarily preserve Nash equilibria – we refer the interested reader to [38] for details.

5.2 Alternative Preference Models

So what if we were to set aside temporal logics and consider different preference relations altogether? Staying in the qualitative mindset, in [13], the authors consider games where the players have ω-regular objectives and look at the NON-EMPTINESS problem, obtaining complexity results ranging from P-completeness all the way up to EXPTIME membership. Alternatively, one can adopt a quantitative approach and consider mean-payoff objectives - one can ask if there exists some Nash equilibrium where each player's payoff lies in a certain interval. As established in [75], this problem is NP-complete.

In order to be able to reason about games in a richer fashion, we can use quantitative and qualitative constructs in the same breath. If we look at games where the players' preferences are given by mean-payoff objectives, and we ask if there exists a Nash equilibrium which models an LTL specification, this problem is PSPACE-complete. Moreover, if we restrict our attention to $\mathrm{GR}(1)$ specifications, then we retain the NP-completeness result of the original mean-payoff NON-EMPTINESS problem. However, balancing qualitative and quantitative goals and specifications is not always as straightforward as this - for instance, in two-player, zero-sum, mean-payoff parity games [23], where the first player gets their mean-payoff if some parity condition is satisfied, and $-\infty$ otherwise, this same player may require infinite memory to act optimally. Thus, given the standard translation from non-deterministic Büchi automata to deterministic parity automata [64], this does not bode well for games with combined mean-payoff and LTL objectives - many of the techniques in rational verification depend on the existence of memoryless or finite-memory strategies in the corresponding two-player, zero-sum version of the game. Despite this, [42, 41] look at games with lexicographic preferences, where the first component is either a Büchi condition or an LTL formula, and the second component is some mean-payoff objective. Rather than considering the standard NON-EMPTINESS problem, they study a closely related analogue - the problem of whether or not there exists some finite-state, strict ε-Nash Equilibrium. These additional restrictions are brought about precisely due to the necessity of infinite memory in mean-payoff parity games, as mentioned above. When the first component is a Büchi condition, then the given decision problem is NP-complete, and in the LTL setting, it is 2EXPTIME-complete. Thus, despite the relaxation of the solution concept, we sadly do not see any gains in computational tractability.

Finally, some work has been to introduce non-dichotomous, qualitative preferences to rational verification. In [51], the authors introduce *Objective LTL* (OLTL) as a goal and specification format. An OLTL formula is simply a tuple of LTL formulae, along with a function with maps 0-1 tuples of the same length to integers. In a given execution of a game, some LTL formulae will be satisfied and others will not. Marking the ones that are satisfied with 1, and the ones which are not by 0, we can pass the resulting tuple into the given function and get an integer - each agent in the game wants to maximise this integer. With this preference model, we can look at games where there is a set of agents, plus a system player, and ask if there exists some strategy for the system player, along with a Nash equilibrium for the remaining players such that the system player's payoff is above a certain threshold. This problem is no harder than the original rational synthesis problem for LTL [35], being 2EXPTIME-complete. Building on this, in [2], the authors study rational verification with LTL$[\mathscr{F}]$ [1] goals and specifications. In short, LTL$[\mathscr{F}]$ generalises LTL by replacing the classical Boolean operators with arbitrary functions which map 0-1 tuples into the interval $[0,1]$. Again, the associated decision problem remains 2EXPTIME-complete.

5.3 Uncertain Environments

Thus far, the investigation into rational verification has focused largely on settings that are deterministic, discrete, fully observable, and fully known. Indeed this is sufficient for modelling a great many scenarios of interest, such as software processes or high-level representations of multi-agent control. Most of the real world, however, cannot be captured quite as neatly. This motivates the study of rational verification in *uncertain environments*, where this uncertainty might arise from stochastic dynamics, continuous or hybrid state and action spaces, or a structure that is only partially observable or partially known. Each of these features (and, moreover, their combination) represents an exciting direction for future work, the challenges of which we briefly outline here.

Perhaps the most natural and well-studied form of uncertainty in formal verification is of systems with *stochastic dynamics*. As noted above in Section 4.2, probabilistic model-checking techniques have recently been extended to the multi-agent setting by way of tools such as PRISM-games [55]. Preliminary work in the (limited) context of Markov Games with goals defined by the almost sure satisfaction of LTL properties suggests that the complexity classes of the main problems in both non-cooperative and cooperative rational verification remain essentially the same as in the non-stochastic setting. Further qualitative results for sure or limit sure winning (as well as for the quantitative case) are still to be obtained, however, and there remain many other interesting, open problems relating to ω-regular objectives in Markov Games [22].

In some situations, especially when considering cyber-physical systems, it is more appropriate to model the state space (and possibly the action space) as *continuous* or as *hybrid* – with some discrete and some continuous elements. Whilst not in itself necessarily introducing uncertainty, such representations bring challenges related to the concise encoding of system dynamics and agents' strategies over uncountable sets, and the careful definition of temporal logic formulae over paths through the state space. As well as modelling state or action spaces as continuous, one may also choose to represent time as being continuous, requiring new logics in which to encode specifications, such as Continuous-Time Stochastic Logic (CSL) [10] or Signal Temporal Logic (STL) [58].

When making a real-world decision in order to achieve a goal, it is rare to be able to observe all of the information relevant to that decision and goal. This intuition can be captured by models in which state space is only *partially observable* by the agents therein; in game-theoretic terms the agents have *imperfect information*. For example, Reactive Module Games in which each player may only observe a subset of the environmental variables are undecidable with three or more players, although the two-player case is solvable in 2EXPTIME [46]. Related work has explored the problem of rational synthesis in turn-based games under imperfect information (which is undecidable with three or more players and EXPTIME-complete for two players) [33], though the effects of partial observability on the rational verification problem remain under-explored.

Finally, there are scenarios in which larger portions of an environment are *unknown*, such as the transition dynamics, not only to the agents but also to those who wish to verify it. Here, traditional model-checking approaches do not apply and some form of learning must be introduced. As a result, different forms of guarantees about such systems are obtained, relying on assumptions about the structure of the environment and the theoretical characteristics of the learning algorithms used. Verification methods that employ learning have recently been developed by those in both the model-checking community [16] and the control and learning community [48], though few have considered the multi-agent setting and those that do restrict their attention to purely cooperative games [47]. A further complication is raised when agents *themselves* employ learning in unknown environments in order to update their strategies over time. With the continuing advance of machine learning, this is likely to become an increasingly common occurrence that requires new techniques for rational verification.

5.4 Cooperative Solution Concepts

Rational verification was first defined for noncooperative games [37, 39, 82]: players were assumed to act alone, and binding agreements between players were assumed to be impossible. As such, the solution concepts used in previous studies have therefore been noncooperative – primarily Nash equilibrium and refinements thereof.

However, in many real-life situations, these assumptions misrepresent reality. In order to address this issue, in [40], such the noncooperative setting for rational verification was extended to include *cooperative* solution concepts [59, 63]. That is, it was assumed, instead, that there is some (exogenous) mechanism through which agents in a system can reach binding agreements and form coalitions in order to collectively achieve goals. The possibility of binding cooperation and coalition formation eliminates some undesirable equilibria that arise in the noncooperative settings, and makes available a range of outcomes (*i.e.*, computations of the system that can be sustained in equilibrium) which cannot be achieved without cooperation.

In this new cooperative setting, the focus was on the *core*, arguably one of the most relevant solution concepts in the cooperative game theory literature. The basic idea behind the core is that a game outcome is said to be core-stable if no subset of agents could benefit by collectively deviating from it; the core of a game is the set of core-stable outcomes. Now, in conventional cooperative games (characteristic function games with transferable utility [20]), this intuition can be given a simple and natural formal definition, and as a consequence the core is probably the most widely-studied solution concept for cooperative games. However, the conventional definition of the core does not easily map into the rational verification framework as originally defined, mainly because coalitions are subject to *externalities*: whether or not a coalition has a beneficial deviation depends not just on the makeup of that coalition, but also *on the behaviour of the remaining agents* in the system.

Coalition formation with externalities has been extensively studied in the cooperative game theory literature [34, 76, 83], where different variants of the core can be found. For instance, α-core takes the pessimistic approach that requires that all members of a deviating coalition will benefit from the deviation regardless of the behaviour of the other coalitions that may be formed. Our main definition of the core precisely follows this approach. Even though coalition formation with externalities is common in and important for multi-agent systems [71], not much work has done regarding the problem of stability, and its properties, in multi-agent coalition formation with externalities. Instead, in AI and multi-agent systems, most research has focused on the structure formation problem itself [67]. Through our work on rational verification, we also address this gap in the literature of verification for AI systems.

The kinds of questions that are asked in the (rational verification) cooperative setting are exactly the same as in the noncooperative framework, only that instead of (variants of) Nash equilibrium one refers to outcomes in the core of game theoretic representations of multi-agent systems. Such questions, *e.g.*, E-CORE, A-CORE, etc., bearing the same meaning as their "Nash" counterparts, are all 2EXPTIME-complete [40] for games with LTL goals, but have some computationally desirable properties: the set of outcomes in the core is never empty, is bisimulation invariant [38], and has an elegant formalisation in ATL* [5], which makes the automated solution of cooperative rational verification problems possible in practice using verification tools for multi-agent systems analysis, such as MCMAS or EVE, described before.

6 Conclusions

Rational verification is a recent approach to the automated verification of multi-agent systems, in which we aim to automatically determine whether given properties of a system, expressed as temporal logic formulae, will hold in that system under the assumption that system components (agent) behave rationally, by choosing (for example) strategies that form a game theoretic equilibrium. Rational verification can be understood as a counterpart to the conventional model checking paradigm for automated verification. Although research in this area is at an early stage, the basic computational, logical, and algorithmic territory relating to rational verification has already been explored, and is described in the present article. An overarching goal for the future will be to make tools more practically applicable, and to understand the fundamental limitations of the paradigm. We have sketched out some of the key challenges that must be overcome to make this a reality: chief among them being dealing with complexity, broader preference models, richer modelling frameworks, and a wider range of game theoretic solution concepts.

References

1. Shaull Almagor, Udi Boker, and Orna Kupferman. Formalizing and reasoning about quality. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2013.
2. Shaull Almagor, Orna Kupferman, and Giuseppe Perelli. Synthesis of controllable nash equilibria in quantitative objective games. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pages 35–41. AAAI Press, 2018.
3. R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(11):7–48, July 1999.
4. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 100–109, Florida, October 1997.
5. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, September 2002.
6. R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer-Verlag, Berlin, Germany, 1998.
7. Rajeev Alur and Salvatore La Torre. Deterministic generators and games for ltl fragments. *ACM Transactions on Computational Logic (TOCL)*, 5(1):1–25, 2004.
8. Benjamin Aminof, Marta Kwiatkowska, Bastien Maubert, Aniello Murano, and Sasha Rubin. Probabilistic strategy logic. In *In Proc. International Joint Conference on Artificial Intelligence (IJCAI-19)*, 2019.
9. C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press: Cambridge, MA, 2008.
10. Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model checking continuous-time markov chains by transient analysis. In *Computer Aided Verification*, pages 358–372. Springer Berlin Heidelberg, 2000.
11. Nicolas Basset, Marta Kwiatkowska, Ufuk Topcu, and Clemens Wiltsche. Strategy synthesis for stochastic games with multiple long-run objectives. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 256–271. Springer Berlin Heidelberg, 2015.
12. Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.
13. P. Bouyer, R. Brenguier, N. Markey, and M. Ummels. Pure Nash equilibria in concurrent deterministic games. *Logical Methods in Computer Science*, 2015.

14. R. S. Boyer and J. S. Moore, editors. *The Correctness Problem in Computer Science*. The Academic Press: London, England, 1981.
15. R. Brafman and C. Domshlak. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence*, 198:52–71, 2013.
16. Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Kwiatkowska, David Parker, and Mateusz Ujma. Verification of markov decision processes using learning algorithms. In *Automated Technology for Verification and Analysis*, pages 98–114. Springer International Publishing, 2014.
17. C. S. Calude, S. Jain, B. Khoussainov, W. Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *STOC*, pages 252–263. ACM, 2017.
18. Petr Čermák, Alessio Lomuscio, Fabio Mogavero, and Aniello Murano. Mcmas-slk: A model checker for the verification of strategy logic specifications. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, pages 525–532, Cham, 2014. Springer International Publishing.
19. Petr Cermák, Alessio Lomuscio, Fabio Mogavero, and Aniello Murano. Practical verification of multi-agent systems against slk specifications. *Information and Computation*, 261(Part):588–614, 2018.
20. G. Chalkiadakis, E. Elkind, and M. Wooldridge. *Computational Aspects of Cooperative Game Theory*. Morgan-Claypool, 2011.
21. Toong Shoon Chan and Ian Gorton. Formal validation of a high performance error control protocol using spin. *Software: Practice and Experience*, 26(1):105–124, 1996.
22. Krishnendu Chatterjee and Thomas A. Henzinger. A survey of stochastic ω-regular games. *Journal of Computer and System Sciences*, 78(2):394–413, 2012.
23. Krishnendu Chatterjee, Thomas A Henzinger, and Marcin Jurdzinski. Mean-payoff parity games. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*, pages 178–187. IEEE, 2005.
24. Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.
25. Yunja Choi. From NuSMV to SPIN: Experiences with model checking flight guidance systems. *Formal Methods in System Design*, 30(3):199–216, January 2007.
26. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs — Proceedings 1981 (LNCS Volume 131)*, pages 52–71. Springer-Verlag: Berlin, Germany, 1981.
27. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press: Cambridge, MA, 2000.
28. Luca de Alfaro and Thomas A. Henzinger. Concurrent omega-regular games. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, LICS '00, page 141, USA, 2000. IEEE Computer Society.
29. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer New York, 1999.

30. E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, pages 996–1072. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.
31. E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *FOCS*, pages 368–377. IEEE, 1991.
32. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. The MIT Press: Cambridge, MA, 1995.
33. Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Rational synthesis under imperfect information. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, 2018.
34. Michael Finus and Bianca Rundshagen. A non-cooperative foundation of core-stability in positive externality ntu-coalition games. *Nota Di Lavoro 31.2003. Economics Energy Environment*, 2003.
35. D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *TACAS*, volume 6015 of *LNCS*, pages 190–204. Springer, 2010.
36. Tong Gao, Julian Gutierrez, and Michael J. Wooldridge. Iterated boolean games for rational verification. In Kate Larson, Michael Winikoff, Sanmay Das, and Edmund H. Durfee, editors, *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, S$\sim$ao Paulo, Brazil, May 8-12, 2017*, pages 705–713. ACM, 2017.
37. J. Gutierrez, P. Harrenstein, and M. Wooldridge. Iterated boolean games. *Information and Computation*, 242:53–79, 2015.
38. Julian Gutierrez, Paul Harrenstein, Giuseppe Perelli, and Michael J. Wooldridge. Nash equilibrium and bisimulation invariance. *Log. Methods Comput. Sci.*, 15(3), 2019.
39. Julian Gutierrez, Paul Harrenstein, and Michael J. Wooldridge. From model checking to equilibrium checking: Reactive modules for rational verification. *Artif. Intell.*, 248:123–157, 2017.
40. Julian Gutierrez, Sarit Kraus, and Michael J. Wooldridge. Cooperative concurrent games. In Edith Elkind, Manuela Veloso, Noa Agmon, and Matthew E. Taylor, editors, *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, pages 1198–1206. International Foundation for Autonomous Agents and Multiagent Systems, 2019.
41. Julian Gutierrez, Aniello Murano, Giuseppe Perelli, Sasha Rubin, Thomas Steeples, and Michael Wooldridge. Equilibria for games with combined qualitative and quantitative objectives. *Acta Informatica*, pages 1–26, 2020. Publisher: Springer.
42. Julian Gutierrez, Aniello Murano, Giuseppe Perelli, Sasha Rubin, and Michael Wooldridge. Nash equilibria in concurrent games with lexicographic preferences. 2017. Publisher: Association for the Advancement of Artificial Intelligence.
43. Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael Wooldridge. Eve: A tool for temporal equilibrium analysis. In *ATVA*, Vol 11138 of LNCS, pages 551–557, Cham, 2018. Springer.
44. Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. On Computational Tractability for Rational Verification. In Sarit Kraus, editor,

Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019, pages 329–335. ijcai.org, 2019.
45. Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. Automated temporal equilibrium analysis: Verification and synthesis of multi-player games. *Artif. Intell.*, 287:103353, 2020.
46. Julian Gutierrez, Giuseppe Perelli, and Michael Wooldridge. Imperfect information in reactive modules games. *Information and Computation*, 261:650 – 675, 2018. 4th International Workshop on Strategic Reasoning (SR 2016).
47. Lewis Hammond, Alessandro Abate, Julian Gutierrez, and Michael Wooldridge. Multi-agent reinforcement learning with temporal logic specifications. In *Proceedings of the 20th International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '21. International Foundation for Autonomous Agents and Multiagent Systems, 2021. Forthcoming.
48. Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning. *arXiv:1801.08099*, 2018.
49. M. Jurdzinski. Deciding the winner in parity games is in UP $\cap$ co-up. *Information Processing Letters*, 68(3):119–124, 1998.
50. Orna Kupferman. Automata theory and model checking. In *Handbook of Model Checking*, pages 107–151. Springer International Publishing, 2018.
51. Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016.
52. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
53. Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. Automated verification of concurrent stochastic games. In *Quantitative Evaluation of Systems*, pages 223–239. Springer International Publishing, 2018.
54. Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. Automatic verification of concurrent stochastic systems. *Formal Methods in System Design*, 2020. To appear.
55. Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. PRISM-games 3.0: Stochastic game verification with concurrency, equilibria and time. In *Computer Aided Verification*, pages 475–487. Springer International Publishing, 2020.
56. 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.
57. Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: an open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer*, 19(1):9–30, 2017.
58. Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer Berlin Heidelberg, 2004.

59. M. Maschler, E. Solan, and S. Zamir. *Game Theory*. Cambridge University Press: Cambridge, England, 2013.
60. Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. Reasoning about strategies: On the model-checking problem. *ACM Trans. Comput. Log.*, 15(4):34:1–34:47, 2014.
61. Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. Reasoning about strategies: on the satisfiability problem. *Log. Methods Comput. Sci.*, 13(1), 2017.
62. N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press: Cambridge, England, 2007.
63. M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press: Cambridge, MA, 1994.
64. Nir Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Log. Methods Comput. Sci.*, 3(3), 2007.
65. A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth IEEE Symposium on the Foundations of Computer Science*, pages 46–57, 1977.
66. A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programs*, 1989.
67. Talal Rahwan, Tomasz Michalak, Michael Wooldridge, and Nicholas R Jennings. Anytime coalition structure generation in multi-agent systems with positive or negative externalities. *Artificial Intelligence*, 186:95–122, 2012.
68. A. Roth and A. Ockenfels. Last-minute bidding and the rules for ending second-price auctions: Evidence from eBay and Amazon auctions on the internet. *American Economic Review*, 92(4):1093–1103, 2002.
69. Lawrence M. Ruane. Process synchronization in the UTS kernel. *Comput. Syst.*, 3(3):387–421, 1990.
70. Theo C. Ruys and Rom Langerak. Validation of bosch' mobile communication network architecture with spin. In *In Proceedings of SPIN97, the Third International Workshop on SPIN, University of Twente*, 1997.
71. Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial intelligence*, 101(1):165–200, 1998.
72. Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press: Cambridge, England, 2008.
73. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.
74. Jan Strejcek. *Linear temporal logic: Expressiveness and model checking*. PhD thesis, PhD thesis, Faculty of Informatics, Masaryk University in Brno, 2004.
75. Michael Ummels and Dominik Wojtczak. The Complexity of Nash Equilibria in Limit-Average Games. *CoRR*, abs/1109.6220, 2011.
76. Metin Uyanık. On the nonemptiness of the α-core of discontinuous games: Transferable and nontransferable utilities. *Journal of Economic Theory*, 158:213–231, 2015.
77. W. van der Hoek, A. Lomuscio, and M. Wooldridge. On the complexity of practical ATL model checking. In *Proceedings of the Fifth International Joint Confer-*

ence on Autonomous Agents and Multiagent Systems (AAMAS-2006), Hakodate, Japan, 2005.

78. M. Y. Vardi. Branching vs. linear time: Final showdown. In T. Margaria and W. Yi, editors, *Proceedings of the 2001 Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001 (LNCS Volume 2031)*, pages 1–22. Springer-Verlag: Berlin, Germany, April 2001.
79. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *First Symposium in Logic in Computer Science (LICS)*, 1986.
80. Glynn Winskel. Event structures. In *Advances in Petri Nets*, 1986.
81. M. Wooldridge. *An Introduction to Multiagent Systems (Second edition)*. John Wiley & Sons, 2009.
82. Michael J. Wooldridge, Julian Gutierrez, Paul Harrenstein, Enrico Marchioni, Giuseppe Perelli, and Alexis Toumi. Rational verification: From model checking to equilibrium checking. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 4184–4191. AAAI Press, 2016.
83. Sang-Seung Yi. Stable coalition structures with externalities. *Games and economic behavior*, 20(2):201–237, 1997.