Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Rational Verification: From Model Checking to Equilibrium Checking

Michael Wooldridge, Julian Gutierrez, Paul Harrenstein, Enrico Marchioni, Giuseppe Perelli, Alexis Toumi

Department of Computer Science, University of Oxford

Abstract

Rational verification is concerned with establishing whether a given temporal logic formula φ is satisfied in some or all equilibrium computations of a multi-agent system – that is, whether the system will exhibit the behaviour φ under the assumption that agents within the system act rationally in pursuit of their preferences. After motivating and introducing the framework of rational verification, we present formal models through which rational verification can be studied, and survey the complexity of key decision problems. We give an overview of a prototype software tool for rational verification, and conclude with a discussion and related work.

Introduction

Verification is one of the most important and widely-studied problems in computer science (Boyer and Moore 1981). 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 (Emerson 1990). 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 (Clarke, Grumberg, and Peled 2000).

In the present paper, we will be concerned with *multi-agent systems* (Shoham and Leyton-Brown 2008; Wooldridge 2009). Multi-agent systems are generally understood as systems composed of multiple interacting (semi-)autonomous software components known as agents, which act in pursuit of goals or preferences that are delegated to them by external principals. Since agents are "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 (Maschler, Solan, and Zamir 2013), 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 (Nisan et al. 2007; Shoham and Leyton-Brown 2008).

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 the classical view of correctness, outlined above, is not appropriate for multi-agent systems. In a multi-agent setting, it is more 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 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 introduce a formal framework, and survey the complexity of key decision problems associated with equilibrium checking – we see that the complexity of decision problems in equilibrium checking can be substantially worse than for conventional model checking problems. We then give an overview of a prototype software tool for rational verification: the EAGLE system takes as input a REACTIVE MODULES specification of a set of agents (Alur and Henzinger 1999), a goal for each agent, specified as a formula of the temporal logic CTL (Emerson 1990), and a collection of strategies, one for each agent in the system; the system then checks whether the strategies form a Nash equilibrium. We conclude with a discussion and related work.

Setting the Scene

Our aim in this section is to explain how the concept of rational verification emerged from various research trends in computer science and artificial intelligence, and how it differs from the conventional conception of verification. Readers will no doubt be familiar with some of this material – we beg their indulgence so that we can tell the story in its entirety.

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 (Boyer and Moore 1981). 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* can be a useful language with which to express formal specifications of system behaviour (Pnueli 1977). Pnueli proposed the use of 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…") (Emerson 1990). For example, the requirement that a system never enters a "crash" state can naturally be expressed in LTL by a formula $\mathbf{G}\neg crash$. 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 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 tense operators with *path quantifiers* $\mathbf{A}$ ("on all paths…") and $\mathbf{E}$ ("on some path…"). While LTL is suited to reasoning about runs or computational histories, CTL is suited to reasoning about transition systems that encode all possible system behaviours.

Model Checking: The most successful approach to verification using temporal logic specifications is *model checking* (Clarke, Grumberg, and Peled 2000). 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 some path 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 $K_P \models \varphi$. 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 $O(|K_P| \cdot |\varphi|)$ (Clarke and Emerson 1981; Emerson 1990); for LTL, the problem is more complex (PSPACE-complete (Emerson 1990)), but using automata theoretic techniques it can be solved in time $O(|K_P| \cdot 2^{|\varphi|})$ (Vardi and Wolper 1986), 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 (Clarke, Grumberg, and Peled 2000).

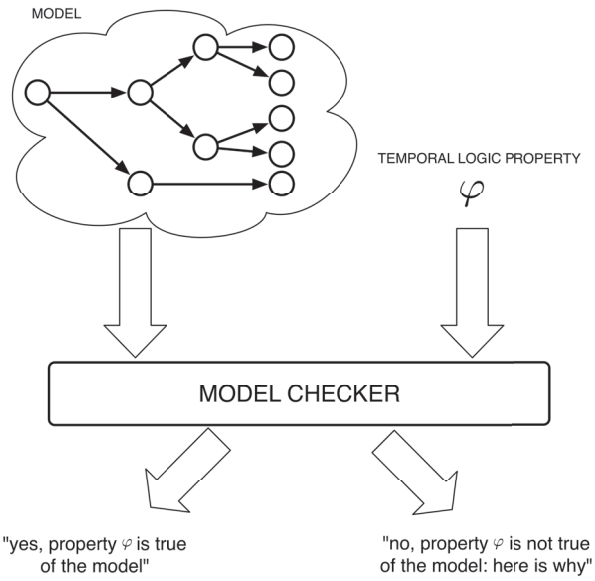

Figure 1: Model checking. A model checker takes as input a model, representing a finite state abstraction of a system, together with a claim about the system behaviour, expressed in temporal logic. It then determines whether or not the claim is true of the model or not; most practical model checkers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

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 software components known as *agents* (Wooldridge 2009; Shoham and Leyton-Brown 2008). 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 model such a system using any number of readily available model checking systems, 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 behaviour known as sniping (Roth and Ockenfels 2002). 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 is no single specification φ against which the correctness of a multi-agent system is judged. Instead, *the agents within such a system each carry their own specification*: an agent is judged to be correct if it acts rationally to achieve its delegated preferences or goals. There is no single standard of correctness by which the system as a whole can be judged, and attempting to apply such a standard does not help to understand the behaviour of the system. So, what should replace the classical notion of correctness in the context of multi-agent systems? We will now argue for the concept of *rational verification* and *equilibrium checking*.

Rational Verification and Equilibrium Checking: We believe, (as do many other researchers (Nisan et al. 2007; Shoham and Leyton-Brown 2008)), that *game theory* provides an appropriate analytical framework for the analysis of multi-agent systems. Originating within economics, game theory is essentially the theory of strategic interaction between self-interested entities (Maschler, Solan, and Zamir 2013). 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 widely used. 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*.

A Formal Model

Let us make our discussion a little more formal with some suggestive notation (we make the model and notation more precise in later sections). Let $P_1, \ldots, P_n$ 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 (Alur, Henzinger, and Kupferman 2002), event structures (Winskel 1986), interpreted systems (Fagin et al. 1995), or multi-agent planning systems (Brafman and Domshlak 2013).

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. 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 $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$, 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 $\rho(\sigma_1, \ldots, \sigma_n)$. The set $R(P_1, \ldots, P_n)$ of all possible runs of $P_1, \ldots, P_n$ is:

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

Where the strategies that lead to a run do not need to be named, we will denote elements of $R(P_1, \ldots, P_n)$ by ρ, ρ', etc. Returning to our earlier discussion, we use 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 present a version of the conventional model checking problem for our setting:

MODEL CHECKING:
Given: System $P_1, \ldots, P_n$; 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 $\exists \rho \in R(P_1, \ldots, P_n)$ 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 $\rho_1, \rho_2 \in R(P_1, \ldots, P_n)$, 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 $\succeq_i \subseteq R(P_1, \ldots, P_n) \times R(P_1, \ldots, P_n)$, 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 $M = (P_1, \ldots, P_n, \succeq_1, \ldots, \succeq_n)$ 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:

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

Obvious generalisations of this idea include representing preference relations via weighted sets of temporal formulae (agents prefer runs that maximise the sum of weights of satisfied formulae), or ordered lists of temporal formulae (agents prefer runs that satisfy formulae as far up the list as possible) (Bouyer et al. 2015).

Nash equilibrium: With this definition, we can now define the standard game theoretic concept of Nash equilibrium for our setting. Let $M = (P_1, \ldots, P_n, \succeq_1, \ldots, \succeq_n)$ be a multi-agent system, and let $\vec{\sigma} = (\sigma_1, \ldots, \sigma_i, \ldots, \sigma_n)$ be a strategy profile. Then we 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 (Maschler, Solan, and Zamir 2013) – to keep things simple, in this paper we will restrict our attention to Nash equilibrium.

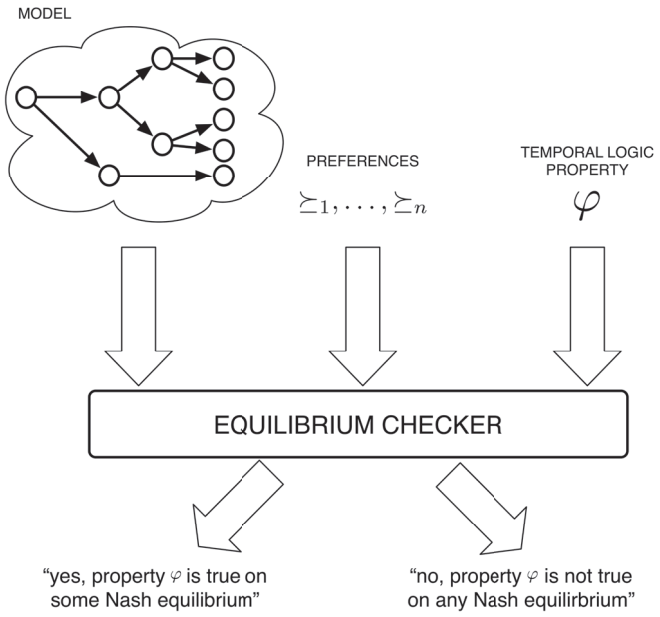

Figure 2: Equilibrium checking. The key difference to model checking is that we also take as input the preferences of each of the system components, and the key question asked is whether or not the temporal property φ holds on some/all equilibria of the system.

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):

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

NON-EMPTINESS:

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. This latter decision problem requires some finite representation scheme for these strategies.

A Concrete Model – Iterated Boolean Games: A concrete computational model that we have found useful to explore questions surrounding rational verification is the framework of *iterated Boolean games* (iBGs) (Gutierrez, Harrenstein, and Wooldridge 2015b). 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 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). Although this might at first sight appear to be a limited computational model, it is in fact rich enough for the setting in which player's goals are LTL formulae. We explored the complexity of the decision problems above in the iBG case: IS-NE was found to be PSPACE-complete (and hence no easier or harder than satisfiability for LTL); however, NON-EMPTINESS, E-NASH, and A-NASH are all 2EXPTIME-complete; these results hold even when goals have a very simple form. Intuitively, this increase in complexity is because all of these latter decision problems imply checking for the existence of winning strategies to achieve temporal logic formulae. This problem is called *LTL synthesis*, and is known to be 2EXPTIME-complete (Pnueli and Rosner 1989). One interesting aspect of work on iBGs is the application of the *Nash folk theorems* (results in game theory for the analysis of iterated games) to obtain results relating to the complexity of equilibrium checking in iBGs.

A Prototype Equilibrium Checking Tool

We now describe a prototype tool we have developed as a proof of concept for equilibrium checking. The tool is called EAGLE (*E*quilibrium *A*nalyser for *G*ame *L*ike *E*nvironments) (Toumi, Gutierrez, and Wooldridge 2015). EAGLE takes as input a set of agents, defined using the REACTIVE MODULES Language (RML) (Alur and Henzinger 1999), a collection of non-deterministic strategies for these agents, also encoded using REACTIVE MODULES, and for each agent P_i, a CTL formula representing the goal γ_i of that player. It then checks whether the strategies represent a Nash equilibrium, i.e., it solves the decision problem IS-NE.

The objects used to define agents in RML are known as *modules*. An RML module consists of an *interface*, which defines the name of the module and lists the Boolean variables under the *control* of the module, and 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 φ–>α 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 it is *enabled* for execution –it *may be chosen*. Formally, a guarded command g over a set of Boolean variables Φ is an expression

$$[\,]\,\varphi\text{->}x'_1 := \psi_1, \cdots, x'_k := \psi_k$$

where [] is a syntactic separator, φ (the guard) is a propositional formula, each x_i is a member of Φ and each ψ_i is a propositional formula. The intended meaning is that if the guard is satisfied in the current state of the system, then one of the choices available to the agent is to execute the assignment statements on the right hand side, with the conditions ψ_i being evaluated in the current state.

Space restrictions prevent us from describing in detail how EAGLE works internally, but the main point is that the IS-NE problem that EAGLE solves is EXPTIME-complete (Toumi, Gutierrez, and Wooldridge 2015), and hence no harder than satisfiability for CTL. In fact, EAGLE solves IS-NE by reducing it to a number of CTL model checking and satisfiability problems.

We now present a case study based on the system presented in (Fisman, Kupferman, and Lustig 2010). Consider a peer-to-peer network with two agents (the extension to $n > 2$ agents is straightforward – we restrict to two agents only due to space and ease of presentation). At each time step, each agent either tries to download or to upload. In order for one agent to download successfully, the other must be uploading at the same time, and both are interested in downloading infinitely often. We will require that an agent cannot both download and upload at the same time.

We can specify the game modelling the above communication protocol as a game with two players, 0 and 1, where each player $i \in \{0, 1\}$ controls two variables u_i ("Player i tries to upload") and d_i ("Player i tries to download"); Player

i downloads successfully if $(d_i \wedge u_{1-i})$. Formally, we model the system in RML as follows:

```
module m0 controls u0, d0
  init
  [] ⊤ -> u0' := ⊤, d0' := ⊥
  [] ⊤ -> u0' := ⊥, d0' := ⊤
  update
  [] ⊤ -> u0' := ⊤, d0' := ⊥
  [] ⊤ -> u0' := ⊥, d0' := ⊤

module m1 controls u1, d1
  init
  [] ⊤ -> u1' := ⊤, d1' := ⊥
  [] ⊤ -> u1' := ⊥, d1' := ⊤
  update
  [] ⊤ -> u1' := ⊤, d1' := ⊥
  [] ⊤ -> u1' := ⊥, d1' := ⊤
```

Agents goals can be easily specified in CTL: the informal *"infinitely often"* requirement can be expressed in CTL as *"From all system states, on all paths, eventually"*. Hence, for $i \in \{0, 1\}$, we define the goals as follows:

$$\gamma_i = \mathbf{AGAF}(d_i \wedge u_{1-i}).$$

This is clearly a very simple system/game: only two players and four controlled variables. Yet, checking the Nash equilibria of the game associated with this system is a hard problem. One can show – and formally verify using EAGLE – that this game has a Nash equilibrium where no player gets its goal achieved, and another that is Pareto optimal, where both players get their goal achieved. (In fact, the game has infinitely many Nash equilibria, but they all fall within the above two categories.) Based on the RML specifications of players' strategies given below, which can be seen to be consistent with modules m_0 and m_1, we can verify that both $(StPlayer(0), StPlayer(1)) \notin NE(G)$ and $(OnlyDown(0), OnlyDown(1)) \in NE(G)$.

```
module StPlayer(i) controls ui, di
  init
  [] ⊤ -> ui' := ⊤, di' := ⊥
  update
  [] ⊤ -> ui' := di, di' := ui

module OnlyDown(i) controls ui, di
  init
  [] ⊤ -> ui' := ⊥, di' := ⊤
  update
  [] ⊤ -> ui' := ⊥, di' := ⊤
```

Discussion & Future Work

Several research groups are working on problems related to equilibrium checking: (Fisman, Kupferman, and Lustig 2010) study the problem of synthesising systems so that certain desirable properties hold in equilibrium; (Chatterjee and Henzinger 2012; Bouyer et al. 2015) give extensive overviews of decision problems related to equilibrium checking, for a range of concurrent game models.

In our own work, we have investigated many issues surrounding rational verification, particularly using *Boolean games* (Harrenstein et al. 2001). Boolean games are essentially "one-shot" versions of iterated Boolean games, as described above, where play consists of a single round, and agent goals γ_i are specified as propositional formulae. A question that we have investigated at length is possible mechanisms for *managing* games. This might be necessary, for example, if the game contains socially undesirable equilibria (Wooldridge 2012), or where a game possesses no equilibrium, and we wish to introduce one (which we call *stabilisation*). One obvious mechanism for manipulating games is the use of taxation schemes, which provide incentives for players to avoid undesirable equilibria, or to prefer desirable equilibria (Wooldridge et al. 2013). (Related issues have recently been studied in the context of concurrent games (Almagor, Avni, and Kupferman 2015)). Another possibility is to try to influence players by changing their beliefs through communication. For example, (Grant et al. 2014) considered a Boolean games setting where players make their choices based on beliefs about some variables in the environment, and a central authority is able to reveal certain information in order to modify these beliefs. Another issue we have investigated is the extent to which we can develop a language that supports reasoning about strategies directly in the object language. Strategy logic is a variation of temporal logic, closely related to Alternating-time temporal logic (Alur, Henzinger, and Kupferman 2002), which includes names for strategies in the object language (Chatterjee, Henzinger, and Piterman 2010), and using Strategy Logic, it is possible to reason about Nash equilibria. However, Strategy logic is in general undecidable, which raises the question of whether weaker languages might be used. (Gutierrez, Harrenstein, and Wooldridge 2014) proposes a temporal logic containing a quantifier $[NE]\varphi$, meaning "φ holds on all Nash equilibrium computations". (Gutierrez, Harrenstein, and Wooldridge 2015a) shows that Nash equilibria can be represented, up to bisimulation, in even weaker languages. Other researchers have investigated similar concerns (Bulling, Jamroga, and Dix 2008). Another interesting question is the extent to which the model of interaction used in a particular setting affects the possible equilibria that may result. In (Gutierrez and Wooldridge 2014), we investigated Nash equilibria in games based on event structures (Winskel 1986), and were able to characterise conditions required for the existence of equilibria. Another research direction is the extent to which we can go beyond the representation of preferences as simple binary formulae; one possible approach, investigated in (Marchioni and Wooldridge 2015), is to represent player's goals as formulae of Lukasiewicz logic, which permits a much richer class of preferences to be directly represented.

Many issues remain for future work. Mixed (stochastic) strategies is an obvious major topic of interest, as is the possibility of imperfect information (Fagin et al. 1995), and of course solution concepts beyond Nash equilibria, such as subgame perfect equilibrium. Our EAGLE tool is a prototype, limited in its scope, and not optimised: extensions supporting LTL, incomplete information, etc, are all highly desirable.

Acknowledgements: We acknowledge the support of the European Research Council under Advanced Grant 291528 ("RACE"), and the EPSRC under grant EP/M009130/1.

References

Almagor, S.; Avni, G.; and Kupferman, O. 2015. Repairing multi-player games. In *Proceedings of the Twenty-Sixth Annual Conference on Concurrency Theory (CONCUR-2015)*.

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

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

Bouyer, P.; Brenguier, R.; Markey, N.; and Ummels, M. 2015. Pure Nash equilibria in concurrent games. *Logical Methods in Computer Science*.

Boyer, R. S., and Moore, J. S., eds. 1981. *The Correctness Problem in Computer Science*. The Academic Press: London, England.

Brafman, R., and Domshlak, C. 2013. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence* 198:52–71.

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

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

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

Clarke, E. M., and Emerson, E. A. 1981. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs — Proceedings 1981 (LNCS Volume 131)*, 52–71. Springer-Verlag: Berlin, Germany.

Clarke, E. M.; Grumberg, O.; and Peled, D. A. 2000. *Model Checking*. The MIT Press: Cambridge, MA.

Emerson, E. A. 1990. Temporal and modal logic. In *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands. 996–1072.

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

Fisman, D.; Kupferman, O.; and Lustig, Y. 2010. Rational synthesis. In *TACAS*, volume 6015 of *LNCS*, 190–204. Springer.

Grant, J.; Kraus, S.; Wooldridge, M.; and Zuckerman, I. 2014. Manipulating games by sharing information. *Studia Logica* 102:267–295.

Gutierrez, J., and Wooldridge, M. 2014. Equilibria of concurrent games on event structures. In *Proceedings of CSL-LICS*.

Gutierrez, J.; Harrenstein, P.; and Wooldridge, M. 2014. Reasoning about equilibria in game-like concurrent systems. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014*.

Gutierrez, J.; Harrenstein, P.; and Wooldridge, M. 2015a. Expressiveness and complexity results for strategic reasoning. In *Proceedings of the Twenty-Sixth Annual Conference on Concurrency Theory (CONCUR-2015)*.

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

Harrenstein, P.; van der Hoek, W.; Meyer, J.-J.; and Witteveen, C. 2001. Boolean games. In van Benthem, J., ed., *Proceeding of the Eighth Conference on Theoretical Aspects of Rationality and Knowledge (TARK VIII)*, 287–298.

Marchioni, E., and Wooldridge, M. 2015. Lukasiewicz games: A logic-based approach to quantitative strategic interactions. *ACM Transactions on Computational Logic*.

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

Nisan, N.; Roughgarden, T.; Tardos, E.; and Vazirani, V. V., eds. 2007. *Algorithmic Game Theory*. Cambridge University Press: Cambridge, England.

Pnueli, A., and Rosner, R. 1989. On the synthesis of an asynchronous reactive module. In *Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programs*.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the Eighteenth IEEE Symposium on the Foundations of Computer Science*, 46–57.

Roth, A., and Ockenfels, A. 2002. 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):10931103.

Shoham, Y., and Leyton-Brown, K. 2008. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press: Cambridge, England.

Toumi, A.; Gutierrez, J.; and Wooldridge, M. 2015. 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)*.

Vardi, M. Y., and Wolper, P. 1986. An automata-theoretic approach to automatic program verification. In *First Symposium in Logic in Computer Science (LICS)*.

Winskel, G. 1986. Event structures. In *Advances in Petri Nets*.

Wooldridge, M.; Endriss, U.; Kraus, S.; and Lang, J. 2013. Incentive engineering for boolean games. *Artificial Intelligence* 195:418–439.

Wooldridge, M. 2009. *An Introduction to Multiagent Systems (Second edition)*. John Wiley & Sons.

Wooldridge, M. 2012. Bad equilibria (and what to do about them). In *Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI-2012)*.