Automatic Verification of Competitive Stochastic Systems

Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska,
David Parker, and Aistis Simaitis

Department of Computer Science, University of Oxford, Oxford, UK

Abstract. We present automatic verification techniques for the modelling and analysis of probabilistic systems that incorporate competitive behaviour. These systems are modelled as turn-based stochastic multi-player games, in which the players can either collaborate or compete in order to achieve a particular goal. We define a temporal logic called rPATL for expressing quantitative properties of stochastic multi-player games. This logic allows us to reason about the collective ability of a set of players to achieve a goal relating to the probability of an event's occurrence or the expected amount of cost/reward accumulated. We give a model checking algorithm for verifying properties expressed in this logic and implement the techniques in a probabilistic model checker, based on the PRISM tool. We demonstrate the applicability and efficiency of our methods by deploying them to analyse and detect potential weaknesses in a variety of large case studies, including algorithms for energy management and collective decision making for autonomous systems.

1 Introduction

Automatic verification techniques for probabilistic systems have been successfully applied in a variety of fields, from wireless communication protocols to dynamic power management schemes to quantum cryptography. These systems are inherently stochastic, e.g. due to unreliable communication media, faulty components or the use of randomisation. Automatic techniques such as *probabilistic model checking* provide a means to model and analyse these systems against a range of quantitative properties. In particular, when systems also exhibit *nondeterministic* behaviour, e.g. due to concurrency, underspecification or control, the subtle interplay between the probabilistic and nondeterministic aspects of the system often makes a manual analysis difficult and error-prone.

When modelling *open* systems, the designer also has to account for the behaviour of components it does not control, and which could have differing or opposing goals, giving rise to *competitive* behaviour. This occurs in many cases, such as security protocols and algorithms for distributed consensus, energy management or sensor network co-ordination. In such situations, it is natural to adopt a *game-theoretic* view, modelling a system as a game between different players. Automatic verification has been successfully deployed in this context, e.g. in the analysis of security [21] or communication protocols [20].

In this paper, we present an extensive framework for modelling and automatic verification of systems with both probabilistic *and* competitive behaviour, using *stochastic multi-player games* (SMGs). We introduce a *temporal logic rPATL* for expressing quantitative properties of this model and develop *model checking algorithms* for it. We then build a probabilistic model checker, based on the PRISM tool [22], which provides a high-level language for modelling SMGs and implements rPATL model checking for their analysis. Finally, to illustrate the applicability of our framework, we develop several large case studies in which we identify potential weaknesses and unexpected behaviour that would have been difficult to find with existing probabilistic verification techniques.

We model competitive stochastic systems as *turn-based* SMGs, where, in each state of the model, one player chooses between several actions, the outcome of which can be probabilistic. Turn-based games are a natural way to model many real-life applications. One example is when modelling several components executing concurrently under the control of a particular (e.g. round-robin, randomised) scheduler; in this case, nondeterminism in the model arises due to the choices made by each individual component. Another example is when we choose to explicitly model the (possibly unknown) scheduling of components as one player and the choices of components as other players.

In order to specify properties of the systems modelled, we formulate a temporal logic, rPATL. This is an extension of the logic PATL [14], which is itself a probabilistic extension of ATL [5] – a widely used logic for reasoning about multi-player games and multi-agent systems. rPATL allows us to state that a *coalition* of players has a *strategy* which can ensure that either the *probability* of an event's occurrence or an *expected reward* measure meets some threshold, e.g. "can processes 1 and 2 collaborate so that the probability of the protocol terminating within 45 seconds is at least 0.95, whatever processes 3 and 4 do?"

We place particular emphasis on *reward* (or, equivalently, *cost*) related measures. This allows us to reason quantitatively about a system's use of resources, such as time spent or energy consumed; or, we can use rewards as an algorithm design mechanism to validate, benchmark or synthesise strategies for components by rewarding or penalising them for certain behaviour. rPATL can state, for example, "can sensor 1 ensure that the expected energy used, *if the algorithm terminates*, is less than $75mJ$, for any actions of sensors 2, 3, and 4?". To the best of our knowledge, this is the first logic able to express such properties.

We include in rPATL three different *cumulative expected reward* operators. Cumulative properties naturally capture many useful system properties, as has been demonstrated for verification of other types of probabilistic models [17], and as proves to be true for the systems we investigate. Indicative examples from our case studies are "the maximum expected execution cost of a task in a Microgrid" and "the minimum expected number of messages required to reach a consensus". Several other reward-based objectives exist that we do not consider, including discounted rewards (useful e.g. in economics, but less so for the kind of systems we target) and long-run average reward (also useful, but practical implementations become complex in stochastic games [16]).

We also devise model checking algorithms for rPATL. A practical advantage of the logic is that, like for ATL, model checking reduces to analysing zero-sum two-player games. rPATL properties referring to the probability of an event are checked by solving simple stochastic two-player games, for which efficient techniques exist [15,16]. For reward-based properties, we present new algorithms.

Lastly, we develop and analyse several large case studies. We study algorithms for smart energy management [19] and distributed consensus in a sensor network [26]. In the first case, we use our techniques to reveal a weakness in the algorithm: we show that users may have a high incentive to deviate from the original algorithm, and propose modifications to solve the problem. For the consensus algorithm, we identify unexpected trade-offs in the performance of the algorithm when using our techniques to evaluate possible strategies for sensors.

Contributions. In summary, the contributions of this paper are:

- A comprehensive *framework* for analysis of competitive stochastic systems;
- A *logic* rPATL for specifying quantitative properties of stochastic multi-player games including, in particular, novel operators for *costs and rewards*, and their *model checking algorithms*;
- Implementation of a *tool* for modelling and rPATL model checking of SMGs;
- Development and analysis of several large new *case studies*.

An extended version of this paper, with proofs, is available as [12].

Related work. There exist theoretical results on probabilistic temporal logics for a game-theoretic setting but, to our knowledge, this is the first work to consider a practical implementation, modelling and automated verification of case studies. [14] introduces the logic PATL, showing its model checking complexity via probabilistic parity games. [28] studies simulation relations preserved by PATL and [1] uses it in a theoretical framework for security protocol analysis. [6] presents (un)decidability results for another richer logic, with emphasis on the subtleties of nested properties. We note that all of the above, except [6], use concurrent, rather than turn-based, games and none consider reward properties.

Probabilistic model checking for a multi-agent system (a negotiation protocol) is considered in [8], but this is done by fixing a particular probabilistic strategy and analysing a Markov chain rather than a stochastic game. [13] describes analysis of a team formation protocol, which involves simple properties on stochastic two-player games. There has been much research on algorithms to solve stochastic games, e.g. [16,11,27], but these do not consider a modelling framework, implementation or case studies. Moreover, the reward-based properties that we introduce in this paper have not been studied in depth. In [25], a quantitative generalisation of the μ-calculus is proposed, and shown to be able to encode stochastic parity games. We also mention the tools MCMAS [23] and MOCHA [4], powerful model checkers for *non-probabilistic* multi-agent systems.

Finally, stochastic games are useful for *synthesis*, as in e.g. [9], which synthesises concurrent programs for randomised schedulers. Also, the tool GIST [10] is a stochastic game solver, but is targeted at synthesis problems, not modelling and verification of competitive systems, and only supports *qualitative* properties.

2 Preliminaries

We begin with some background on stochastic multi-player games. For a finite set X, we denote by $\mathcal{D}(X)$ the set of discrete probability distributions over X.

Definition 1 (SMG). *A (turn-based)* stochastic multi-player game *(SMG) is a tuple* $\mathcal{G} = \langle \Pi, S, A, (S_i)_{i\in\Pi}, \Delta, AP, \chi\rangle$*, where:* Π *is a finite set of* players*;* S *is a finite, non-empty set of* states*;* A *is a finite, non-empty set of* actions*;* $(S_i)_{i\in\Pi}$ *is a partition of* S*;* $\Delta : S \times A \to \mathcal{D}(S)$ *is a (partial)* transition function*;* AP *is a finite set of* atomic propositions*; and* $\chi : S \to 2^{AP}$ *is a* labelling function*.*

In each state $s \in S$ of the SMG $\mathcal{G}$, the set of *available* actions is denoted by $A(s) \stackrel{\text{def}}{=} \{a \in A \mid \Delta(s,a) \neq \bot\}$. We assume that $A(s) \neq \emptyset$ for all s. The choice of action to take in s is under the control of exactly one player, namely the player $i \in \Pi$ for which $s \in S_i$. Once action $a \in A(s)$ is selected, the successor state is chosen according to the probability distribution $\Delta(s,a)$. A *path* of $\mathcal{G}$ is a possibly infinite sequence $\lambda = s_0 a_0 s_1 a_1 \ldots$ such that $a_j \in A(s_j)$ and $\Delta(s_j, a_j)(s_{j+1}) > 0$ for all j. We use st_λ to denote $s_0 s_1 \ldots$, and $st_\lambda(j)$ for s_j. The set of all infinite paths is $\Omega_\mathcal{G}$ and the set of infinite paths starting in state s is $\Omega_{\mathcal{G},s}$.

A *strategy* for player $i \in \Pi$ in $\mathcal{G}$ is a function $\sigma_i : (SA)^* S_i \to \mathcal{D}(A)$ which, for each path $\lambda{\cdot}s$ where $s \in S_i$, assigns a probability distribution $\sigma_i(\lambda{\cdot}s)$ over $A(s)$. The set of all strategies for player i is denoted Σ_i. A strategy σ_i is called *memoryless* if $\forall \lambda, \lambda' : \sigma_i(\lambda{\cdot}s) = \sigma_i(\lambda'{\cdot}s)$, and *deterministic* if $\forall \lambda : \sigma_i(\lambda{\cdot}s)$ is a Dirac distribution. A *strategy profile* $\sigma = \sigma_1, \ldots, \sigma_{|\Pi|}$ comprises a strategy for all players in the game. Under a strategy profile σ, the behaviour of $\mathcal{G}$ is fully probabilistic and we define a *probability measure* $\Pr^\sigma_{\mathcal{G},s}$ over the set of all paths $\Omega_{\mathcal{G},s}$ in standard fashion (see, e.g. [11]). Given a random variable $X : \Omega_{\mathcal{G},s} \to \mathbb{R}$, we define the *expected value* of X to be $\mathbb{E}^\sigma_{\mathcal{G},s}[X] \stackrel{\text{def}}{=} \int_{\Omega_{\mathcal{G},s}} X \, d\Pr^\sigma_{\mathcal{G},s}$.

We also augment games with *reward structures* $r : S \to \mathbb{Q}_{\geq 0}$, mapping each state to a non-negative rational reward. To simplify presentation, we only use state rewards, but note that transition/action rewards can easily be encoded by adding an auxiliary state per transition/action to the model.

3 Property Specification: The Logic rPATL

We now present a temporal logic called rPATL (Probabilistic Alternating-time Temporal Logic with Rewards) for expressing quantitative properties of SMGs. Throughout the section, we assume a fixed SMG $\mathcal{G} = \langle \Pi, S, A, (S_i)_{i\in\Pi}, \Delta, AP, \chi\rangle$.

Definition 2 (rPATL). *The syntax of rPATL is given by the grammar:*

$$\phi ::= \top \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\!\langle C \rangle\!\rangle \mathsf{P}_{\bowtie q}[\psi] \mid \langle\!\langle C \rangle\!\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^\star \phi]$$
$$\psi ::= \mathsf{X}\,\phi \mid \phi\,\mathsf{U}^{\leq k}\,\phi \mid \phi\,\mathsf{U}\,\phi$$

where $a \in AP$*,* $C \subseteq \Pi$*,* $\bowtie \in \{<, \leq, \geq, >\}$*,* $q \in \mathbb{Q} \cap [0,1]$*,* $x \in \mathbb{Q}_{\geq 0}$*,* $\star \in \{0, \infty, c\}$*,* r *is a reward structure and* $k \in \mathbb{N}$*.*

rPATL is a CTL-style branching-time temporal logic, where we distinguish state formulae (ϕ) and path formulae (ψ). We adopt the coalition operator $\langle\langle C\rangle\rangle$ of ATL [5], combining it with the probabilistic operator $\mathsf{P}_{\bowtie q}[\cdot]$ from PCTL [18] and a generalised variant of the reward operator $\mathsf{R}^r_{\bowtie x}[\cdot]$ from [17].

An example of typical usage of the coalition operator is $\langle\langle\{1,2\}\rangle\rangle\mathsf{P}_{\geq 0.5}[\psi]$, which means "players 1 and 2 have a strategy to ensure that the probability of path formula ψ being satisfied is at least 0.5, regardless of the strategies of other players". As path formulae, we allow the standard temporal operators X ("next"), bounded $\mathsf{U}^{\leq k}$ ("bounded until") and U ("until").

Rewards. Before presenting the semantics of rPATL, we discuss the reward operators in the logic. We focus on *expected cumulative reward*, i.e. the expected sum of rewards cumulated along a path until a state from a specified target set $T \subseteq S$ is reached. To cope with the variety of different properties encountered in practice, we introduce three variants, which differ in the way they handle the case where T is *not* reached. The three types are denoted by the parameter $\star$, one of 0, ∞ or c. These indicate that, when T is not reached, the reward is zero, infinite or equal to the cumulated reward along the whole path, respectively.

Each reward type is applicable in different situations. If our goal is, for example, to minimise the expected time for algorithm completion, then it is natural to assume a value of infinity upon non-completion ($\star{=}\infty$). Consider, on the other hand, the case where we try to optimise a distributed algorithm by designing a reward structure that incentivises certain kinds of behaviour and then maximising it over the lifetime of the algorithm's execution. In this case, we might opt for type $\star{=}0$ to avoid favouring situations where the algorithm does not terminate. In other cases, e.g. when modelling algorithm's resource consumption, we might prefer to use type $\star{=}\, c$, to compute resources used regardless of termination.

We formalise these notions of rewards by defining *reward functions* that map each possible path in the game $\mathcal{G}$ to a cumulative reward value.

Definition 3 (Reward Function). *For an SMG $\mathcal{G}$, a reward structure r, type $\star \in \{0, \infty, c\}$ and a set $T \subseteq S$ of target states, the* reward function *$rew(r, \star, T) : \Omega_{\mathcal{G}} \to \mathbb{R}$ is a random variable defined as follows.*

$$rew(r,\star,T)(\lambda) \stackrel{\text{def}}{=} \begin{cases} g(\star) & \text{if } \forall j \in \mathbb{N} : st_\lambda(j) \notin T, \\ \sum_{j=0}^{k-1} r(st_\lambda(j)) & \text{otherwise, where } k = \min\{j \mid st_\lambda(j) \in T\}, \end{cases}$$

and where $g(\star) = \star$ if $\star \in \{0, \infty\}$ and $g(\star) = \sum_{j\in\mathbb{N}} r(st_\lambda(j))$ if $\star = c$. The expected reward *from a state $s \in S$ of $\mathcal{G}$ under a strategy profile σ is the expected value of the reward function, $\mathbb{E}^\sigma_{\mathcal{G},s}[rew(r, \star, T)]$.*

Semantics. Now, we define the semantics of rPATL. Formulae are interpreted over states of a game $\mathcal{G}$; we write $s \models \phi$ to indicate that the state s of $\mathcal{G}$ satisfies the formula ϕ and define $Sat(\phi) \stackrel{\text{def}}{=} \{s \in S \mid s \models \phi\}$ as the states satisfying ϕ. The meaning of atomic propositions and logical connectives is standard. For the $\langle\langle C\rangle\rangle\mathsf{P}_{\bowtie q}$ and $\langle\langle C\rangle\rangle\mathsf{R}^r_{\bowtie x}$ operators, we give the semantics via a reduction to a two-player game called a *coalition game*.

Definition 4 (Coalition Game). *For a* coalition *of players $C \subseteq \Pi$ of SMG $\mathcal{G}$, we define the* coalition game *of $\mathcal{G}$ induced by C as the stochastic two-player game $\mathcal{G}_C = \langle\{1,2\}, S, A, (S'_1, S'_2), \Delta, AP, \chi\rangle$ where $S'_1 = \cup_{i\in C} S_i$ and $S'_2 = \cup_{i\in\Pi\setminus C} S_i$.*

Definition 5 (rPATL Semantics). *The* satisfaction relation $\models$ *for rPATL is defined inductively for every state s of $\mathcal{G}$. The semantics of $\top$, atomic propositions and formulae of the form $\neg\phi$ and $\phi_1 \wedge \phi_2$ is defined in the usual way. For the temporal operators, we define:*

$$\begin{aligned} s \models \langle\!\langle C\rangle\!\rangle \mathsf{P}_{\bowtie q}[\psi] &\Leftrightarrow \textit{In coalition game } \mathcal{G}_C, \exists \sigma_1 \in \Sigma_1 \textit{ such that } \forall \sigma_2 \in \Sigma_2 \\ &\quad \Pr^{\sigma_1,\sigma_2}_{\mathcal{G}_C,s}(\psi) \bowtie q \\ s \models \langle\!\langle C\rangle\!\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^\star \phi] &\Leftrightarrow \textit{In coalition game } \mathcal{G}_C, \exists \sigma_1 \in \Sigma_1 \textit{ such that } \forall \sigma_2 \in \Sigma_2 \\ &\quad \mathbb{E}^{\sigma_1,\sigma_2}_{\mathcal{G}_C,s}[rew(r,\star,Sat(\phi))] \bowtie x \end{aligned}$$

where $\Pr^{\sigma_1,\sigma_2}_{\mathcal{G}_C,s}(\psi) \stackrel{\text{def}}{=} \Pr^{\sigma_1,\sigma_2}_{\mathcal{G}_C,s}(\{\lambda \in \Omega_{\mathcal{G}_C,s} \mid \lambda \models \psi\})$ *and for any path λ in $\mathcal{G}$:*

$$\begin{aligned} \lambda \models \mathsf{X}\,\phi &\Leftrightarrow st_\lambda(1) \models \phi \\ \lambda \models \phi_1 \,\mathsf{U}^{\leq k}\, \phi_2 &\Leftrightarrow st_\lambda(i) \models \phi_2 \textit{ for some } i \leq k \textit{ and } st_\lambda(j) \models \phi_1 \textit{ for } 0 \leq j < i \\ \lambda \models \phi_1 \,\mathsf{U}\, \phi_2 &\Leftrightarrow \lambda \models \phi_1 \,\mathsf{U}^{\leq k}\, \phi_2 \textit{ for some } k \in \mathbb{N}. \end{aligned}$$

Equivalences and Extensions. We can handle "negated path formulae" in a $\langle\!\langle C\rangle\!\rangle \mathsf{P}_{\bowtie q}$ operator by inverting the probability threshold, e.g.:

$$\langle\!\langle C\rangle\!\rangle \mathsf{P}_{\geq q}[\neg\psi] \equiv \langle\!\langle C\rangle\!\rangle \mathsf{P}_{\leq 1-q}[\psi].$$

This allows us to derive, for example, the G ("globally") and R ("release") operators. Also, from the determinacy result of [24] for zero-sum stochastic two-player games with Borel measurable payoffs, it follows that, e.g.:

$$\langle\!\langle C\rangle\!\rangle \mathsf{P}_{\geq q}[\psi] \equiv \neg\langle\!\langle \Pi \setminus C\rangle\!\rangle \mathsf{P}_{<q}[\psi]. \tag{1}$$

Finally, it is useful to consider "quantitative" versions of the $\langle\!\langle C\rangle\!\rangle \mathsf{P}$ and $\langle\!\langle C\rangle\!\rangle \mathsf{R}$ operators, in the style of PRISM [22], which return numerical values:

$$\begin{aligned} \langle\!\langle C\rangle\!\rangle \mathsf{P}_{\max=?}[\psi] &\stackrel{\text{def}}{=} \Pr^{\max,\min}_{\mathcal{G}_C,s}(\psi) \stackrel{\text{def}}{=} \sup_{\sigma_1\in\Sigma_1} \inf_{\sigma_2\in\Sigma_2} \Pr^{\sigma_1,\sigma_2}_{\mathcal{G}_C,s}(\psi) \\ \langle\!\langle C\rangle\!\rangle \mathsf{R}^r_{\max=?}[\mathsf{F}^\star\phi] &\stackrel{\text{def}}{=} \mathbb{E}^{\max,\min}_{\mathcal{G}_C,s}[rew(r,\star,Sat(\phi))] \\ &\stackrel{\text{def}}{=} \sup_{\sigma_1\in\Sigma_1} \inf_{\sigma_2\in\Sigma_2} \mathbb{E}^{\sigma_1,\sigma_2}_{\mathcal{G}_C,s}[rew(r,\star,Sat(\phi))]. \end{aligned} \tag{2}$$

Example 1. Consider the SMG in Fig. 1. with Π=$\{1,2,3\}$. The player i controlling a state s is shown as i:s in the figure, e.g. S_1=$\{s_0, s_3\}$. We have actions A=$\{a,b\}$ and e.g. $\Delta(s_0,a)(s_1)$=0.7. State s_3 is labelled with atomic proposition t. Consider the rPATL formulae $\langle\!\langle\{1,3\}\rangle\!\rangle \mathsf{P}_{\geq 0.5}[\mathsf{F}\, t]$ and $\langle\!\langle\{1,2\}\rangle\!\rangle \mathsf{P}_{\geq 0.5}[\mathsf{F}\, t]$. The first is satisfied in states $\{s_0, s_2, s_3\}$, the latter in s_3 only. Let r be a reward structure that assigns i to state s_i. rPATL formula $\langle\!\langle\{1,3\}\rangle\!\rangle \mathsf{R}^r_{\leq 2}[\mathsf{F}^\infty t]$ is true in states $\{s_2, s_3\}$. Formula $\langle\!\langle\{1\}\rangle\!\rangle \mathsf{R}^r_{\geq q}[\mathsf{F}^0 t]$ is false in all states for any $q > 0$ but $\langle\!\langle\{3\}\rangle\!\rangle \mathsf{R}^r_{\geq q}[\mathsf{F}^c t]$ is true in $\{s_0, s_1, s_2\}$ for any $q > 0$.

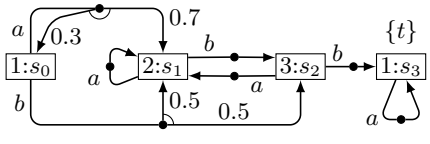


Fig. 1: Example SMG.

4 Model Checking for rPATL

We now discuss model checking for rPATL, the key part of which is computation of probabilities and expected rewards for stochastic two-player games. The complexity of the rPATL model checking problem can be stated as follows.

Theorem 1. (a) *Model checking an rPATL formula with no* $\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^0\phi]$ *operator and where* k *for* $\mathsf{U}^{\leq k}$ *is given in unary is in* $\mathrm{NP} \cap \mathrm{coNP}$.
(b) *Model checking an unrestricted rPATL formula is in* $\mathrm{NEXP} \cap \mathrm{coNEXP}$.

Nevertheless, we present efficient and practically usable algorithms for model checking rPATL, in which computation of numerical values is done by evaluating fixpoints (up to a desired level of convergence[1]).

The basic algorithm for model checking an rPATL formula ϕ on an SMG $\mathcal{G}$ proceeds as for other branching-time logics, determining the set $Sat(\phi)$ recursively. Furthermore, as can be seen from the semantics, computing this set for atomic propositions or logical connectives is trivial. Thus, we only consider the $\langle\langle C\rangle\rangle \mathsf{P}_{\bowtie q}$ and $\langle\langle C\rangle\rangle \mathsf{R}^r_{\bowtie x}$ operators. Like for the logic PCTL, model checking of these reduces to computation of *optimal* probabilities or expected rewards, respectively, on the coalition game $\mathcal{G}_C$. For example, if $\triangleright \in \{\geq, >\}$, then:

$$s \models \langle\langle C\rangle\rangle \mathsf{P}_{\triangleright q}[\psi] \;\Leftrightarrow\; \Pr^{\max,\min}_{\mathcal{G}_C,s}(\psi) \triangleright q$$
$$s \models \langle\langle C\rangle\rangle \mathsf{R}^r_{\triangleright x}[\mathsf{F}^\star\phi] \;\Leftrightarrow\; \mathbb{E}^{\max,\min}_{\mathcal{G}_C,s}[rew(r,\star,Sat(\phi))] \triangleright x.$$

Analogously, for operators $\leq$ and $<$, we simply swap min and max in the above. The following sections describe how to compute these values.

4.1 Computing Probabilities

Below, we show how to compute the probabilities $\Pr^{\max,\min}_{\mathcal{G}_C,s}(\psi)$ where ψ is each of the temporal operators X, $\mathsf{U}^{\leq k}$ and U. We omit the dual case since, thanks to determinacy (see equation (1)), we have that $\Pr^{\min,\max}_{\mathcal{G}_C,s}(\psi) = \Pr^{\max,\min}_{\mathcal{G}_{\Pi\setminus C},s}(\psi)$. The following results follow in near identical fashion to the corresponding results for Markov decision processes [7]. We let opt^s denote max if $s \in S_1$ and min if $s \in S_2$. For the X operator and state $s \in S$:

$$\Pr^{\max,\min}_{\mathcal{G}_C,s}(\mathsf{X}\,\phi) = \mathrm{opt}^s_{a\in A(s)} \sum\nolimits_{s'\in Sat(\phi)} \Delta(s,a)(s').$$

Probabilities for the $\mathsf{U}^{\leq k}$ operator can be computed recursively. We have that $\Pr^{\max,\min}_{\mathcal{G}_C,s}(\phi_1\,\mathsf{U}^{\leq k}\,\phi_2)$ is equal to: 1 if $s \in Sat(\phi_2)$; 0 if $s \notin (Sat(\phi_1) \cup Sat(\phi_2))$; 0 if $k{=}0$ and $s \in Sat(\phi_1)\setminus Sat(\phi_2)$; and otherwise:

$$\Pr^{\max,\min}_{\mathcal{G}_C,s}(\phi_1\,\mathsf{U}^{\leq k}\,\phi_2) = \mathrm{opt}^s_{a\in A(s)} \sum\nolimits_{s'\in S} \Delta(s,a)(s') \cdot \Pr^{\max,\min}_{\mathcal{G}_C,s'}(\phi_1\,\mathsf{U}^{\leq k-1}\,\phi_2).$$

The unbounded case can be computed via *value iteration* [15], i.e. using:

$$\Pr^{\max,\min}_{\mathcal{G}_C,s}(\phi_1\,\mathsf{U}\,\phi_2) = \lim_{k\to\infty} \Pr^{\max,\min}_{\mathcal{G}_C,s}(\phi_1\,\mathsf{U}^{\leq k}\,\phi_2).$$

[1] This is the usual approach taken in probabilistic verification tools.

In practice, this computation is terminated with a suitable convergence check (see, e.g. [17]). In addition, we mention that for the case $\mathsf{F}\,\phi \equiv \top\,\mathsf{U}\,\phi$, the computation can also be reduced to quadratic programming [16].

4.2 Computing Rewards

Now, we show how to compute the optimal values $\mathbb{E}^{\max,\min}_{\mathcal{G}_C,s}[rew(r,\star,Sat(\phi))]$ for different types of $\star$. As above, we omit the dual case where max and min are swapped. In this section, we fix a coalition game $\mathcal{G}_C$, a reward structure r, and a target set $T = Sat(\phi)$. We first make the following modifications to $\mathcal{G}_C$:

- labels are added to target and positive reward states: $AP := AP \cup \{t, a_{\text{rew}}\}$, $\forall s \in T : \chi(s) := \chi(s) \cup \{t\}$ and $\forall s \in S\,.\,r(s) > 0 : \chi(s) := \chi(s) \cup \{a_{\text{rew}}\}$;
- target states are made absorbing: $\forall s \in T : A(s) := \{a\}, \Delta(s,a)(s){=}1, r(s){=}0$.

Our algorithms, like the ones for similar properties on simpler models [7], rely on computing fixpoints of certain sets of equations. As in the previous section, we assume that this is done by *value iteration* with an appropriate convergence criterion. We again let opt^s denote max if $s \in S_1$ and min if $s \in S_2$.

An important observation here is that optimal expected rewards for $\star \in \{\infty, c\}$ can be achieved by memoryless, deterministic strategies. For $\star = 0$, however, finite-memory strategies are needed. See [12] for details.

The case $\star = c$. First, we use the results of [3] to identify the states from which the expected reward is infinite:

$$I := \{s \in S \mid \exists \sigma_1 {\in} \Sigma_1\ \forall \sigma_2 {\in} \Sigma_2\ \Pr^{\sigma_1,\sigma_2}_{\mathcal{G}_C,s}(\mathit{inf}(a_{rew})) > 0\}$$

where $\mathit{inf}(a_{rew})$ is the set of all paths that visit a state satisfying a_{rew} infinitely often. We remove the states of I from $\mathcal{G}_C$. For the other states, we compute the *least* fixpoint of the following equations:

$$f(s) = \begin{cases} 0 & \text{if } s \in T \\ r(s) + \text{opt}^s_{a \in A(s)} \sum_{s' \in S} \Delta(s,a)(s') \cdot f(s') & \text{otherwise} \end{cases} \tag{3}$$

The case $\star = \infty$. Again, we start by identifying and removing states with infinite expected reward; in this case: $I := \{s \in S \mid s \models \langle\!\langle \{1\} \rangle\!\rangle \mathsf{P}_{<1}[\,\mathsf{F}\,t\,]\}$. Then, for all other states s, we compute the *greatest* fixpoint, *over* $\mathbb{R}$, of equations (3). The need for the greatest fixpoint arises because, in the presence of zero-reward cycles, multiple fixpoints may exist. The computation is over $\mathbb{R}$ since, e.g. the function mapping all non-target states to ∞ may also be a fixpoint. To find the greatest fixpoint over $\mathbb{R}$, we first compute an over-approximation by changing all zero rewards to any $\varepsilon > 0$ and then evaluating the *least* fixpoint of (3) for the *modified* reward. Starting from the *new* initial values, value iteration now converges from above to the correct fixpoint [12]. For the simpler case of MDPs, an alternative approach based on removal of zero-reward end-components is possible [2], but this cannot be adapted efficiently to stochastic games.

The case $\star = 0$. As mentioned above, it does not suffice to consider memoryless strategies in this case. The optimal strategy may depend on the reward accumulated so far, $\mathrm{r}(\lambda) \stackrel{\text{def}}{=} \sum_{s \in st_\lambda} r(s)$ for history λ. However, this is only needed until a certain reward bound B is reached, after which the optimal strategy picks actions that maximise the probability of reaching T (if multiple such actions exist, it picks the one with the highest expected reward). The bound B can be computed efficiently using algorithms for $\star = c$ and $\Pr^{\max,\min}_{\mathcal{G}_C,s}(\psi)$ and, in the worst case, can be exponential in the size of $\mathcal{G}$ (see [12]).

For clarity, we assume that rewards are integers. Let $R_{(s,k)}$ be the maximum expectation of $rew(r, 0, T)$ in state s after history λ with $\mathrm{r}(\lambda) = k$:

$$R_{(s,k)} \stackrel{\text{def}}{=} \max_{\sigma_1 \in \Sigma_1} \min_{\sigma_2 \in \Sigma_2} k \cdot \Pr^{\sigma_1,\sigma_2}_{\mathcal{G}_C,s}(\mathsf{F}\, t) + \mathbb{E}^{\sigma_1,\sigma_2}_{\mathcal{G}_C,s}[rew(r, 0, T)],$$

and $r_{\max} = \max_{s \in S} r(s)$. The algorithm works as follows:

1. Using the results of [3], identify the states that have infinite reward:

   $$I := \{s \in S \mid \exists \sigma_1 \in \Sigma_1\ \forall \sigma_2 \in \Sigma_2\ \Pr^{\sigma_1,\sigma_2}_{\mathcal{G}_C,s}(inf^t(a_{rew})) > 0\}\}$$

   where $inf^t(a_{rew})$ is the set of all paths that visit a state satisfying $\mathsf{P}_{>0}[\mathsf{F}\, t]) \wedge a_{rew}$ infinitely often. Remove all states of I from the game.
2. For $B \leq k \leq B + r_{\max} - 1$ and for each state s:
   (a) Assign new reward $r'(s) = r(s) \cdot \Pr^{\max,\min}_{\mathcal{G}_C,s}(\mathsf{F}\, t)$;
   (b) Remove from $A(s)$ actions a that are sub-optimal for $\Pr^{\max,\min}_{\mathcal{G}_C,s}(\mathsf{F}\, t)$, i.e.:

   $$\textstyle\sum_{s' \in S} \Delta(s,a)(s') \cdot \Pr^{\max,\min}_{\mathcal{G}_C,s'}(\mathsf{F}\, t) < \Pr^{\max,\min}_{\mathcal{G}_C,s}(\mathsf{F}\, t)$$

   (c) Compute $R_{(s,k)}$ using the algorithm for $rew(r', c, T)$:

   $$R_{(s,k)} = k \cdot \Pr^{\max,\min}_{\mathcal{G}_C,s}(\mathsf{F}\, t) + \mathbb{E}^{\max,\min}_{\mathcal{G}_C,s}[rew(r', c, T)].$$

3. Find, for all $0 \leq k < B$ and states s, the *least* fixpoint of the equations:

   $$R_{(s,k)} = \begin{cases} k & \text{if } s \in T \\ \mathrm{opt}^s_{a \in A(s)} \sum_{s' \in S} \Delta(s,a)(s') \cdot R_{(s',k+r(s))} & \text{otherwise.} \end{cases}$$

4. The required values are then $\mathbb{E}^{\max,\min}_{\mathcal{G}_C,s}[rew(r, 0, T)] = R_{(s,0)}$.

5 Implementation and Case Studies

Based on the techniques in this paper, we have built a probabilistic model checker for stochastic multi-player games as an extension of the PRISM tool [22]. For modelling of SMGs, we have extended the PRISM modelling language. This allows multiple parallel components (called modules) which can either operate asynchronously or by synchronising over common action labels. Now, a model

Case study [parameters]		SMG statistics			Model checking		
		Players	States	Transitions	Prop. type	Constr. (s)	Model ch. (s)
mdsm [N]	5	5	743,904	2,145,120	$\langle\langle C\rangle\rangle \mathsf{R}^{r}_{\max=?}[\mathsf{F}^{0}\phi]$	14.5	61.9
	6	6	2,384,369	7,260,756		55.0	221.7
	7	7	6,241,312	19,678,246		210.7	1,054.8
cdmsn [N]	3	3	1,240	1,240	$\langle\langle C\rangle\rangle \mathsf{P}_{\bowtie q}[\,\mathsf{F}^{\leq k}\,\phi]$	0.2	0.2
	4	4	11,645	83,252		0.8	0.8
	5	5	100,032	843,775		3.2	6.4
investor [$vmax$]	10	2	10,868	34,264	$\langle\langle C\rangle\rangle \mathsf{R}^{r}_{\min=?}[\mathsf{F}^{c}\phi]$	1.4	0.7
	100	2	750,893	2,474,254		9.8	121.8
	200	2	2,931,643	9,688,354		45.9	820.8
team-form [N]	3	3	17,041	20,904	$\langle\langle C\rangle\rangle \mathsf{P}_{\max=?}[\mathsf{F}\,\phi]$	0.3	0.5
	4	4	184,753	226,736		4.2	2.1
	5	5	2,366,305	2,893,536		36.9	12.9

Table 1: Performance statistics for a representative set of models

also includes a set of *players*, each of which controls transitions for a disjoint subset of the modules and/or action labels. Essentially, we retain the existing PRISM language semantics (for Markov decision processes), but, in every state, each nondeterministic choice belongs to one player. For the current work, we detect and disallow the possibility of concurrent decisions between players.

Our tool constructs an SMG from a model description and then executes the algorithms from Sec. 4 to check rPATL formulae. Currently, we have developed an explicit-state model checking implementation, which we show to be efficient and scalable for various large models. It would also be relatively straightforward to adapt PRISM's symbolic model checking engines for our purpose, if required.

5.1 Experimental Results

We have applied our tool to the analysis of several large case studies: two developed solely for this work, and two others adapted from existing models. Experimental results from the new case studies are described in detail in the next sections. First, we show some statistics regarding the performance of our tool on a representative sample of models from the four case studies: Microgrid Demand-Side Management (*mdsm*) and Collective Decision Making for Sensor Networks (*cdmsn*), which will be discussed shortly; the Futures Market Investor (*investor*) example of [25]; and the team formation protocol (*team-form)* of [13]. Tab. 1 shows model statistics (number of players, states and transitions) and the time for model construction and checking a sample property on a 2.80GHz PC with 32GB RAM. All models and properties used are available online [29].

5.2 MDSM: Microgrid Demand-Side Management

Microgrid is an increasingly popular model for the future energy markets where neighbourhoods use electricity generation from local sources (e.g. wind/solar power) to satisfy local demand. The success of microgrids is highly dependent on *demand-side management*: active management of demand by users to avoid peaks. Thus, the infrastructure has to incentivise co-operation and discourage abuse. In this case study, we use rPATL model checking to analyse the MDSM infrastructure of [19] and identify an important incentive-related weakness.

The algorithm. The system in [19] consists of N *households* (HHs) connected to a single *distribution manager* (DM). At every time-step, the DM randomly contacts a HH for submission of a load for execution. The probability of the HH generating a load is determined by a daily demand curve from [19]. The duration of a load is random between 1 and D time-steps. The cost of executing a load for a single step is the number of tasks currently running. Hence, the total cost increases quadratically with HHs executing more loads in a single step.

Each household follows a very simple algorithm, the essence of which is that, when it generates a load, if the cost is below an agreed limit c_{lim}, it executes it, otherwise it only does so with a pre-agreed probability P_{start}. In [19], the *value* for each household in a time-step is measured by $V=\frac{\text{loads executing}}{\text{cost of execution}}$ and it is shown (through simulations) that, *provided every household sticks to this algorithm*, the peak demand and the total cost of energy are reduced significantly while still providing a good (expected) value V for each household.

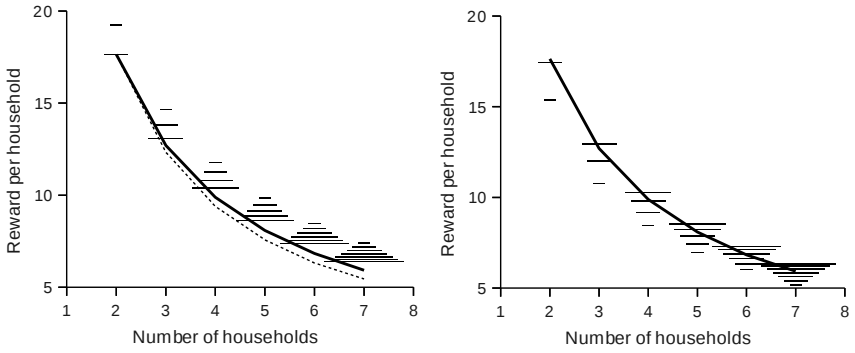
Modelling and analysis. We modelled the system as an SMG with N players, one per household. We vary $N \in \{2, \ldots, 7\}$ and fix D=4 and c_{lim}=1.5. We analyse a period of 3 days, each of 16 time-steps (using a piecewise approximation of the daily demand curve). First, as a benchmark, we assume that all households follow the algorithm of [19]. We define a reward structure r_i for the value V for household i at each step, and let $r_C = \sum_{i \in C} r_i$ be the total reward for coalition C. To compute the expected value per household, we use the rPATL query:

$$\tfrac{1}{|C|}\langle\!\langle C \rangle\!\rangle \mathsf{R}^{r_C}_{\max=?}[\,\mathsf{F}^0\ \text{time=max\_time}\,]$$

fixing, for now, C to be the set Π of all N players (households). We use this to determine the optimal value of P_{start} achievable by a memoryless strategy for each player, which we will then fix. These results are shown by the bold lines in Fig. 2. We also plot (as a dotted line) the values obtained if no demand-side management is applied.

Next, we consider the situation where the set of households C is allowed to deviate from the pre-agreed strategy, by choosing to ignore the limit c_{lim} if they wish. We check the same rPATL query as above, but now varying C to be coalitions of different sizes, $C \in \{\{1\}, \ldots, \Pi\}$. The resulting values are also plotted in Fig. 2a, shown as horizontal dashes of width proportional to $|C|$: the shortest dash represents individual deviation, the longest is a collaboration of all HHs. The former shows the maximum value that can be achieved by following the optimal collaborative strategy, and in itself presents a benchmark for the performance of the original algorithm. The key result is that deviations by individuals or small coalitions guarantee a better expected value for the HHs than *any* larger collaboration: a highly undesired weakness for an MDSM system.

Fixing the algorithm. We propose a simple punishment mechanism that addresses the problem: we allow the DM to cancel *one* job per step if the cost exceeds c_{lim}. The intuition is that, if a HH is constantly abusing the system, its job could be cancelled. This modification inverts the incentives (see Fig. 2b). The best option now is full collaboration and small coalitions who deviate *cannot* guarantee better expected values any more.



(a) Original version. (b) Version with punishment.

Fig. 2: Expected value per household for MDSM. The bold line shows all households following the algorithm of [19]; the dotted line shows the case without DSM. Horizontal dashes show deviations by collaborations of increasing size (shortest dash: individual deviation; longest dash: deviation of all households).

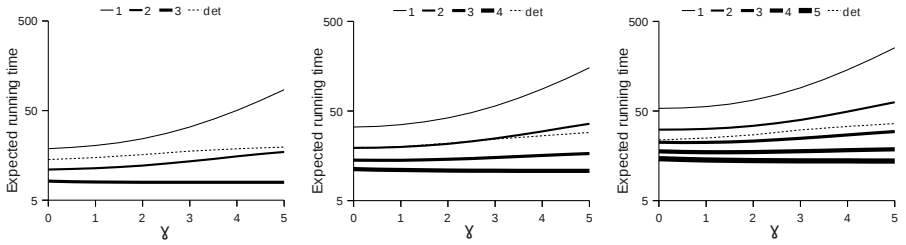
5.3 CDMSN: Collective Decision Making for Sensor Networks

Sensor networks comprise a set of low-power, autonomous devices which must act collaboratively in order to achieve a particular goal. In this case study, we illustrate the use of rPATL model checking to aid the analysis and design of such systems by studying a distributed consensus algorithm for sensor networks [26].

The algorithm. There are N sensors and a set of targets $K = \{k_1, k_2, \dots\}$, each with *quality* $Q_k \in [0, 1]$. The goal is for the sensors to agree on a target with maximum Q_k. Each sensor i stores a *preferred* target $p_i \in K$, its quality Q_{p_i} and an integer $l_i \in \{1, \dots, L\}$ to represent *confidence* in the preference. The algorithm has parameters η and λ, measuring the influence of target quality for the decision, and a parameter γ measuring the influence of the confidence level.

A sensor has three actions: *sleep*, *explore* and *communicate*. As proposed by [26], each sensor repeatedly *sleeps* for a random time t and then either *explores* (with probability P_{exp}) or *communicates*. For the *explore* action, sensor i picks a target $k \in K$ uniformly at random and with probability $P_k = Q_k^\eta / (Q_k^\eta + Q_{p_1}^\eta)$ switches its preference (p_i) to k and resets confidence to 1. To *communicate*, it compares its preference with that of a random sensor j. If they agree, both confidences are increased. If not, with probability $P_{\text{s}} = Q_{p_j}^\lambda l_j^\gamma / (Q_{p_j}^\lambda l_j^\gamma + Q_{p_i}^\lambda l_i^\gamma)$, sensor i switches preference to p_j, resets confidence to 1 and increases sensor j's confidence; with probability $1 - P_{\text{s}}$, the roles of sensors i and j are swapped.

Modelling and analysis. We have modelled the system as an SMG with N players, one per sensor. We consider models with N=3, 4, 5, three targets K=$\{k_1, k_2, k_3\}$ with qualities Q_{k_1}=1, Q_{k_2}=0.5, Q_{k_3}=0.25 and two confidence levels $l_i \in \{1, 2\}$. As in [26], we assume a random scheduling and fix parameters η=1 and λ=1. In [26], two key properties of the algorithm are studied: *speed of convergence* and *robustness*. We consider the same two issues and evaluate al-



(a) $N = 3$ (b) $N = 4$ (c) $N = 5$

Fig. 3: Expected running time until the selection of the best quality target for different models and increasing sizes of coalition C. Dotted lines show optimal performance that can be achieved using the original algorithm from [26].

ternative strategies for sensors (i.e. allowing sensors to execute any action when active). We also assume that only a subset C of the sensors are under our control, e.g., because the others are faulty. We use rPATL (with coalition C) to optimise performance, under the worst-case assumption about the other sensors.

First, we study the *speed of convergence* and the influence of parameter γ upon it. Fig. 3 shows the expected running time to reach the *best decision* (i.e. select k_1) for various values of γ and sizes of the coalition C. We use the reward structure: $r(s) = 1$ for all $s \in S$ and rPATL query:

$$\langle\!\langle C \rangle\!\rangle \mathsf{R}^{r}_{\min=?}[\, \mathsf{F}^{\infty} \textstyle\bigwedge_{i=1}^{|\Pi|} p_i = k_1 \,] \,.$$

Fig. 3 also shows the performance of the original algorithm [26] (line 'det'). We make several important observations. First, if we lose control of a few sensors, we can still guarantee convergence time comparable to the original algorithm, indicating the fault tolerance potential of the system. On the other hand, the original version performs almost as well as the optimal case for large coalitions.

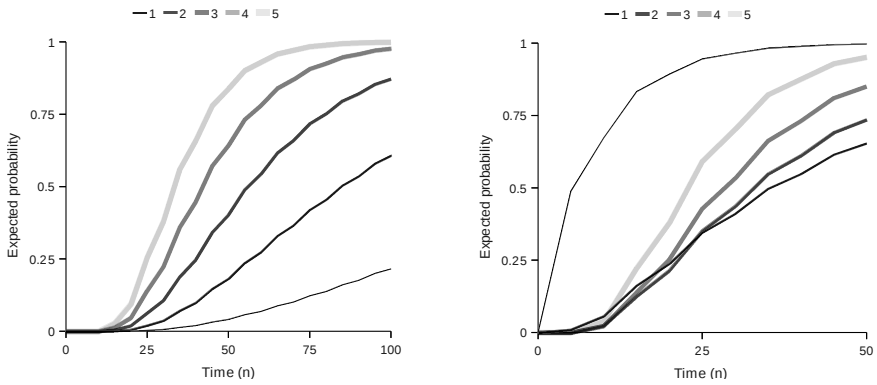
Secondly, we consider *robustness*: the ability to recover from a 'bad decision' (i.e., $\bigwedge_{i=1}^{|\Pi|} p_i = k_3$) to a 'good state' in n steps. We provide two interpretations of a 'good state' and show that the results for them are quite different.

(1) A 'good state': there exists a strategy for coalition C to make all sensors, *with probability* > 0.9*, select* k_1 *within 10 steps.* So robustness in rPATL is:

$$\langle\!\langle C \rangle\!\rangle \mathsf{P}_{\max=?}[\, \mathsf{F}^{\leq n} \langle\!\langle C \rangle\!\rangle \mathsf{P}_{>0.9}[\, \mathsf{F}^{\leq 10} \textstyle\bigwedge_{i=1}^{|\Pi|} p_i = k_1]] \,.$$

(2) A 'good state': there exists a strategy for coalition C to make all sensors *select* k_1 *while using less than* $0.5mJ$ *of energy.* We use a reward structure r_C representing energy usage by sensors in C: power consumption is $10mW$ for each communication and $1mW$ for each exploration, and each activity takes 0.1s. Then, robustness in rPATL is:

$$\langle\!\langle C \rangle\!\rangle \mathsf{P}_{\max=?}[\, \mathsf{F}^{\leq n} \langle\!\langle C \rangle\!\rangle \mathsf{R}^{r_C}_{<50}[\mathsf{F}^{c} \textstyle\bigwedge_{i=1}^{|\Pi|} p_i = k_1]] \,.$$



(a) Probability to select k_1 within 10 steps is greater than 0.9.

(b) Expected energy usage for coalition to select k_1 is less than $0.5mJ$.

Fig. 4: Minimum probability to *recover* from a state where all sensors prefer the lowest quality target, k_3, within n steps for different coalition sizes. Graphs (a) and (b) show results for two types of *recovery state* (see captions). $\gamma = 2$.

Fig. 4 shows, for each definition and for a range of values of n, the *worst-case* (minimum) value for the rPATL query from all possible 'bad states'. For (1), the results are intuitive: the larger the coalition, the faster it recovers. For (2), however, the one-sensor coalition outperforms all others. Also, we see that, in the early stages of recovery, 2-sensor coalitions outperform larger ones. This shows that small coalitions can be more resource efficient in achieving certain goals.

6 Conclusions

We have designed and implemented a framework for automatic verification of systems with both probabilistic and competitive behaviour, based on stochastic multi-player games. We proposed a new temporal logic rPATL, designed model checking algorithms, implemented them in a tool and then used our techniques to identify unexpected behaviour in several large case studies.

There are many interesting directions for future work, such as investigating extensions of our techniques to incorporate partial-information strategies or more complex solution concepts such as Nash, subgame-perfect or secure equilibria.

Acknowledgments. The authors are part supported by ERC Advanced Grant VERIWARE, the Institute for the Future of Computing at the Oxford Martin School and EPSRC grant EP/F001096/1. Vojtěch Forejt is supported by a Royal Society Newton Fellowship. We also gratefully acknowledge the anonymous referees for their helpful comments.

References

1. Aizatulin, M., Schnoor, H., Wilke, T.: Computationally sound analysis of a probabilistic contract signing protocol. In: Proc. ESORICS'09. LNCS, vol. 5789 (2009)

2. de Alfaro, L.: Computing minimum and maximum reachability times in probabilistic systems. In: CONCUR. pp. 66–81 (1999)
3. de Alfaro, L., Henzinger, T.A.: Concurrent omega-regular games. In: LICS (2000)
4. Alur, R., Henzinger, T., Mang, F., Qadeer, S., Rajamani, S., Tasiran, S.: MOCHA: Modularity in model checking. In: Proc. CAV'98. LNCS, vol. 1427 (1998)
5. Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49(5), 672–713 (2002)
6. Baier, C., Brázdil, T., Größer, M., Kucera, A.: Stochastic game logic. In: Proc. QEST'07. pp. 227–236. IEEE (2007)
7. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
8. Ballarini, P., Fisher, M., Wooldridge, M.: Uncertain agent verification through probabilistic model-checking. In: Proc. SASEMAS'06 (2006)
9. Černý, P., Chatterjee, K., Henzinger, T., Radhakrishna, A., Singh, R.: Quantitative synthesis for concurrent programs. In: Proc. CAV'11. LNCS, vol. 6806 (2011)
10. Chatterjee, K., Henzinger, T., Jobstmann, B., Radhakrishna, A.: Gist: A solver for probabilistic games. In: Proc. CAV'10. pp. 665–669. LNCS, Springer (2010)
11. Chatterjee, K.: Stochastic ω-Regular Games. Ph.D. thesis (2007)
12. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Tech. Rep. RR-11-11, University of Oxford (2011)
13. Chen, T., Kwiatkowska, M., Parker, D., Simaitis, A.: Verifying team formation protocols with probabilistic model checking. In: Proc. CLIMA'11 (2011)
14. Chen, T., Lu, J.: Probabilistic alternating-time temporal logic and model checking algorithm. In: Proc. FSKD'07. pp. 35–39. IEEE (2007)
15. Condon, A.: On algorithms for simple stochastic games. Advances in computational complexity theory, DIMACS 13, 51–73 (1993)
16. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer (1997)
17. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: SFM'11. LNCS, vol. 6659. Springer (2011)
18. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (1994)
19. Hildmann, H., Saffre, F.: Influence of variable supply and load flexibility on demand-side management. In: Proc. EEM'11. pp. 63–68 (2011)
20. van der Hoek, W., Wooldridge, M.: Model checking cooperation, knowledge, and time - A case study. Research In Economics 57(3), 235–265 (2003)
21. Kremer, S., Raskin, J.F.: A game-based verification of non-repudiation and fair exchange protocols. Journal of Computer Security 11(3), 399–430 (2003)
22. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. CAV'11. LNCS, vol. 6806, pp. 585–591. Springer (2011)
23. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A model checker for the verification of multi-agent systems. In: Proc. CAV'09. LNCS, vol. 5643. Springer (2009)
24. Martin, D.: The determinacy of Blackwell games. J. Symb. Log. 63(4) (1998)
25. McIver, A., Morgan, C.: Results on the quantitative mu-calculus qMu. ACM Transactions on Computational Logic 8(1) (2007)
26. Saffre, F., Simaitis, A.: Host selection through collective decision. ACM Transactions on Autonomous and Adaptive Systems (TAAS) (2012), to appear
27. Ummels, M.: Stochastic Multiplayer Games: Theory and Algorithms. Ph.D. thesis, RWTH Aachen University (2010)
28. Zhang, C., Pang, J.: On probabilistic alternating simulations. In: Proc. TCS'10. IFIP, vol. 323, pp. 71–85. Springer (2010)
29. `http://www.prismmodelchecker.org/files/tacas12smg/`