PRISM-games: A Model Checker for Stochastic Multi-Player Games

Taolue Chen[1], Vojtěch Forejt[1], Marta Kwiatkowska[1], David Parker[2], and Aistis Simaitis[1]

[1] Department of Computer Science, University of Oxford, UK
[2] School of Computer Science, University of Birmingham, UK

Abstract. We present PRISM-games, a model checker for stochastic multi-player games, which supports modelling, automated verification and strategy synthesis for probabilistic systems with competitive or co-operative behaviour. Models are described in a probabilistic extension of the Reactive Modules language and properties are expressed using rPATL, which extends the well-known logic ATL with operators to reason about probabilities, various reward-based measures, quantitative properties and precise bounds. The tool is based on the probabilistic model checker PRISM, benefiting from its existing user interface and simulator, whilst adding novel model checking algorithms for stochastic games, as well as functionality to synthesise optimal player strategies, explore or export them, and verify other properties under the specified strategy.

1 Introduction

Stochastic games are a natural model for systems that exhibit probabilistic behaviour and which contain components that may either compete or cooperate in order to achieve a certain goal. The model has a rich underlying theory and applications in areas as diverse as economics and biology. Stochastic games also have many applications in computer science. Game-theoretic models of competitive or collaborative behaviour can be used to model, for example, distributed systems, security protocols or sensor networks; furthermore, many such systems are inherently probabilistic, e.g. due to failures or randomisation.

For simpler model subclasses, various verification tools are available and widely used. For probabilistic models such as Markov chains or Markov decision processes and their variants, probabilistic model checking tools like PRISM [9] and MRMC [8] provide verification of quantitative properties in probabilistic temporal logics. For (non-stochastic) games, model checkers such as MCMAS [10] and MOCHA [1] verify properties in ATL or epistemic logics. GAVS+ [7] is a general-purpose algorithmic game solver which includes support for simple stochastic games. Game-based verification tools also have applications to scheduling and synthesis problems, for example using timed games [2], qualitative stochastic games [3] or mean-payoff games [4].

In this paper, we present *PRISM-games*, which is, to the best of our knowledge, the first tool to provide modelling and quantitative verification for *stochastic multi-player games* (SMGs). The games are specified using an extension of

the existing PRISM modelling language, which is a guarded-command based language inspired by the Reactive Modules formalism. Properties are specified in the temporal logic rPATL [5], which combines features of the multi-agent logic ATL, the probabilistic logic PCTL, as well as operators to reason about several different notions of reward/cost measures, numerical properties and precise probability values [6]. Currently, PRISM-games supports turn-based, perfect-information SMGs; future work will investigate efficient techniques for more general problem classes such as concurrent games and partial information. Turn-based games have, though, already proved to be sufficient to model, analyse and detect potential weaknesses systems in algorithms for energy management and collective decision making for autonomous systems [5].

PRISM-games builds upon the code-base of the existing PRISM model checker, extending existing features to provide a modelling language for stochastic multi-player games and a graphical user interface with model editor, discrete-event simulator and graph-plotting functionality. The core functionality of the new tool comprises novel methods for verifying quantitative properties of stochastic games [5,6]; and support for synthesising optimal player strategies, exploring or exporting them, and verifying other properties under the specified strategy.

2 Modelling Stochastic Multi-Player Games

A (turn-based) stochastic multi-player game (SMG) comprises a finite set of *players* and a finite set of *states*. In each state, exactly one player chooses (possibly randomly) from a set of available *probabilistic transitions* to determine the next state. To reason about SMGs, we use *strategies*, which determine the choices of transitions made by each player, based on the execution of the game so far.

In PRISM-games, SMGs are described in a modelling language similar to the Reactive Modules formalism. A model is composed of *modules*, whose state is determined by a set of *variables* and whose behaviour is specified by a set of *guarded commands*, containing an (optional) action label, a guard and a probabilistic update for the module's variables:

$$\texttt{[action] guard -> prob}_1\texttt{ : update}_1\texttt{ + ... + prob}_n\texttt{ : update}_n\texttt{;}$$

When a module has a command whose guard is satisfied in the current state, it can update its variables probabilistically, accordingly to the update. For action-labelled commands, multiple modules execute updates synchronously, if all their guards are satisfied. Each probabilistic transition in the model is thus associated with either an action label or a single module. A model also defines *players*, each of which is assigned a disjoint subset of the model's synchronising action labels and modules. This assigns each probabilistic transition to one player. Currently, to ensure a turn-based SMG, all possible probabilistic transitions in a state must belong to the same player; the tool detects and disallows concurrent actions.

An excerpt from a PRISM-games model of *futures market investors* is shown in Fig. 1. There are 3 players: 2 investors and the market. At the start of each month, *investors* decide whether to invest or not; the *market* can decide to bar

```
smg

// Player definitions
player investor1
    [invest1], [noinvest1], [cashin1]
endplayer
player investor2
    [invest2], [noinvest2], [cashin2]
endplayer
player market
    [nobar], [bar1], [bar2], sched, [month], [done]
endplayer

// Investor 1
module investor1
    // State: 0 = no reservation
    // 1 = made reservation
    // 2 = finished
    i1 : [0..2];
    // Decide whether invest or not
    [noinvest1] i1=0 | i1=1 & b1=1 → (i1'=0);
    [invest1] i1=0 | i1=1 & b1=1 → (i1'=1);
    // Cash in shares (if not barred)
    [cashin1] i1=1 & b1=0 → (i1'=2);
    // Finished
    [done] i1=2 | term=1 → true;
endmodule

// Investor 2
module investor2 = investor1 [ ... ] endmodule
```

```
// Market
module market
    // State: 0 = !barred, 1 = barred
    b1 : [0..1] init 1;
    b2 : [0..1] init 1;
    // Share value
    v : [0..vmax] init vinit;
    // Bar one or none of the investors
    [nobar] true → (b1'=0) & (b2'=0);
    [bar1] b1=0 → (b1'=1) & (b2'=0);
    [bar2] b2=0 → (b2'=1) & (b1'=0);
    // Share price movement
    [month] true → p/10 : (v'=up)
                + (1 − p/10) : (v'=down);
endmodule

// Scheduling module
module sched
    // Turn-based scheduling of players
endmodule

// Reward: Shares collection value
// for investor1, and both investors
rewards "profit1"
    [cashin1] i1=1 : v;
endrewards
rewards "profit12"
    [cashin1] i1=1 : v;
    [cashin2] i2=1 : v;
endrewards
```

Fig. 1: Excerpt from a three-player game modelling *futures market investors*.

one of the investors from investing and also picks the order in which the investors take decisions. Share price fluctuations are modelled as a random process. The full model, along with several larger examples, is available from [11].

3 Property Specification

PRISM-games' property specification language is based on rPATL [5]. rPATL is a CTL-style branching-time temporal logic used to express properties of SMGs, which combines the coalition operator $\langle\langle C\rangle\rangle$ of ATL, the probabilistic operator $\mathsf{P}_{\bowtie q}$ from PCTL, and an operator $\mathsf{R}^r_{\bowtie x}$ for reasoning about several types of expected reward/cost measures. The syntax of rPATL is given by the grammar:

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

where a is an atomic proposition used to label SMG states, C is a *coalition* (a set of players), $\bowtie \in \{<, \leq, \geq, >\}$, $q \in \mathbb{Q} \cap [0,1]$, $x \in \mathbb{Q}_{\geq 0}$, r is a *reward structure* mapping states to non-negative rationals, $\star \in \{0, \infty, c\}$ and $k \in \mathbb{N}$.

An example rPATL formula is $\langle\langle\{1,2\}\rangle\rangle \mathsf{P}_{\geq 0.75}[\mathsf{F}^{\leq 5} \mathit{goal}]$, which means "players 1 and 2 have a (combined) strategy to ensure that the probability of reaching

a '*goal*' state within 5 steps is at least 0.75, regardless of the strategies of other players in the game". The $\langle\langle C\rangle\rangle \mathsf{R}^{r}_{\bowtie x}[\mathsf{F}^{\star}\phi]$ operator is used in a similar fashion, but is annotated with a reward structure r and a type $\star \in \{0, \infty, c\}$. It states that coalition C has a strategy to ensure that expected amount of reward r cumulated until a ϕ-state is reached satisfies $\bowtie x$. The type $\star$ allows us to treat the case where ϕ is *not* reached differently, assigning zero reward ($\star$=0), infinite reward ($\star$=∞) or allowing reward to accumulate indefinitely ($\star$=c).

We support several extensions of rPATL, including 'quantitative' (numerical) operators, e.g., $\langle\langle C\rangle\rangle \mathsf{P}_{\max=?}[\psi]$, which gives the maximum probability of ψ that coalition C can guarantee, instead of a true/false value. PRISM-games also supports *precise* value operators $\langle\langle C\rangle\rangle \mathsf{P}_{=q}[\psi]$ and $\langle\langle C\rangle\rangle \mathsf{R}^{r}_{=x}[\mathsf{F}^{c}\phi]$ for *stopping* games (i.e., stochastic games where terminal states are reached with probability 1 under any pair of strategies), using the model checking algorithms of [6].

Examples. Some sample properties for the *futures market investor* model from the previous section (see Fig. 1) are provided below.

- $\langle\langle\{\text{investor1,investor2}\}\rangle\rangle \mathsf{R}^{\text{profit12}}_{\geq 10}[\mathsf{F}^{c}(\mathit{done1} \wedge \mathit{done2})]$ – "the two investors have a joint strategy guaranteeing them an expected profit of at least 10";
- $\langle\langle\{\text{investor1,market}\}\rangle\rangle \mathsf{R}^{\mathit{profit1}}_{\max=?}[\mathsf{F}^{c}\mathit{done1}]$ – "what is the maximum expected profit that investor 1 can achieve with the help of the market?";
- $\langle\langle\{\text{investor1,investor2}\}\rangle\rangle \mathsf{R}^{\text{profit12}}_{=5}[\mathsf{F}^{c}\mathit{done}]$ – "both investors can collaborate to achieve an expected profit of precisely 5";
- $\langle\langle\{\text{investor1}\}\rangle\rangle \mathsf{P}_{\max=?}[\mathsf{F}\,(\mathit{done1} \wedge v > 5)]$ – "what is the maximum probability with which investor 1 can guarantee a share value greater than 5?"

4 Synthesis and Analysis of Strategies

Reasoning about *strategies* is an essential aspect of modelling and verifying games. rPATL queries check for the existence of a strategy that satisfies a given probability/reward bound or which optimises some objective. When PRISM-games model checks such properties, it also supports *strategy synthesis*, allowing the user to obtain a corresponding satisfying/optimal strategy.

An SMG strategy resolves the choices in each state, using a (possibly infinite) set of *memory elements*, each representing a possible "state" of the strategy. The memory element is updated (possibly stochastically) at each transition, and the action picked by the player is determined by the current memory element and the current state. A strategy is *memoryless* if it has only one memory element, and *finite-memory* if there are finitely many. It is *deterministic* if the functions that update memory elements and pick actions are not probabilistic, and *stochastic-update* otherwise. Currently, PRISM-games supports three types of strategies: *memoryless deterministic*, *finite-memory deterministic* and *finite-memory stochastic-update*.

Strategies can be analysed manually in the simulator view or exported to files (see Fig. 2 for screenshots). PRISM-games also supports 'implementation'

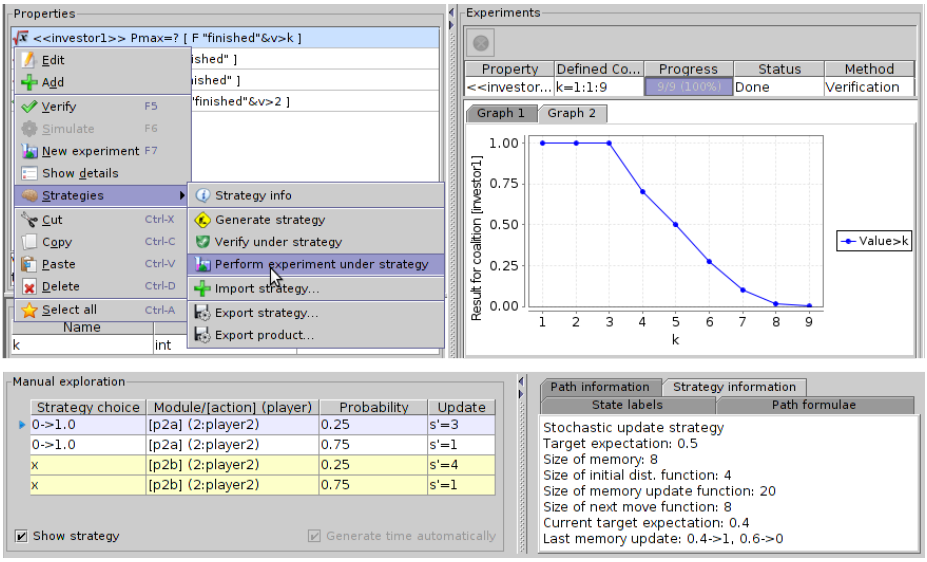

Fig. 2: PRISM-games screenshots: simulation of a synthesised strategy (bottom) and verification of a property under the strategy (top).

of strategies – the product of a strategy and the original game can be built, resulting in a new model on which other properties can be verified. For example, in a game with 3 players we can generate a strategy for player 1 specified by $\langle\langle 1 \rangle\rangle \mathsf{P}_{\geq 0.5}[\mathsf{F}\ \mathit{goal1}]$. Implementing this strategy would then result in a two-player game on which further properties may be verified, e.g., in rPATL formula $\langle\langle 2,3 \rangle\rangle \mathsf{P}_{\geq 0.99}[\mathsf{F}\ \mathit{goal2}]$, player 1 now does not minimise the probability of reaching a *goal2* state; instead its strategy is fixed to one which achieves the first rPATL formula. Strategy import functionality is also supported.

5 Algorithms and Further Details

Underlying algorithms. The core parts of the model checking algorithm for rPATL are based on reductions to two-player stochastic games, by constructing a *coalition game* where player 1 plays represents the coalition C from the rPATL formula being verified and player 2 is all other players. The basic techniques for solving games formulate systems of equations over $+, \cdot, \max, \min$, and then perform *value iteration* to compute their least or greatest solutions. This works directly for $\langle\langle C \rangle\rangle \mathsf{P}_{\bowtie q}[\psi]$ and $\langle\langle C \rangle\rangle \mathsf{R}^r_{\bowtie x}[\mathsf{F}^\star \phi]$ where $\star \in \{c, \infty\}$. For $\star = 0$, the optimal strategy may depend on the reward accumulated so far and so is not memoryless. Here, we compute a bound after which the optimal strategy picks actions that maximise the probability of reaching ϕ-states, and reduce the problem to previous cases; see [5] for details. For the *precise* value operators $\langle\langle C \rangle\rangle \mathsf{P}_{=q}[\psi]$ and $\langle\langle C \rangle\rangle \mathsf{R}^r_{=x}[\mathsf{F}^c \phi]$, we need to compute the sup inf and inf sup values

Case study		SMG statistics			Model checking		
		Players	States	Transitions	Property type	Constr.	Check
mdsm [*N*]	5	5	743,904	2,145,120	$\langle\langle C\rangle\rangle \mathsf{R}^{r}_{\max=?}[\mathsf{F}^{0}\,\phi]$	14.5s	61.9s
	7	7	6,241,312	19,678,246		210.7s	1,054.8s
cdmsn [*N*]	3	3	1,240	1,240	$\langle\langle C\rangle\rangle \mathsf{P}_{\bowtie q}[\,\mathsf{F}^{\leq k}\,\phi]$	0.2s	0.2s
	5	5	100,032	843,775		3.2s	6.4s
investor [*vmax*]	10	2	10,868	34,264	$\langle\langle C\rangle\rangle \mathsf{R}^{r}_{\min=?}[\mathsf{F}^{c}\,\phi]$	1.4s	0.7s
	200	2	2,931,643	9,688,354		45.9s	820.8s
team-form [*N*]	3	3	17,041	20,904	$\langle\langle C\rangle\rangle \mathsf{P}_{\max=?}[\mathsf{F}\,\phi]$	0.3s	0.5s
	5	5	2,366,305	2,893,536		36.9s	12.9s

Table 1: Performance statistics for a representative set of models [5,11].

for the property (using rPATL model checking algorithms). We then construct a finite-memory stochastic-update strategy based on the results of [6].

Implementation and availability. The model checking implementation is currently built upon PRISM's "explicit" engine, which uses Java-based explicit-state data structures (sparse matrices, bit-sets, etc.). Illustrative experimental results are shown in Table 1, for games of the order 10^6-10^7 states (run on a 2.8GHz PC with 32GB of RAM). A symbolic (BDD-based) implementation is under development to offer improved scalability on models exhibiting regularity.

PRISM-games is released as open source software, currently licensed under the GPL. The tool, with documentation and examples, is available from [11].

Acknowledgments. The authors are part supported by ERC Advanced Grant VERIWARE, the Institute for the Future of Computing at the Oxford Martin School, EPSRC grant EP/F001096/1 and a Royal Society Newton Fellowship.

References

1. 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)
2. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Proc. CAV'07. LNCS, vol. 4590 (2007)
3. Chatterjee, K., Henzinger, T., Jobstmann, B., Radhakrishna, A.: Gist: A solver for probabilistic games. In: Proc. CAV'10. pp. 665–669. LNCS, Springer (2010)
4. Chatterjee, K., Henzinger, T., Jobstmann, B., Singh, R.: QUASY: Quantitative synthesis tool. In: Proc. TACAS'11. LNCS, vol. 6605. Springer (2011)
5. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. In: Proc. TACAS'12. pp. 315–330 (2012)
6. Chen, T., Forejt, V., Kwiatkowska, M., Simaitis, A., Trivedi, A., Ummels, M.: Playing stochastic games precisely. In: Proc. CONCUR'12. pp. 348–363 (2012)
7. Cheng, C., Knoll, A., Luttenberger, M., Buckl, C.: GAVS+: An open platform for the research of algorithmic game solving. Proc. TACAS'11 pp. 258–261 (2011)
8. Katoen, J.P., Hahn, E.M., Hermanns, H., Jansen, D., Zapreev, I.: The ins and outs of the probabilistic model checker MRMC. In: Proc. QEST'09. IEEE (2009)
9. 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)
10. 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)
11. PRISM-games website, `http://www.prismmodelchecker.org/games/`