PRISM-games 3.0: Stochastic Game Verification with Concurrency, Equilibria and Time

Marta Kwiatkowska[1], Gethin Norman[2], David Parker[3](✉), and Gabriel Santos[1]

[1] Department of Computing Science, University of Oxford, Oxford, UK
[2] School of Computing Science, University of Glasgow, Glasgow, UK
[3] School of Computer Science, University of Birmingham, Birmingham, UK
d.a.parker@cs.bham.ac.uk

Abstract. We present a major new release of the PRISM-games model checker, featuring multiple significant advances in its support for verification and strategy synthesis of stochastic games. Firstly, *concurrent* stochastic games bring more realistic modelling of agents interacting in a concurrent fashion. Secondly, *equilibria*-based properties provide a means to analyse games in which competing or collaborating players are driven by distinct objectives. Thirdly, a *real-time* extension of (turn-based) stochastic games facilitates verification and strategy synthesis for systems where timing is a crucial aspect. This paper describes the advances made in the tool's modelling language, property specification language and model checking engines in order to implement this new functionality. We also summarise the performance and scalability of the tool, and describe a selection of case studies, ranging from security protocols to robot coordination, which highlight the benefits of the new features.

1 Introduction

Quantitative verification and strategy synthesis are powerful techniques for the modelling and analysis of computerised systems which require reasoning about *quantitative* aspects such as probability, time or resource usage. They can be used either to produce formal *guarantees* about a system's behaviour, for example relating to its safety, reliability or efficiency, or to synthesise controllers which ensure that such guarantees will be met at runtime. Examples of applications where these techniques have been used include power controllers, unmanned aerial vehicles, autonomous driving and communication protocols.

As computing systems increasingly involve concurrently acting autonomous agents, *game-theoretic* approaches are becoming widespread in computer science as a faithful modelling abstraction. These techniques can be used to reason

S. K. Lahiri and C. Wang (Eds.): CAV 2020, LNCS 12225, pp. 475–487, 2020.
https://doi.org/10.1007/978-3-030-53291-8_25

about the *competitive* or *collaborative* behaviour of multiple rational agents or entities with distinct goals or objectives. Applications include designing a defence strategy against attackers in a cybersecurity context or building controllers for autonomous robots operating in an unknown or potentially malicious environment. More broadly, game theory techniques such as mechanism design can be used to design protocols that are robust in the context of selfish participants, for example by incorporating *incentive/reward* schemes. They have been successfully deployed in diverse contexts such as network routing [29], auction design [10], public good provisioning [15] and ranking or recommender systems [30].

However, designing game-theoretic systems correctly is a challenge, in view of the complexity of behaviours arising from the interactions between autonomy, concurrency and quantitative rewards. This motivates the development of formal verification techniques to check their correctness and synthesise correct-by-construction strategies for them. Furthermore, many of these applications require reasoning about *stochasticity*: protocols may employ randomisation, e.g., for reliable dissemination across a network, or to minimise the impact of information leakage to an observer; autonomous robots operate in uncertain environments and may use unreliable hardware components or noisy sensors; and data-driven systems such as ranking or navigation systems rely on learnt probabilistic models for their execution.

These challenges have inspired the development of PRISM-games [22], a model checking tool for *stochastic games*. To date, it supports verification and strategy synthesis for *turn-based* stochastic multi-player games (TSGs) using a variety of objectives, expressed in the temporal logic rPATL (probabilistic alternating-time temporal logic with rewards) [8]. This allows specification of *zero-sum* objectives relating to one coalition of players trying to maximise a probabilistic or reward-based objective, while the remaining players form a second coalition trying to minimise the objective. It has also been extended to include (zero-sum) *multi-objective* properties and additional reward measures such as *long-run average* and *ratio reward* [22]. These methods have been successfully applied to several case studies such as autonomous vehicles, user-centric networks, temperature control and an aircraft electric power system [21,23,32].

In this paper, we present PRISM-games 3.0, which significantly extends its predecessor's functionality in several ways [18–20]. First, it supports the modelling and analysis of *concurrent stochastic multi-player games* (CSGs). Previous versions of the tool supported TSGs, in which it is assumed that each state of the game is controlled by a specific player. CSGs allow players to make decisions simultaneously, without knowledge of each other's choices, providing a more realistic model of concurrent execution and decision making. For this, we extend the PRISM-games modelling language, allowing the user to specify concurrency and synchronisation among agents, as well as to associate rewards to either joint or single actions.

In the first instance, PRISM-games now supports verification and strategy synthesis for CSGs using zero-sum specifications in rPATL [19], which we extend to accommodate *instantaneous rewards*. The second major addition to the tool is

the possibility of reasoning about *equilibria-based* properties, which allow players to have distinct, not necessarily conflicting objectives. We extend rPATL to express properties relating to (subgame perfect) *social-welfare optimal Nash equilibria (SWNE)* [20]. This provides synthesis of strategies for all players (or coalitions) from which there is no incentive for any of them to unilaterally deviate in any state of the game, and where the combined probabilities or rewards are maximised (or minimised).

Thirdly, PRISM-games now adds support for *probabilistic timed multi-player games* (TPTGs) [18] (currently just the turn-based variant of the model). These extend stochastic multi-player games with real-valued clocks, in the style of (probabilistic) timed automata. This allows real-time aspects of a system to be more accurately modelled. Using the *digital clocks* approach [18], timed models are automatically translated to discrete-time models in order to be verified.

In this paper, we describe the key enhancements made to the tool, notably to its modelling and property specification languages. We also summarise the results, algorithms and implementation of the verification and strategy synthesis techniques developed [18–20] to support the new functionality. We then describe a selection of case studies which showcase the advantages of the new features, and summarise the performance and scalability of the tool.

PRISM-games is open source and runs on all major operating systems. It is available from the tool's website [34]. Supporting material for the paper, including a virtual machine that allows easy running of the tool and reproduction of the results presented in Sect. 4, can be found at [33].

Related Tools. Other model checking tools have been developed to provide support for games. For non-stochastic games, model checking tools such as PRALINE [5], EAGLE [31] and EVE [16] support Nash equilibria [27], as does MCMAS-SLK [6] via strategy logic. Uppaal Stratego [11] is a tool that uses machine learning, model checking and simulation for the synthesis of strategies for stochastic priced timed games. GAVS+ [9] is a general-purpose tool for algorithmic game solving, supporting TSGs and (non-stochastic) concurrent games, but not CSGs. GIST [7] allows the analysis of ω-regular properties on probabilistic games, but again focuses on turn-based, not concurrent, games. General purpose tools such as Gambit [26] can compute a variety of equilibria but not for stochastic games.

2 Modelling and Property Specification Languages

2.1 Modelling Concurrent and Timed Games

The new features in PRISM-games 3.0 have required some significant enhancements to the language used to specify models. For the addition of real-time aspects (i.e., TPTGs), the changes are a straightforward combination of the existing language features for specifying TSGs in PRISM-games (player specifications and mapping of model states to them) and for probabilistic timed automata in PRISM (clock variables, module invariants, guards and clock resets).

We therefore focus in this paper on the specification of CSGs, where the language changes are more fundamental.

PRISM-games has an existing language for specifying TSGs, which is an extension of the native PRISM modelling language [22]. Components of the system to be modelled are encapsulated as *modules*, whose states are defined by a set of finite-range *variables* and whose behaviour is specified using action-labelled *guarded commands*. In a state, one or more modules can execute a command to make a transition: if the guard (a predicate over state variables) is satisfied, the state can be modified (probabilistically) by applying the *updates* of the command. Multiple modules can execute simultaneously if their commands are labelled with the same action.

```
csg
// Player specification
player p1 mac1 endplayer
player p2 mac2 endplayer
// Max energy per user
const int emax;
// User 1
module mac1
        s1 : [0..1] init 0; // Has user 1 sent?
        e1 : [0..emax] init emax; // Energy level of user 1
        [w1] true -> (s1'=0); // Wait
        [t1] e1>0 -> (s1'=c'?0:1) & (e1'=e1-1); // Transmit
endmodule
// Define second user using module renaming
module mac2 = mac1 [ s1=s2, e1=e2, w1=w2, t1=t2 ] endmodule
```

```
// Probability qi for transmission success when i users send
const double q1;
const double q2;
// Channel (computes joint transmission probabilities)
module channel
        c : bool init false; // Did a collision occur during transmission?
        [t1,w2] true -> q1:(c'=false) + (1-q1):(c'=true); // User 1 transmits
        [w1,t2] true -> q1:(c'=false) + (1-q1):(c'=true); // User 2 transmits
        [t1,t2] true -> q2:(c'=false) + (1-q2):(c'=true); // Both transmit
endmodule
```

```
// Reward structures
rewards "mess1" // Number of messages sent by user 1
        s1=1 : 1;
endrewards
rewards "mess2" // Number of messages sent by user 2
        s2=1 : 1;
endrewards
rewards "send2" // Number of times users 1 and 2 transmit simultaneously
        [t1,t2] true : 1;
endrewards
```

Fig. 1. An example PRISM-games 3.0 CSG model of medium access control.

CSGs cannot naturally be modelled with this approach for several reasons: (i) players need to be able to concurrently choose between multiple commands with different action labels; (ii) the update performed by one player may be different depending on the action chosen by another player; (iii) when multiple

players execute, variables may need to be updated according to an arbitrary probability distribution, rather than being limited to the product of separate distributions specified locally by individual modules.

Figure 1 shows an example of the PRISM-games 3.0 modelling language, which we use to illustrate some of its new features. It models a probabilistic version of the *medium access control* problem, previously described in [5]. Two users share a communication channel. At each time step, user `maci` (`i` = 1, 2) can choose between transmitting a message (`ti`) or waiting (`wi`). Variable `si` tracks whether a user successfully sent its message in the last time step and `ei` represents its energy level: transmissions can only occur when energy is positive. A third component is the channel `channel`, modelled by Boolean variable `c` denoting whether a collision occurred on the last transmission attempt.

The first difference (with respect to modelling of TSGs) is the player specification: players are associated with modules (rather than states). In the example, module `maci` constitutes player `i`. Modules with no nondeterministic choice (like `channel`) do not need to be tied to a player.

In each state of the CSG, each player chooses between enabled commands of the corresponding modules; if no command is enabled, the player idles. The players move simultaneously so transitions are labelled with *lists* of action labels $[\mathtt{a}_1, \ldots, \mathtt{a}_n]$. So the guarded command notation is extended accordingly: note how the channel's behaviour depends on which actions the two users take (the same principle applies when specifying reward structures; see `send2`). Furthermore, variable updates within a command can now be dependent on the updated values of other variables, provided there are no cyclic dependencies. See for example `(s1'=c'?0:1)`, which updates `s1` depending on whether there was a channel collision (reflected in `c'`, the updated value of `c`). We use this mechanism to model interference on the channel: module `channel` specifies a joint probability distribution which is used to update variables `s1` and `s2` simultaneously.

2.2 Property Specification

PRISM-games 3.0 also extends the language used to specify properties for verification and strategy synthesis. The previous version already supported *zero-sum* queries for TSGs using the logic rPATL, which combines the game logic ATL with reward-based extensions of the probabilistic logic PCTL. Again, for the new real-time models, it is relatively easy to combine the existing rPATL notation with real-valued time bounds. So, we focus here on the case of CSGs, and in particular *equilibria-based* properties.

We compute values or synthesise strategies which are *social-welfare optimal Nash equilibria (SWNE)*, i.e., which maximise (or minimise) the sum of the values associated to the objectives for each player, but from which there is no incentive for any of them to unilaterally deviate in any state of the game. We express such properties by adding to rPATL the + operator, which is then used to denote the sum of the values associated to both *bounded* and *unbounded* objectives.

When using the rewards operator in equilibria-based properties, we can reason about *cumulative* ($\mathtt{C}^{\leqslant \mathtt{k}}$), *instantaneous* ($\mathtt{I}^{=\mathtt{k}}$) and *expected reachability* ($\mathtt{F}$)

objectives. For properties with the probability operator, we support bounded and unbounded reachability using the temporal operators *next* (`X`), *eventually* (`F`) and *until* (`U`). In order to express zero-sum properties for CSGs, we have implemented all the previous temporal operators for probabilistic queries and a subset of the rPATL operators reported in [8] for reward-based queries, adding to that the instantaneous reward operator.

Finally, following the style of rPATL we separate players into *coalitions* with the syntax $\langle\langle coalition \rangle\rangle$, in order to specify the player or association of players for which we seek to maximise or minimise the values for a given zero-sum property. For equilibria-based properties, given that we maximise/minimise the sum, we use the same operator to separate players in different coalitions using a colon, while players in the same coalition are separated by a comma.

The following are examples of both zero-sum and equilibria-based properties for the medium access CSG model described in Fig. 1.

- $\langle\langle$`p1`$\rangle\rangle$`P`$_{\max=?}$`[ s2=0 U s1=1 ]` – what is the maximum probability user 1 can ensure of being the first to transmit, regardless of the behaviour of user 2?
- $\langle\langle$`p2`$\rangle\rangle$`R`$^{\texttt{mess2}}_{\geqslant 2.0}$`[ F e2=0 ]` – can user 2 ensure the expected number of messages it sends before running out of energy is at least 2, whatever user does?
- $\langle\langle$`p1:p2`$\rangle\rangle_{\max\geqslant 2}$(`P[ F s1=1 ] + P[ F s2=1 ]`) – if each user's objective is to send their packet with the maximum probability, is it possible for them to collaborate and both transmit their packets with probability 1?
- $\langle\langle$`p1:p2`$\rangle\rangle_{\max=?}$(`P[ s2=0 U s1=1 ] + P[ s1=0 U s2=1 ]`) – what is the sum of SWNE values if each user tries to maximise the probability of being the first to successfully transmit?
- $\langle\langle$`p1:p2`$\rangle\rangle_{\max=?}$(`R`$^{\texttt{mess1}}$`[ F e1=0 ] + R`$^{\texttt{mess2}}$`[ C`$^{\leqslant k}$`]`) – what is the sum of SWNE values if user 1 tries to maximise the expected number of packets before running out of energy and user 2 maximises the expected number of packets in the first k steps?

3 Verification and Strategy Synthesis Algorithms

3.1 Zero-Sum Properties for CSGs

When verifying zero-sum properties of CSGs, PRISM-games makes use of the model checking algorithms described in [19], which were based on the methods formulated in [2,3]. We rely on *value iteration* and classical convergence criteria to approximate/compute the values for all states of the game under study, and on solving a *linear program* to compute a *minimax* strategy at each state. This corresponds to solving a *matrix game*, which represents a *one-shot zero-sum* game for the actions of each player in a state. For unbounded properties, the solutions of the matrix games are used to synthesise an optimal (memoryless and randomised) strategy for each player. Prior to this numerical solution phase, we find and remove the states for which the optimal expected reward values are infinite by using the qualitative algorithms developed in [1].

Our current implementation uses the LPsolve [24] library to solve the matrix games at each state. CSGs are built and stored in a explicit-state fashion using an extension of PRISM's Java-implemented *explicit* (sparse-matrix based) engine.

3.2 Equilibria-Based Properties for CSGs

For equilibria-based properties of CSGs, PRISM-games implements the methods described in [20]. We rely on value iteration and *backwards induction* to approximate/compute values and synthesise strategies that are SWNE. For unbounded properties, we can only compute values that are ε-Nash equilibria, since Nash equilibria are not guaranteed to exist. At each state, we solve a *bimatrix game*, which is a representation of a *one-shot nonzero-sum* game and is a linear complementarity problem. We solve these games via *labelled polytopes*, finding all equilibria values through an SMT-based implementation, for which we use third-party SMT solvers Z3 [12] and Yices [13]. We make use of a precomputation step of finding and removing *dominated strategies* in order to minimise the number of calls to the solver.

Unlike zero-sum properties, the synthesised strategies for bounded and unbounded equilibria-based properties require (finite) memory. This is needed due to the fact that a player's choices may change once their objectives have been satisfied. We synthesise strategies by combining the strategy vectors computed for each bimatrix game and the strategy generated by computing optimal values for the MDP resulting from playing the game after either goal has been met. As we use value iteration to approximate values for infinite-horizon properties, we can only synthesise ε-Nash strategy profiles.

3.3 Turn-Based Probabilistic Timed Games

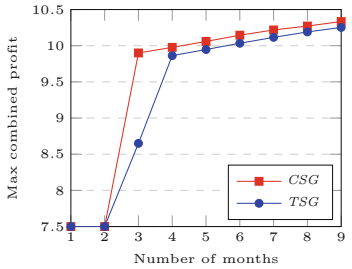
Verification and strategy synthesis of TPTGs relies on the algorithms from [18], which use the *digital clocks* approach that has been a developed for a variety of real-time models. A translation, at the level of the PRISM-games modelling languages, automatically converts the problem of analysing a TPTG into one of solving a (discrete-time) TSG, for which PRISM-games's existing engines can be used. Time-bounded properties are handled by automatically integrating a timing clock into the model prior to translation. As in the rest of PRISM-games, TSGs are also built and solved using the Java-based *explicit* engine.

4 Case Studies and Experimental Results

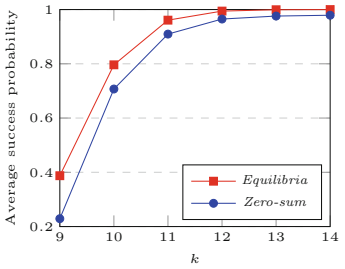
The features added in PRISM-games 3.0 have been used for over 10 new case studies across a wide range of application domains, including computer security (intrusion detection, radio jamming, non-repudiation), communication protocols (medium access control, Aloha), incentive schemes for cooperative networking, multi-robot navigation problems and processor task scheduling. Details can be found in [18–20] and on the case studies section of the PRISM-games website [35].

Supporting material is at [33]. In this section, we showcase four selected case studies that demonstrate the benefits of the tool's new functionality. We also include a discussion of the scalability and performance of the tool.

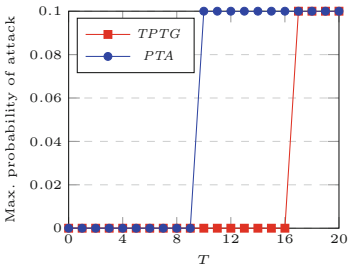
Future Markets Investor. This example models two investors playing against the stock market. Investors choose when to invest or to cash in, and the stock market can decide to bar investments at certain points; fluctuations in share values are modelled stochastically. PRISM-games can, for example, synthesise optimal strategies for the two investors to maximise their expected joint profit over time, acting against the stock market which aims to minimise it.



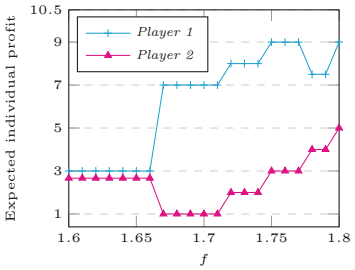
(a) **Future markets investor:** avoiding unrealistic strategy choices using CSGs



(b) **Robot coordination:** using equilibria for mutually beneficial navigation plans



(c) **Non-repudiation:** Attack & defence strategies in a timed, randomised protocol



(d) **Public good game:** Tuning incentive parameter f by synthesising equilibria

Fig. 2. Results illustrating the benefits of the new verification and strategy synthesis techniques implemented in PRISM-games 3.0; see Sect. 4 for details. (Color figure online)

Figure 2(a) shows the results obtained for this property using both a *turn-based* stochastic game (TSG) and a *concurrent* stochastic game (CSG). The former leads to unrealistic modelling as the market can see the choices made by the investors and gain an unfair advantage: the values in the blue plot in Fig. 2(a) are artificially low. In the CSG model, using PRISM 3.0, decisions are taken simultaneously, yielding the correct strategies and values (red plot).

Robot Coordination. Our next example models two robots navigating in opposite directions across a 10-by-10 grid as a CSG. Obstacles which hinder the robots as they move from location to location are modelled stochastically; and if the robots collide, both of them fail in their attempt to reach their goal. We use PRISM-games to find navigation strategies for the two robots, where each robot does not know the choice being made by the other at each step.

The objective for each robot is to navigate successfully, so we maximise the average probability (across the two robots) of success. Figure 2(b) shows the best value that can be achieved within a fixed period of k moves across the grid. One robot aiming single-handedly to achieve this goal performs reasonably well (blue plot), but we can achieve better collective performance by using PRISM-games to synthesise a (social welfare Nash) *equilibrium* strategy (red plot).

Non-repudiation. Next we consider a non-repudiation protocol [25], which permits an originator O to transfer information to a recipient R while guaranteeing non-repudiation, i.e., that neither O nor R can deny that they participated in the transfer. Here, both *probability* (the protocol is randomised) and *time* (the protocol relies on acknowledgement time-outs) are essential ingredients for checking correctness. Furthermore, we model the two participants of the protocol as opposing players, resulting in a TPTG model.

To verify the protocol, we check the worst-case probability that a malicious recipient R can obtain the information being transferred within time T. This can be done with a PTA model (as in [28]) but, with a timed game model, we can also analyse counter-strategies of the honest participant. The results (see Fig. 2(c)) show that, while it is not possible to prevent the information being received, it *is* possible to delay it (the red plot shows lower probabilities for higher times). Note that the bound T is an actual time bound, unlike the examples above, where step-bounded properties measure the number of steps or rounds.

Public Good Game. Lastly, we show a new case study modelling a *public good game*, a well studied model of social choice in economics where participants repeatedly decide how much of an endowment to keep for themselves or to share it with the other players. The total shared by the players is boosted by a factor f in order to incentivise sharing and then divided equally between the players.

Figure 2(d) shows results from a 2-player game, modelled as a CSG. Player choices are necessarily *concurrent*, to avoid cheating. We also need to use *equilibria* since the players have distinct individual goals (maximising personal expected profit). Figure 2(d) shows the values for each player in a synthesised optimal (social welfare Nash) equilibrium for varying f. Changes in f affect both the resulting profit *and* potential inequalities between players in equilibria, indicating the subtleties involved when tuning parameters in an incentive mechanism and the usefulness of analysing this with PRISM-games.

Scalability and Performance. Finally, we show some experimental results for a representative selection of larger examples, to give an indication of the scalability and performance of PRISM-games 3.0. Table 1 shows a range of models (the first 4 are CSGs; the last is a TPTG), the statistics for each one (number of

Table 1. Model statistics for some of the case studies.

Case study	Players	States transitions	Constr. time(s)	Property	Verif. time(s)
Robot coordination	2	159,202 10,765,010	30.94	$\langle\langle p_1 \rangle\rangle \mathtt{P}_{\max=?}[\,\neg \mathtt{c}\, \mathtt{U}^{\leqslant k} \mathtt{g}_1\,]$	114.5
	2	159,202 10,765,010	39.00	$\langle\langle p_1{:}p_2 \rangle\rangle_{\max=?}(\mathtt{P}[\,\neg \mathtt{c}\, \mathtt{U}^{\leqslant k} \mathtt{g}_1\,]{+}\mathtt{P}[\,\neg \mathtt{c}\, \mathtt{U}^{\leqslant k} \mathtt{g}_2\,])$	1,080
Future markets investors	3	1,398,441 7,374,616	51.2	$\langle\langle i_1 \rangle\rangle \mathtt{R}_{\max=?}[\,\mathtt{F}^{c}\ \mathsf{cashed}_1\,]$	1,030
	3	478,761 2,265,560	13.47	$\langle\langle i_1{:}i_2 \rangle\rangle_{\max=?}(\mathtt{R}[\mathtt{F}\ \mathtt{c}_1\,]{+}\mathtt{R}[\mathtt{F}\ \mathtt{c}_2\,])$	13,110
User-centric networks	7	2,993,308 11,392,196	198.6	$\langle\langle \mathit{user} \rangle\rangle \mathtt{R}_{\max=?}[\,\mathtt{F}^{c}\ \mathit{services}{=}K\,]$	1,061
Aloha	3	556,168 2,401,113	15.7	$\langle\langle p_2, p_3 \rangle\rangle \mathtt{R}_{\min=?}[\,\mathtt{F}\ \mathsf{sent}_{2,3}\,]$	317.8
	3	3,334,681 17,834,254	146.1	$\langle\langle p_1{:}p_2{,}p_3 \rangle\rangle_{\min=?}(\mathtt{R}[\mathtt{F}\ \mathtt{s}_1\,]{+}\mathtt{R}[\mathtt{F}\ \mathtt{s}_{2,3}\,])$	3,129
Task graph scheduling	2	659,948 1,798,198	11.16	$\langle\langle \mathit{sched} \rangle\rangle \mathtt{R}_{\max=?}[\,\mathtt{F}\ \mathit{done}\,]$	89.7

players, states, transitions) and the time taken to build and verify the model for some example properties on a 2.10 GHz Intel Xeon with 8 GB of JVM memory.

Verification of CSGs is more computationally expensive than for TSGs supported in earlier versions of the tool, but PRISM-games 3.0 is able to build and analyse CSGs with more than 3 million states on relatively modest hardware. The majority of the time is spent solving (bi)matrix games, which is done repeatedly for all states of the model. Hence, the number of choices per state, which dictates the size of these games, has a greater impact on performance than for TSGs. Unsurprisingly, equilibria properties are slower than zero-sum ones. For both types of property, the number of players in the game does not have a major impact since they are grouped into coalitions yielding a 2-player game to solve. For TPTGs, the digital clocks translation is fast since it is done syntactically, and then a TSG is solved whose size depends on several factors, primarily the number of locations and the magnitude of any time bound in the property.

5 Conclusions

We have presented PRISM-games 3.0, which adds three major new features: (i) concurrent stochastic games; (ii) synthesis of equilibria; and (iii) timed probabilistic games. The usefulness of these has been illustrated on several newly created or extended applications.

CSGs are considerably more expensive to solve than their turn-based counterparts and a key challenge is efficiently solving the matrix game at each state, which is itself a non-trivial optimisation problem. For equilibria, the main difficulty is finding an optimal equilibrium, which currently relies on iteratively restricting the solution search space. Both problems are sensitive to the limitations and issues of floating-point arithmetic, particularly equilibria computation, and might benefit from arbitrary precision representations. Recent research has

also pointed out the shortcomings of only using a lower bound approximation as a stopping criterion for value iteration, as it can lead to inaccuracies [4,14,17]. The impact of similar issues on model checking for games is still to be studied.

A range of further challenges exist for future work. These include providing support for *multi-coalitional* properties and implementing other techniques for equilibria computation. For timed games, we plan to investigate concurrent variants, and also zone-based solution techniques. More broadly speaking, partial information variants of games would be a useful addition.

Acknowledgements. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 834115) and the EPSRC Programme Grant on Mobile Autonomy (EP/M019918/1).

References

1. de Alfaro, L., Henzinger, T.: Concurrent omega-regular games. In: LICS 2000, pp. 141–154 (2000)
2. de Alfaro, L., Henzinger, T., Kupferman, O.: Concurrent reachability games. Theor. Comput. Sci. **386**(3), 188–217 (2007)
3. de Alfaro, L., Majumdar, R.: Quantitative solution of omega-regular games. J. Comput. Syst. Sci. **68**(2), 374–397 (2004)
4. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: interval iteration for Markov decision processes. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 160–180. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_8
5. Brenguier, R.: PRALINE: a tool for computing nash equilibria in concurrent games. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 890–895. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_63
6. Čermák, P., Lomuscio, A., Mogavero, F., Murano, A.: MCMAS-SLK: a model checker for the verification of strategy logic specifications. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 525–532. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_34
7. Chatterjee, K., Henzinger, T.A., Jobstmann, B., Radhakrishna, A.: GIST: a solver for probabilistic games. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 665–669. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_57. pub.ist.ac.at/gist/
8. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Form. Methods Syst. Des. **43**(1), 61–92 (2013)
9. Cheng, C.-H., Knoll, A., Luttenberger, M., Buckl, C.: GAVS+: an open platform for the research of algorithmic game solving. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 258–261. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_22. sourceforge.net/projects/gavsplus/
10. Cramton, P., Shoham, Y., Steinberg, R.: An overview of combinatorial auctions. SIGecom Exch. **7**, 3–14 (2007)

11. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal Stratego. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 206–211. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_16. people.cs.aau.dk/marius/stratego/
12. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24. github.com/Z3Prover/z3
13. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_49. yices.csl.sri.com
14. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. **735**, 111–131 (2018)
15. Hauser, O., Hilbe, C., Chatterjee, K., Nowak, M.: Social dilemmas among unequals. Nature **572**, 524–527 (2019)
16. Gutierrez, J., Najib, M., Perelli, G., Wooldridge, M.: EVE: a tool for temporal equilibrium analysis. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 551–557. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_35. github.com/eve-mas/eve-parity
17. Kelmendi, E., Krämer, J., Křetínský, J., Weininger, M.: Value iteration for simple stochastic games: stopping criterion and learning algorithm. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 623–642. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_36
18. Kwiatkowska, M., Norman, G., Parker, D.: Verification and control of turn-based probabilistic real-time games. In: Alvim, M.S., Chatzikokolakis, K., Olarte, C., Valencia, F. (eds.) The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy. LNCS, vol. 11760, pp. 379–396. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31175-9_22
19. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: Automated verification of concurrent stochastic games. In: McIver, A., Horvath, A. (eds.) QEST 2018. LNCS, vol. 11024, pp. 223–239. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99154-2_14
20. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: Equilibria-based probabilistic model checking for concurrent stochastic games. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 298–315. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30942-8_19
21. Kwiatkowska, M., Parker, D., Simaitis, A.: Strategic analysis of trust models for user-centric networks. In: Proceedings of the SR'13, EPTCS, vol. 112, pp. 53–60. Open Publishing Association (2013)
22. Kwiatkowska, M., Parker, D., Wiltsche, C.: PRISM-games 2.0: a tool for multi-objective strategy synthesis for stochastic games. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 560–566. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_35
23. Kwiatkowska, M., Parker, D., Wiltsche, C.: PRISM-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives. Softw. Tools Technol. Transf. **20**(2), 195–210 (2018)
24. LPSolve (version 5.5). lpsolve.sourceforge.net/5.5/
25. Markowitch, O., Roggeman, Y.: Probabilistic non-repudiation without trusted third party. In: Proceedings of the 2nd Workshop on Security in Communication Networks (1999)
26. McKelvey, R., McLennan, A., Turocy, T.: Gambit: Software tools for game theory, version 16.0.1 (2016). gambit-project.org

27. Nash, J.: Equilibrium points in n-person games. Proc. Natl. Acad. Sci **36**, 48–49 (1950)
28. Norman, G., Parker, D., Sproston, J.: Model checking for probabilistic timed automata. Form. Methods Syst. Des. **43**(2), 164–190 (2013). https://doi.org/10.1007/s10703-012-0177-x
29. Roughgarden, T., Tardos, E.: How bad is selfish routing? J. ACM **49**, 236–259 (2002)
30. Tennenholtz, M., Kurland, O.: Rethinking search engines and recommendation systems: a game theoretic perspective. Commun. ACM **62**, 66–75 (2019)
31. Toumi, A., Gutierrez, J., Wooldridge, M.: A tool for the automated verification of nash equilibria in concurrent games. In: Leucker, M., Rueda, C., Valencia, F.D. (eds.) ICTAC 2015. LNCS, vol. 9399, pp. 583–594. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25150-9_34
32. Wiltsche, C.: Assume-guarantee strategy synthesis for stochastic games. Ph.D. thesis, University of Oxford (2015)
33. Supporting materials and artifact. prismmodelchecker.org/files/cav20pg3/
34. PRISM-games website. prismmodelchecker.org/games/
35. PRISM-games case studies. prismmodelchecker.org/games/casestudies.php