Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

PRISM-games: Verification and Strategy Synthesis for Stochastic Multi-player Games with Multiple Objectives

Marta Kwiatkowska[1], **David Parker**[2], **Clemens Wiltsche**[1]

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

Received: date / Revised version: date

Abstract. PRISM-games is a tool for modelling, verification and strategy synthesis for stochastic multi-player games. These allow models to incorporate both probability, to represent uncertainty, unreliability or randomisation, and game-theoretic aspects, for systems where different entities have opposing objectives. Applications include autonomous transport, security protocols, energy management systems and many more. We provide a detailed overview of the PRISM-games tool, including its modelling and property specification formalisms, and its underlying architecture and implementation. In particular, we discuss some of its key features, which include multi-objective and compositional approaches to verification and strategy synthesis. We also discuss the scalability and efficiency of the tool and give an overview of some of the case studies to which it has been applied.

1 Introduction

Automatic verification and strategy synthesis are techniques for analysing probabilistic systems. They can be used to produce formal guarantees with respect to quantitative properties such as safety, reliability and efficiency. For example, they can be employed to synthesise controllers in applications such as autonomous vehicles, network protocols and robotic systems. These often operate in uncertain and adverse environments, models of which require both stochasticity, e.g., to represent noise, failures or delays, and game-theoretic aspects, to model non-cooperative agents or uncontrollable events.

PRISM-games is a tool for verification and strategy synthesis for turn-based *stochastic multi-player games*, a model in which each state is controlled by one of a set of players. That player resolves nondeterminism in its states by selecting an action to perform. The resulting behaviour, i.e., to which state the model then evolves, is probabilistic. This allows the model to capture both game-theoretic aspects and stochasticity.

The crucial ingredient for reasoning about stochastic multi-player games is *strategies*, which represent the choices made by a given player, based on the execution of the model so far. For a stochastic game comprising just one player (in other words, a Markov decision process), we may choose to consider the behaviour of the player to be adversarial (for example, representing the malicious environment of a security protocol). We can then *verify* that the model exhibits certain formally specified properties, regardless of the behaviour of the adversary.

Alternatively, we could assume that we are able to control the choices of the single player in this model (imagine, for example, it represents the navigation control system in an autonomous vehicle). In this setting, we can instead use *strategy synthesis* to generate a strategy (a controller) under which the behaviour of the game satisfies a formally specified property.

The general case, in which there are multiple players, allows us to model situations where there are entities with opposing objectives, for example a controller *and* a malicious environment. PRISM-games provides strategy synthesis techniques that can generate a strategy for one player of a stochastic game such that it is guaranteed to satisfy a property, regardless of the strategies employed by the other players. Returning to the autonomous vehicle above, we could generate a strategy for the vehicle controller which guarantees that the probability of successfully completing a journey is above a specified threshold, regardless of the behaviour of other, uncontrollable aspects of the system such as other road users.

This paper provides an overview of PRISM-games and the strategy synthesis techniques that it provides. These fall into two categories. The first, *single-objective* case is used to express zero-sum properties in which two opposing sets of players aim to minimise and maximise

a single objective: either the probability of an event or the expected reward accumulated before it occurs. The second, *multi-objective* case enables the exploration of trade-offs, such as between performance and resource requirements. The tool also performs computation and visualisation of the *Pareto sets* representing the optimal achievable trade-offs.

We also discuss the support in PRISM-games for *compositional* system development. This is done through *assume-guarantee* strategy synthesis, based on contracts over component interfaces that ensure cooperation between the components to achieve a common goal. For example, if one component satisfies the goal B under an assumption A on its environment (i.e., $A \rightarrow B$), while the other component ensures that the assumption A is satisfied, we can compose strategies for the components into a strategy for the full system achieving B. Multi-objective strategy synthesis, e.g., for an implication $A \rightarrow B$, can be conveniently employed to realise such assume-guarantee contracts. Again, Pareto set computation can be performed to visualise the relationship between properties and across interfaces.

The underlying verification and strategy synthesis techniques developed for PRISM-games have been published elsewhere, in [12,5,14,16,44,7]. Existing short tool papers focusing on the functionality added in versions 1.0 and 2.0 of PRISM-games were presented in [13] and [34], respectively. This paper provides a comprehensive overview of the full tool, including detailed examples of the modelling and property specification and summaries of the key theory and algorithms. We also discuss implementation details, the scalability of the tool and the application domains to which it has been applied.

Structure of the paper. Section 2 provides basic details of the underlying model of stochastic multi-player games, and explains how these can be described using the PRISM-games modelling language. Section 3 covers the property specification language, giving the formal syntax, semantics and examples of the various classes of quantitative properties that are supported. Section 4 gives an overview of the underlying algorithms used to perform verification and strategy synthesis and Section 5 describes the architecture of the tool and some lower-level aspects of its implementation. Section 6 presents some experimental results and discusses the scalability and efficiency of PRISM-games. We conclude, in Sections 7, 8 and 9, with a discussion of case studies to which the tool has been applied, a survey of related tools and some areas of current and future work.

2 Models and Modelling

We begin by explaining the underlying models used by PRISM-games and the means by which they are specified to the tool. We will use $Dist(S)$ to denote the set of discrete probability distributions over a set S.

2.1 Stochastic multi-player games

The primary probabilistic model supported by PRISM-games is *stochastic multi-player games* (SMGs). These model systems whose evolution is determined by the decisions of multiple *players* plus the presence of probabilistic behaviour. We restrict our attention here to *turn-based* (as opposed to *concurrent*) stochastic games, in which a single player controls each state of the model.

Definition 1 (SMG). A *stochastic multi-player game* (SMG) is a tuple $\mathcal{G} = (\Pi, S, (S_i)_{i\in\Pi}, \overline{s}, A, \delta, L)$, where:

- Π is a finite set of *players*,
- S is a finite set of *states*,
- $(S_i)_{i\in\Pi}$ is a partition of S,
- $\overline{s} \in S$ is an initial state,
- A is a finite set of *actions*,
- $\delta : S \times A \rightarrow Dist(S)$ is a (partial) probabilistic transition function,
- $L : S \rightarrow 2^{AP}$ is a labelling function mapping states to sets of atomic propositions from a set AP.

The state of an SMG $\mathcal{G}$ is initially $\overline{s}$ and it then evolves as follows. In any state s, there is a nondeterministic choice between the set of enabled actions $A(s) \subseteq A$, where $A(s) \stackrel{\text{def}}{=} \{a \in A \mid \delta(s,a) \text{ is defined}\}$. We assume that $A(s)$ is non-empty for all states s, i.e., that there are no deadlock states in the model (this is checked and enforced by the tool). The choice of an action from $A(s)$ is resolved by the player that *controls* the state s, i.e., the unique player $i \in \Pi$ for which $s \in S_i$. Once this player selects an action $a \in A$, a transition to a successor state s' occurs randomly, according to the probability distribution $\delta(s,a)$, i.e., the probability that a transition to s' occurs from the current state s is $\delta(s,a)(s')$.

A *path* through $\mathcal{G}$, representing one possible execution of the system that it models, is a (finite or infinite) sequence of states and actions $\pi = s_0 a_0 s_1 a_1 s_2 \ldots$, where $s_i \in S$, $a_i \in A(s_i)$ and $\delta(s_i, a_i)(s_{i+1}) > 0$ for all $i \in \mathbb{N}$. We write $FPath_{\mathcal{G},s}$ and $IPath_{\mathcal{G},s}$, respectively, for the set of all finite and infinite paths of $\mathcal{G}$ starting in state s, and denote by $FPath_{\mathcal{G}}$ and $IPath_{\mathcal{G}}$ the sets of *all* such paths.

To reason about the various possible ways in which the game can behave, we use the notion of *strategies*, which define the choices of actions made by players in each state, based on the history of the game's execution so far. Formally, these are defined as follows.

Definition 2 (Strategy). A *strategy* for player $i \in \Pi$ in SMG $\mathcal{G}$ is a function $\sigma_i : (SA)^* S_i \rightarrow Dist(A)$ which, for each path $\lambda{\cdot}s \in FPath_{\mathcal{G}}$ where $s \in S_i$, selects a probability distribution $\sigma_i(\lambda{\cdot}s)$ over $A(s)$. The set of all strategies for player i is denoted Σ_i.

Various important classes of strategies can be identified, by classifying the extent to which they use the

history of the game's execution so far and whether or not they use randomisation. For the latter, a strategy is called *deterministic* (or pure) if the distribution used to select actions is always a point distribution (i.e., selects a single action with probability 1), and *randomised* otherwise. Regarding the use of history (i.e., *memory*), we identify the following important classes.

Definition 3 (Memoryless strategy). A strategy σ_i is called *memoryless* if it only considers the current state when resolving nondeterminism, i.e., if $\sigma_i(\lambda{\cdot}s) = \sigma_i(\lambda'{\cdot}s)$ for all paths $\lambda{\cdot}s, \lambda'{\cdot}s \in FPath_{\mathcal{G}}$.

A *finite-memory* strategy has a *mode*, which is updated each time a transition is taken in the game and then used to select an action in each state.

Definition 4 (Finite-memory strategy). A *finite-memory* strategy is defined by a tuple $(Q, q_0, \sigma_i^u, \sigma_i^s)$ comprising:

- a finite set of *modes* (or *memory elements*) Q;
- an initial mode $q_0 \in Q$;
- a mode update function $\sigma_i^u : Q{\times}S_i \to Dist(Q)$;
- an action selection function $\sigma_i^s : Q{\times}S_i \to Dist(A)$

The mode q of the strategy initially takes the value q_0 and is updated according to the distribution $\sigma_i^u(q, s)$ for each observed state s of the game $\mathcal{G}$. At each step of the game's execution, if the mode is q and the current state of $\mathcal{G}$ is s, then the strategy chooses an action according to the distribution $\sigma_i^s(q, s)$.

In addition to the distinction mentioned above between randomised and deterministic strategies, finite-memory strategies are classified as *deterministic memory update* strategies if the mode update function σ_i^u always gives a point distribution over modes, and *stochastic memory update* strategies otherwise.

Given multiple strategies $\sigma_i \in \Sigma_i$ for several players $i \in \Pi$, we can combine them into a single strategy that resolves choices in all the states controlled by those players. For example, for strategies σ_1 and σ_2 for players 1 and 2, we write $\sigma = (\sigma_1, \sigma_2)$ to denote the combined strategy σ. For a *coalition* of players $C \subseteq \Pi$, we use Σ_C to denote the set of all (combined) strategies for players in C. If a strategy σ comprises strategies for all players of the game (sometimes called a *strategy profile*), we can construct a probability space $Pr_{\mathcal{G}}^{\sigma}$ over the infinite paths of $\mathcal{G}$. This measure also allows us to define the *expectation* $\mathbb{E}_{\mathcal{G}}^{\sigma}[\rho]$ of a measurable function ρ over infinite paths. We use $Pr_{\mathcal{G}}^{\sigma}$ and $\mathbb{E}_{\mathcal{G}}^{\sigma}[\rho]$ to formally define a variety of quantitative properties of a game's behaviour under a particular strategy, notably the probability of some measurable event, or the expected value of a reward/cost measure, respectively.

As shown in Definition 1, states of SMGs are labelled with atomic propositions, from some set AP. These are used to identify particular states of interest when writing temporal logic formulas to describe an event (see Section 3). We also augment SMGs with *reward structures* of the form $r\colon S \to \mathbb{R}_{\geqslant 0}$, which assign non-negative, real-valued rewards to the states of a game. These can have many different interpretations and are, in fact, often used to capture *costs* rather than rewards (for example, energy usage or elapsed time).

2.2 Subclasses of SMGs

Next, we identify some useful subclasses of stochastic multi-player games, which may be simpler to analyse, or be amenable to verification against a wider range of properties than the general model.

Stochastic two-player games. Firstly, we observe that, when the cardinality of the player set Π of an SMG is 2, the SMG is a (turn-based) *stochastic two-player game* [37], a widely studied class of models, sometimes also known as $2\frac{1}{2}$-player games.

In practice, these suffice for modelling many real-life scenarios in which there is a natural separation into two competing entities (for example, defender vs. attacker in the context of a security protocol, or controller vs. environment in the context of a control problem). In fact, for the properties currently supported by PRISM-games, the verification of stochastic *multi-player* games actually reduces to the problem of solving one or more stochastic two-player games.

Markov decision processes. It is also worth noting that, when an SMG contains only one player (or, when all but one player has a singleton set of choices in all of their states), the SMG is in fact a *Markov decision process* (MDP), sometime also called a $1\frac{1}{2}$-player game. The basic problem of strategy synthesis for MDPs has a lower time complexity than for SMGs (polynomial, rather than NP $\cap$ coNP) and support for this model is already implemented in the regular version of PRISM.

Stopping games. When we discuss multi-objective techniques for SMGs later in the paper, we will refer to two subclasses of models for which additional property classes are available. The first are *stopping games*. We call states in an SMG from which no other state can be reached under any strategy *terminal states*. A stopping game is an SMG in which terminal states are reached with probability 1 under any strategy.

Controllable multichain games. The second subclass of SMGs relevant for multi-objective properties are *controllable multichain* games [7]. Intuitively, an SMG is controllable multichain if one player (or set of players) has the ability to control which subset of states the

game eventually reaches and remains within. The formal definition relies on a generalisation of the notion of *end components* for MDPs [20] to *irreducible components* [7], which are strongly connected fragments of the SMG that, once reached, will never be left. Controllable multichain means that one player can, for each irreducible component H, and starting in any state s of the SMG, ensure that H is reached from s with probability 1. We refer the reader to [7] for a full definition.

2.3 Modelling SMGs in PRISM-games

PRISM-games extends the existing PRISM modelling language to provide a formalism for specifying SMGs to be analysed by the tool. In this section, we explain the language and illustrate it using an example.

PRISM uses a textual modelling language, originally inspired by the Reactive Modules formalism of [1], and based on guarded command notation. A model comprises one or more *modules* whose state is determined by a set of *variables* and whose behaviour is defined by a set of *guarded commands* of the form:

```
[act] guard -> p1 : update1 + ... + pn : updaten;
```

This comprises an (optional) action label `act`, a *guard* (a predicate over the variables of all modules in the model), and a list of *updates*, with associated probabilities. Each update $\texttt{update}_\texttt{i}$ is a list of assignments $(\texttt{v}_\texttt{j}\texttt{'=expr}_\texttt{j})$ which, if executed, would evaluate the expression $\texttt{expr}_\texttt{j}$ and assign the value to variable $\texttt{v}_\texttt{j}$. Each $\texttt{p}_\texttt{i}$ is an expression over variables of the model which, when evaluated, gives the probability for each update. Intuitively, when a module has a command whose guard is satisfied in the current state, that module can update its variables probabilistically, according to the updates.

Action labels serve several purposes. First, they are used to annotate the transitions in the SMG, i.e., they represent the actions in the set $A(s)$ for each state s that are chosen by the players. Secondly, they provide a means for multiple modules to execute their updates synchronously. More formally, this is done using a multi-way parallel composition of processes, which is based on the definition of parallel composition proposed by Segala for probabilistic automata [36]. Lastly, we also use action labels to specify which parts of the model belong to which player. This is explained below, using an example.

Example. Figure 1 shows a PRISM-games model of an SMG representing 3 robots navigating a space that is divided into a 3×2 grid. The grid, and the movements that can be made around it, are shown in Figure 2. A robot can select a direction of movement in each grid location (*north*, *east*, etc.) but, due to the possibility of obstacles, may with some probability end up remaining in the same location or moving to a different one.

```
smg

// Player definitions
player robot1 [north1], [east1], [south1], [west1], [stuck1] endplayer
player robot2 [north2], [east2], [south2], [west2], [stuck2] endplayer
player robot3 [north3], [east3], [south3], [west3], [stuck3] endplayer

// Robot 1, moving around a 3 by 2 grid
module robot1

    // Location of robot 1
    r1 : [0..5] init 0;

    // Movement of robot 1 around the grid
    [east1] r1=0 → 0.6 : (r1'=1) + 0.4 : (r1'=0);
    [south1] r1=0 → 0.8 : (r1'=3) + 0.1 : (r1'=1) + 0.1 : (r1'=4);
    [east1] r1=1 → 1 : (r1'=2);
    [south1] r1=1 → 0.5 : (r1'=4) + 0.5 : (r1'=2);
    [stuck1] r1=2 → 1 : (r1'=2);
    [stuck1] r1=3 → 1 : (r1'=3);
    [east1] r1=4 → 1 : (r1'=(r2=5|r3=5)?r1 : 5);
    [west1] r1=4 → 0.6 : (r1'=3) + 0.4 : (r1'=4);
    [north1] r1=5 → 0.9 : (r1'=2) + 0.1 : (r1'=5);
    [west1] r1=5 → 1 : (r1'=4);

endmodule

// Robots 2 and 3, defined by module renaming

module robot2 = robot1 [ r1=r2, r2=r1,
    north1=north2 , east1=east2 , south1=south2 ,
    west1=west2, stuck1=stuck2 ]
endmodule

module robot3 = robot1 [ r1=r3, r3=r1,
    north1=north3 , east1=east3 , south1=south3 ,
    west1=west3, stuck1=stuck3 ]
endmodule

// A scheduler, moving the three robots in turn, one by one
module sched

    // Which robot moves next
    t : [1..3] init 1;

    [north1] t=1 → (t'=2);
    [east1] t=1 → (t'=2);
    [south1] t=1 → (t'=2);
    [west1] t=1 → (t'=2);
    [stuck1] t=1 → (t'=2);

    [north2] t=2 → (t'=3);
    [east2] t=2 → (t'=3);
    [south2] t=2 → (t'=3);
    [west2] t=2 → (t'=3);
    [stuck2] t=2 → (t'=3);

    [north3] t=3 → (t'=1);
    [east3] t=3 → (t'=1);
    [south3] t=3 → (t'=1);
    [west3] t=3 → (t'=1);
    [stuck3] t=3 → (t'=1);

endmodule

// Labels: goal and hazard states for each robot
label "hazard1" = r1=1;
label "hazard2" = r2=1;
label "hazard3" = r3=1;
label "goal1A" = r1=5;
label "goal2A" = r2=5;
label "goal3A" = r3=5;
label "goal1B" = r1=2|r1=3;
label "goal2B" = r2=2|r2=3;
label "goal3B" = r3=2|r3=3;

// A reward structure to model the elapse of time
rewards "time"
    true : 1;
endrewards
```

Fig. 1: A PRISM-games description of a 3-player SMG modelling 3 robots navigating around a 3×2 grid.

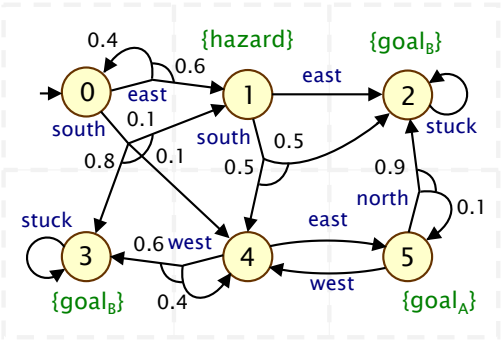

Fig. 2: Probabilistic movement of a robot around a 3×2 grid; used for the example model of Section 2.3 (see Figure 1).

The first module shown in Figure 1 represents robot 1, whose variable `r1` gives its current grid location (using the numbering from Figure 2). The guarded commands represent the possible actions that can be taken and the resulting probabilistic updates to variable `r1`. The second and third modules, for robots 2 and 3, are identical to the one for robot 1, except that the states of those robots are represented by different variables (`r2` and `r3`) and the action labels are different (e.g., `north2` and `north3`, rather than `north1`). So, these modules are defined using PRISM's *module renaming* feature.

The actions of the robots are mostly independent; however, no two can be in location 5 simultaneously. Hence, the update corresponding to a move into this location is written as `(r1'=(r2=5|r3=5)?r1:5)`, which means that `r1` is updated to 5 only if `r2` or `r3` is not already equal to 5; otherwise it remains unchanged.

We also comment on the use of parallel composition in the model. PRISM-games currently only supports turn-based SMGs. As a simple way of ensuring this is respected for the example model, we make the robots move in sequence, one by one. To enforce this, we make the action labels for modules `robot1`, `robot2` and `robot3` disjoint, and add a fourth module `sched`, with a variable `t` denoting the robot who is next to move and which synchronises with the appropriate module, depending on the current value of `t`. This is a relatively common approach in PRISM-games models where the concurrency between parallel components is controlled to allow it to be represented by a turn-based SMG.

The specification of the players in the SMG can be seen near the top of Figure 1. This defines the set of players in the SMG, and their names, and then assigns to each one a set of action names. This effectively associates each transition in the SMG with a particular player. Currently, since PRISM-games only supports turn-based games, it checks each state to ensure that all actions from that state are associated with the same player, and then assigns the state to that player. This approach (identifying player states via action names) is taken in order to provide compatibility with models such as concurrent SMGs, for which we plan to add support in the future.

Finally, we note that the last part of the PRISM-games model contains some straightforward definitions of *labels* and *rewards*, which are done in the same way as for standard PRISM models. Labels are used to identify states of interest, for the purposes of property specification, and are defined by a predicate over state variables; for this example, they are used to identify when each robot is in a location marked as a *goal* (either A or B) or a *hazard* in Figure 2. There is also a very simple reward structure *time*, modelling the elapse of time, defined by assigning a fixed reward of 1 to every state.

Compositional modelling. Version 2.0 of the PRISM-games tool added support for compositional strategy synthesis using assume-guarantee proof rules (see Section 4.3 for more details). In order to facilitate this, the modelling language also provides a specific compositional modelling approach designed for assume-guarantee strategy synthesis. This works for stochastic *2-player* games, specifically targeting controller synthesis problems modelled with games where player 1 represents a controller and player 2 its environment.

It allows a game to be defined as several *subsystems*, each of which comprises a set of modules, which are combined using the normal parallel composition of PRISM-games. Subsystems are combined using the game composition operator introduced in [6]; actions controlled by player 1 in subsystems are also controlled by player 1 in the composition, thus enabling composition of the synthesised player 1 strategies. This allows controller synthesis problems to be solved in a compositional fashion. For more details of this aspect of the language, we refer the reader to [44].

3 Property Specification

In order to formally specify the desired behaviour of an SMG, we use *properties*. In PRISM-games, properties are specified as temporal logic formulas. As a basis for this, we use the core part of the existing property specification logic of PRISM [32], which is itself based on the probabilistic temporal logic PCTL [28], extended with operators to reason about rewards [25].

PRISM-games currently supports two main classes of properties: (i) single-objective; and (ii) multi-objective. Although these fit within the same unified property specification language, there are different syntactic restrictions imposed in each case, so we present the two classes separately. In the explanations that follow, we will assume that we are specifying properties for a fixed SMG $\mathcal{G} = (\Pi, S, (S_i)_{i\in\Pi}, \overline{s}, A, \delta, L)$. When giving examples of

such properties, we will continue to use the SMG illustrated in the previous section, which has 3 players, $robot_1$, $robot_2$ and $robot_3$. Each label from Figure 1 (e.g., `"goal1A"`) corresponds to an atomic proposition (e.g., $goal_{1A}$), which can be used in temporal logic properties.

3.1 Single-objective properties

For *single-objective* properties of SMGs, we use a fragment of the property specification language called rPATL (probabilistic alternating-time temporal logic with rewards), originally proposed in [12]. This adopts the coalition operator $\langle\!\langle\cdot\rangle\!\rangle$ from the logic ATL [2], used for reasoning about strategies in non-probabilistic games, and adds the probabilistic operator P from PCTL and an extension of PRISM's reward operator R [25]. The formal definition is as follows.

Definition 5 (rPATL syntax). The syntax of the logic rPATL is given by the grammar

$$\phi ::= \texttt{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\!\langle C \rangle\!\rangle \theta$$
$$\theta ::= \texttt{P}_{\bowtie p}[\,\psi\,] \mid \texttt{R}^{r}_{\bowtie x}[\,\texttt{F}^{\star}\phi\,]$$
$$\psi ::= \texttt{X}\,\phi \mid \phi\ \texttt{U}^{\leqslant k}\ \phi \mid \phi\ \texttt{U}\ \phi$$

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

A property of an SMG $\mathcal{G}$ expressed in rPATL is a formula from the rule ϕ in the syntax above. The key operator is $\langle\!\langle C\rangle\!\rangle\theta$, where $C \subseteq \Pi$ is a *coalition* of players from $\mathcal{G}$ and θ is a quantitative objective that this set of players will aim to satisfy. An objective θ is a single instance of either the P or R operator: $\texttt{P}_{\bowtie p}[\,\cdot\,]$ means that the probability of some event being satisfied should meet the bound $\bowtie p$ and $\texttt{R}^{r}_{\bowtie x}[\,\cdot\,]$ means that expected value of a specified reward measure (using reward structure r) meets the bound $\bowtie x$. For example, the property:

$$\langle\!\langle\{robot_1\}\rangle\!\rangle \texttt{P}_{\geqslant 0.75}[\,\neg hazard_1\ \texttt{U}^{\leqslant 10}\ goal_{1A}\,]$$

asserts that there exists a strategy for $robot_1$ under which, regardless of the strategies chosen by the other players (in this case, $robot_2$ and $robot_3$), the probability of reaching a "goal A" state within 10 steps, without passing through a "hazard" state, is at least 0.75.

In general, the form of the probabilistic operator is $\texttt{P}_{\bowtie p}[\,\psi\,]$, where the event whose probability is being referred to is specified as a *path formula* ψ. As shown in the grammar, we allow three basic types of path formulas, taken from PCTL: $\texttt{X}\,\phi$ ("next": ϕ is true in the next state); $\phi_1\ \texttt{U}^{\leqslant k}\ \phi_2$ ("bounded until": ϕ_2 is true within k steps and ϕ_1 remains true in all states until that point); and $\phi_1\ \texttt{U}\ \phi_2$ ("until": ϕ_2 is eventually true and ϕ_1 remains true in all states until that point). As usual, we can derive other common temporal operators such as $\texttt{F}\,\phi \equiv \texttt{true}\ \texttt{U}\ \phi$ (eventually ϕ is true) and $\texttt{G}\,\phi \equiv \neg\texttt{F}\,\neg\phi$ (ϕ always remains true forever). An example is:

– $\langle\!\langle\{robot_1, robot_2\}\rangle\!\rangle \texttt{P}_{\geqslant 0.9}[\,\texttt{F}\ (goal_{1B}\wedge goal_{2B})\,]$ – robots 1 and 2 have strategies which ensure that, with probability at least 0.9, they are eventually both in a goal B state simultaneously, regardless of the actions taken by robot 3.

The reward operator $\texttt{R}^{r}_{\bowtie x}[\,\texttt{F}^{\star}\phi\,]$ allows us to reason about the expected amount of reward r accumulated until ϕ is true, i.e., the expected sum of rewards along a path until a state satisfying ϕ is first reached. The $\star$ parameter (which takes one of the values 0, ∞ or c) is used to indicate how to treat the situation where a ϕ-state is *not* reached. When this occurs, the cumulated reward for that path is, for the three cases $\star = 0$, ∞ or c, taken to be zero, infinite, or equal to the cumulated reward along the whole path, respectively.

These three different variants are provided because each has its own uses depending on the nature of the reward structure being used. Consider, for example, a situation where the goal of a player is to minimise the expected time for an algorithm to complete. In this case, it is natural to assume a value of infinity upon non-completion ($\star{=}\infty$). An alternative example, on the other hand, would be where we are trying to model and analyse a distributed algorithm by investigating a reward structure that incentivises certain kinds of behaviour, and aiming to maximise it over the lifetime of the algorithm's execution. Then, parameter $\star{=}0$ might be preferred to avoid favouring situations where the algorithm does not terminate. Lastly, when modelling for example, an algorithm's resource consumption, we might opt to use type $\star{=}\,c$, to represent resources used regardless of termination. An example of the first type would be:

– $\langle\!\langle\{robot_1\}\rangle\!\rangle \texttt{R}^{time}_{\leqslant 10}[\,\texttt{F}^{\infty} goal_{1A}\,]$ – it is possible for robot 1 to ensure that the expected time taken to reach goal A is at most 10.

Recall that the *time* reward structure from the example SMG simply assigns a fixed reward of 1 to every state.

Notice also that the syntax of rPATL allows us to construct Boolean combinations of properties ϕ and that, further, instances of the $\langle\!\langle C\rangle\!\rangle\theta$ operator can be nested to create more complex formulas. Examples of each include:

– $\langle\!\langle\{robot_1\}\rangle\!\rangle \texttt{P}_{\geqslant 1}[\,\texttt{F}\ goal_{1B}\,] \vee \langle\!\langle\{robot_2\}\rangle\!\rangle \texttt{P}_{\geqslant 1}[\,\texttt{F}\ goal_{2B}\,]$ – at least one of robot 1 and robot 2 has a strategy to ensure it reaches goal B with probability 1;

– $\langle\!\langle\{robot_1\}\rangle\!\rangle \texttt{P}_{<0.01}[\,\texttt{F}\ \neg\langle\!\langle\{robot_3\}\rangle\!\rangle \texttt{P}_{\geqslant 0.95}[\,\texttt{G}\ \neg hazard_3\,]\,]$ – it is possible for robot 1 to ensure that, with probability less than 0.01, a state is reached from which robot 3 is unable to guarantee that it avoids hazard states with probability at least 0.95.

Another useful class of properties, which are not explicitly included in the syntax above, are *numerical* queries. For a property comprising a single $\langle\!\langle C\rangle\!\rangle\theta$ operator, the

bound $\bowtie p$ or $\bowtie x$ in the `P` or `R` is replaced with either "min=?" or "max=?" and, rather than asking about the existence of a strategy satisfying the bound, yields the optimal (minimum or maximum) value obtainable by a strategy. Examples are:

- $\langle\!\langle \{robot_1\} \rangle\!\rangle \mathtt{P}_{\max=?}[\,\mathtt{G}\ \neg hazard_1\,]$ – what is the maximum probability with which robot 1 can guarantee that it avoids *hazard* states?
- $\langle\!\langle \{robot_2\} \rangle\!\rangle \mathtt{R}^{time}_{\min=?}[\,\mathtt{F}\ goal_{2A}\,]$ – what is the minimum expected time with which robot 2 can reach goal A?

3.2 Multi-objective properties

An important class of properties, added in version 2.0 of PRISM-games, are those which characterise the goals of a player (or players) using *multiple objectives*. We continue to use the coalition operator $\langle\!\langle C \rangle\!\rangle \theta$ discussed above, but now allow θ to be, for example, a conjunction of different objectives all of which need to be satisfied. More generally, we allow θ to be a Boolean combination of different objectives. For this class of properties, we focus purely on reward-based properties and consider a different range of reward objectives, which we explain in more detail below. The full syntax is as follows.

Definition 6 (Multi-objective syntax). The syntax for multi-objective properties is given by the grammar:

$$\phi ::= \mathtt{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\!\langle C \rangle\!\rangle \theta$$

$$\theta ::= \mathtt{R}^{r}_{\bowtie x}[\,\mathtt{C}\,] \mid \mathtt{R}^{r}_{\bowtie x}[\,\mathtt{S}\,] \mid \mathtt{R}^{r/r}_{\bowtie x}[\,\mathtt{S}\,] \mid \mathtt{P}_{\geqslant 1}[\,\psi\,] \mid \neg\theta \mid \theta \wedge \theta$$

$$\psi ::= \mathtt{R}^{r}_{\bowtie x}[\,\mathtt{S}\,] \mid \mathtt{R}^{r/r}_{\bowtie x}[\,\mathtt{S}\,]$$

where $a \in AP$, $C \subseteq \Pi$, $\bowtie \in \{<, \leqslant, \geqslant, >\}$, $x \in \mathbb{Q}_{\geqslant 0}$ and r is a reward structure.

As can be seen from the rule for θ, multi-objective properties can incorporate three different types of reward objective in the `R` operator and an alternative form of the `P` operator. We can reason about *total reward* (indefinitely cumulated rewards; $\mathtt{R}^{r}_{\bowtie x}[\,\mathtt{C}\,]$), *mean payoff* (long-run average reward; $\mathtt{R}^{r}_{\bowtie x}[\,\mathtt{S}\,]$), or the long-run *ratio* of two rewards ($\mathtt{R}^{r_1/r_2}_{\bowtie x}[\,\mathtt{S}\,]$). The `P` operator is used to specify almost-sure (i.e., probability 1) satisfaction objectives for mean-payoff and ratio rewards.

As for the single-objective case, each individual objective is associated with a threshold $\bowtie x$ to be met. Objectives are combined with the standard Boolean connectives (only $\neg$ and $\wedge$ are shown in the grammar, but $\vee$ and $\rightarrow$ can be derived in the usual fashion). The objectives within a single multi-objective query must be of the same type, and a query comprising almost-sure satisfaction objectives must take the form of a single conjunction. Multi-objective properties focus solely on expected reward properties, but note that expected total reward properties can be used to encode conventional probabilistic reachability.

Examples of multi-objective queries, again for the robot navigation example of Section 2, are as follows:

- $\langle\!\langle \{robot_1\} \rangle\!\rangle (\mathtt{R}^{energy_1}_{\leqslant e_1}[\,\mathtt{C}\,] \wedge \mathtt{R}^{tasks_1}_{\geqslant t_1}[\,\mathtt{C}\,])$ – robot 1 has a strategy which ensures that both the expected total energy consumed is at most e_1 and the expected total number of tasks completed is at least t_1, regardless of the behaviour of robots 2 and 3.
- $\langle\!\langle \{robot_1, robot_2\} \rangle\!\rangle (\mathtt{R}^{energy_1/time}_{\leqslant e_1}[\,\mathtt{S}\,] \vee \mathtt{R}^{energy_2/time}_{\leqslant e_2}[\,\mathtt{S}\,])$ – robots 1 and 2 have a combined strategy which ensures that either the expected long-run energy consumption rate of robot 1 is at most e_1 or it is at most e_2 for robot 2, whatever robot 3 does.

For these properties, we assume that additional reward structures $energy_i$ and $tasks_i$ (for $i = 1, 2, 3$) are added to the model, which track the energy consumed by robot i and the number of tasks that it has completed, respectively. For the purposes of these examples, we also ignore the restrictions imposed by PRISM-games as to whether the SMG is a stopping game or controllable multichain game (see Section 4.2 for details).

In the same way that single-objective properties can be written in a *numerical* form (min =? or max =?) to obtain optimal values directly, PRISM-games allows the extraction of Pareto sets to visualise the trade-offs between objectives in a multi-objective query (see Section 4 and Section 5 for details).

3.3 Property semantics

We conclude our discussion of the PRISM-games property specification language with a presentation of its formal semantics. A property ϕ of an SMG $\mathcal{G}$ (for either of the two grammars presented above) is interpreted for a state s of $\mathcal{G}$. We write $s \models \phi$ to denote that ϕ is satisfied (i.e., is true) in state s. The formal definition of the relation $\models$ is shown in Figure 3. As can be seen, the definition also uses several auxiliary satisfaction relations. In particular, $s, \sigma \models \theta$ means that, when the game is under the control of strategy σ, θ is satisfied in state s. This is used for the definition of the semantics of the coalition operator $\langle\!\langle C \rangle\!\rangle \theta$, which requires the existence of a (combined) strategy σ_1 for all players in C such that, for all possible strategies σ_2 of the other players, $s, (\sigma_1, \sigma_2) \models \theta$ holds, where (σ_1, σ_2) is the strategy combining the individual strategies for all players in the game.

For any state $s \in S$:		
$s \models \texttt{true}$		always
$s \models a$	$\iff$	$a \in L(s)$
$s \models \neg\phi$	$\iff$	$s \not\models \phi$
$s \models \phi_1 \wedge \phi_2$	$\iff$	$s \models \phi_1 \wedge s \models \phi_2$
$s \models \langle\!\langle C \rangle\!\rangle \theta$	$\iff$	$\exists \sigma_1 \in \Sigma_C$ such that $\forall \sigma_2 \in \Sigma_{\Pi \setminus C} \,.\, s, (\sigma_1, \sigma_2) \models \theta$
For any state $s \in S$ and strategy σ:		
$s, \sigma \models \texttt{P}_{\bowtie p}[\,\psi\,]$	$\iff$	$Pr^{\sigma}_{\mathcal{G},s}(\{\pi \in IPath_{\mathcal{G},s} \mid \pi \models \psi\}) \bowtie p$
$s, \sigma \models \texttt{R}^{r}_{\bowtie x}[\,\rho\,]$	$\iff$	$\mathbb{E}^{\sigma}_{\mathcal{G},s}(rew(r,\rho)) \bowtie x$
$s, \sigma \models \neg\theta$	$\iff$	$s, \sigma \not\models \theta$
$s, \sigma \models \theta_1 \wedge \theta_2$	$\iff$	$s, \sigma \models \theta_1 \wedge s, \sigma \models \theta_2$
For any path $\pi = s_0 a_0 s_1 a_1 s_2 \ldots \in IPath_{\mathcal{G}}$:		
$\pi \models \texttt{X}\,\phi$	$\iff$	$s_1 \models \phi$
$\pi \models \phi_1\ \texttt{U}^{\leqslant k}\ \phi_2$	$\iff$	$\exists i \leqslant k \,.\, \big(\, s_i \models \phi_2 \wedge (\,\forall j{<}i \,.\, s_j \models \phi_1\,)\,\big)$
$\pi \models \phi_1\ \texttt{U}\ \phi_2$	$\iff$	$\exists i \geqslant 0 \,.\, \big(\, s_i \models \phi_2 \wedge (\,\forall j{<}i \,.\, s_j \models \phi_1\,)\,\big)$
For any reward structure r and path $\pi = s_0 a_0 s_1 a_1 s_2 \ldots \in IPath_{\mathcal{G}}$:		
$rew(r, \texttt{F}^{\star}\phi)(\pi)$	$\stackrel{\text{def}}{=}$	$\begin{cases} 0 & \text{if } \forall j \in \mathbb{N} : s_j \not\models \phi \text{ and } \star = 0 \\ \infty & \text{if } \forall j \in \mathbb{N} : s_j \not\models \phi \text{ and } \star = \infty \\ \sum_{j \in \mathbb{N}} r(s_j) & \text{if } \forall j \in \mathbb{N} : s_j \not\models \phi \text{ and } \star = c \\ \sum_{j=0}^{k-1} r(s_j) & \text{otherwise, where } k = \min\{j \mid s_j \models \phi\} \end{cases}$
$rew(r, \texttt{C})(\pi)$	$\stackrel{\text{def}}{=}$	$\lim_{N \to \infty} r^N(\pi)$
$rew(r, \texttt{S})(\pi)$	$\stackrel{\text{def}}{=}$	$\liminf_{N \to \infty} \frac{1}{N+1} r^N(\pi)$
$rew(r_1/r_2, \texttt{S})(\pi)$	$\stackrel{\text{def}}{=}$	$\liminf_{N \to \infty} r_1^N(\pi)/(1 + r_2^N(\pi))$
$r^N(\pi)$	$\stackrel{\text{def}}{=}$	$\sum_{j=0}^{N} r(s_j)$ for $N \geqslant 0$

Fig. 3: Semantics of the PRISM-games property specification language for an SMG $\mathcal{G}$

4 Algorithms

We now describe the implementation of PRISM-games in more detail. In this section, we summarise the key algorithms required for model checking and give pointers to further information. In the next section we will describe the overall architecture of the tool and explain some of the low-level implementation details.

The key algorithmic task performed by PRISM-games is model checking of a property expressed in the logic of Section 3 against an SMG constructed from a description in the PRISM modelling language. Since the property specification language is based on a branching-time logic (it extends PCTL, which in turn extends CTL), the basic model checking algorithm performs a recursive traversal of the parse tree of the logical formula to be checked and, for each subformula, determines the set of states of the model that satisfy it.

The key operator for model checking is the coalition operator $\langle\!\langle C \rangle\!\rangle \theta$, where θ is either a single (probabilistic or reward) objective or a Boolean combination of such objectives. In either case, model checking essentially reduces to a strategy synthesis problem: determining whether the set of players C has a combined strategy that achieves the objective (or objectives), irrespective of the strategies of the other players.

This problem always reduces to a strategy synthesis problem on a (turn-based) *stochastic 2-player game*, in which the first player represents the players in C and the second player represents the others (a so-called *coalition game*). Thus the main component of the SMG model checking algorithm is strategy synthesis for either single- or multi-objective queries on a stochastic 2-player game.

4.1 Single-Objective Strategy Synthesis

For single-objective strategy synthesis, determining if a suitable strategy exists in this game reduces to computing *optimal* strategies for player 1, i.e., strategies that minimise or maximise the value of the objective. The problems of computing optimal probabilities of reaching a set of states or expected cumulative rewards in stochastic 2-player games are known to be in the complexity class NP ∩ coNP [18]. However, in practice, we can use *value iteration* algorithms, which iteratively approximate the optimal values for all states of the model, until some specified convergence criterion is met. This approach can be adapted to handle each of the variants of the $\texttt{F}^{\star}$ reward operators described in Section 3.1. This relies on the use of graph-based precomputation algorithms (e.g., to identify states from which the expected reward is infinite) and, for the case $\star = 0$, the use of two successive instances of value iteration which compute an

upper and lower bound on the required values, respectively. The process is described in more detail in [12].

For the cases of probabilistic reachability or until formulas ($\mathtt{P}_{\bowtie p}[\,\mathtt{F}\ \phi\,]$ or $\mathtt{P}_{\bowtie p}[\,\phi_1\ \mathtt{U}\ \phi_2\,]$) and cumulative expected reward ($\mathtt{R}^r_{\bowtie x}[\,\mathtt{F}^\star\phi\,]$), PRISM-games synthesises optimal strategies that are both memoryless and deterministic. For the bounded variants of reachability/until, a finite-memory strategy is generated. In each case, the optimal strategy can be inspected in the tool, or exported to a file for further analysis.

4.2 Multi-Objective Strategy Synthesis

For *multi-objective* properties (which, again, only need to be considered on a stochastic *2-player* game), PRISM-games implements the techniques presented in [5,14,16,7]. The syntax described in Section 3.2 allows a variety of different reward objectives to be used: expected total reward, expected long-run average reward (mean payoff), expected long-run reward ratio, and almost sure (probability 1) variants of the latter two.

For the purposes of model checking, PRISM-games imposes some restrictions on the classes of SMGs for which these can be checked. For expected total rewards, games must be *stopping* and, for expected long-run objectives, games must be *controllable multichain* (see Section 2.2 for details of these subclasses).

At the heart of the strategy synthesis algorithm is a unified fixpoint computation, used for all classes of properties after appropriate transformations have been applied. In particular, Boolean combinations of expectation objectives are converted to conjunctions by selecting appropriate *weights* for the individual objectives (see [14, Theorem 6] for details).

The core task is then to compute the set of *achievable* values for a conjunction of expected reward objectives. Assuming a conjunction of n reward objectives, with thresholds $x_1, \ldots, x_n$, a vector of n values is said to be achievable if a single (player 1) strategy of the game can simultaneously satisfy the threshold x_i for each of the n reward objectives.

Strategy synthesis is again performed using a value iteration style algorithm, but now computing, for each state s of the model, a *polytope* $P_S \subseteq \mathbb{R}^n$ of achievable values for that state, rather than a single value as in the single-objective case. In fact, the value iteration algorithm computes and stores the *Pareto set* of optimal values for the polytope, i.e., the frontier set containing achievable values that cannot be improved in any direction without degrading another. The full polytope is represented by its downward closure.

Since value iteration performs an approximate computation, up to a specified accuracy, the algorithm in fact constructs an ε-approximation of the Pareto set. Improvements in performance can be achieved by computing successive polytopes using in-place (i.e. Gauss-Seidel) updates, as well as rounding the corners of the polytopes at every iteration (where the latter comes at the cost of precision) [16].

From the results of the value iteration computation, a succinct *stochastic memory update* strategy (see Section 2.1 for the definition) can be constructed, which achieves the required values. Intuitively, the vertices of the polytope for each state form the memory elements of the constructed strategy and are used to track the values that need to be achieved for each objective. As for multi-objective strategy synthesis on the simpler model of MDPs [23], randomisation is needed in the strategy to capture trade-offs between objectives. See [16,44] for more information and some detailed examples.

In a similar fashion to the handling of numerical queries for single-objective properties, PRISM-games can also provide direct access to the Pareto set for a multi-objective property. In practice, these are visualised by selecting two-dimensional *slices* of the full set (see Section 5 and Figure 5 for an example).

4.3 Compositional Strategy Synthesis

Finally, we discuss the functionality in PRISM-games for *compositional* strategy synthesis techniques. Building upon assume-guarantee verification rules for probabilistic automata (i.e., games with only a single player), proposed in [33], support is provided for assume-guarantee strategy synthesis in stochastic 2-player games [6,7].

Given a system $\mathcal{G}$ composed of subsystems $\mathcal{G}_1$, $\mathcal{G}_2$, ..., a designer supplies respective *local property specifications* φ_1, φ_2, ... via the construct `comp(`φ_1`, `φ_2`, ...)`. By synthesising *local strategies* σ_i for $\mathcal{G}_i$ satisfying φ_i, a *global strategy* σ can be constructed for $\mathcal{G}$. Using *assume-guarantee rules*, one can then derive a *global property* φ for $\mathcal{G}$ that is satisfied by σ. The rules require *fairness* conditions, and we write $\mathcal{G}, \sigma \models^{\mathtt{u}} \varphi$ if the player 1 strategy σ satisfies φ against all unconditionally fair player 2 strategies. For example, the rule:

$$\frac{\mathcal{G}_1, \sigma_1 \models^{\mathtt{u}} \varphi^A \qquad \mathcal{G}_2, \sigma_2 \models^{\mathtt{u}} \varphi^A \rightarrow \varphi^G}{(\mathcal{G}_1 \parallel \mathcal{G}_2), (\sigma_1 \parallel \sigma_2) \models^{\mathtt{u}} \varphi^G} \qquad (\textsc{Asym})$$

states that player 1 wins with strategy $\sigma_1 \parallel \sigma_2$ for φ^G in the top-level system if σ_2 in $\mathcal{G}_2$ achieves φ^G under the contract $\varphi^A \rightarrow \varphi^G$, and σ_1 in $\mathcal{G}_1$ satisfies φ^A.

The tool compositionally computes a Pareto set for the property φ of the top-level system, which is an under-approximation of the Pareto set computed directly on the monolithic system. For a target in the compositional Pareto set, the targets for the local property specifications φ_i can be derived, so that the local strategies can be

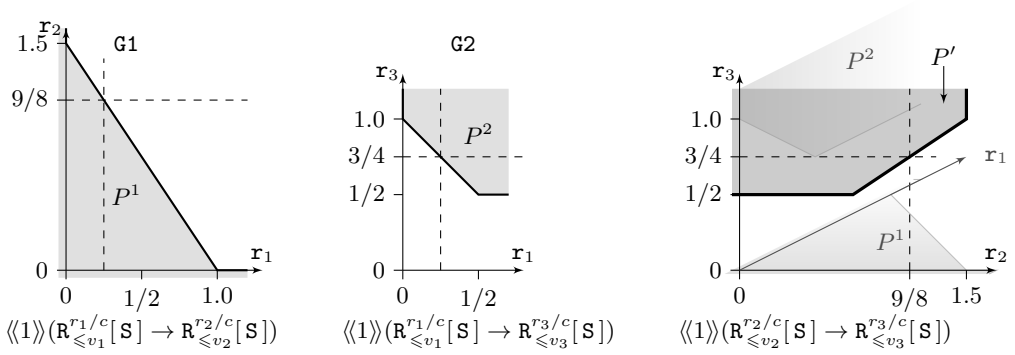

Fig. 4: Pareto sets generated by PRISM-games for compositional strategy synthesis (see Section 4.3).

synthesised. Figure 4 shows an example, with the property specifications given beneath each of the generated Pareto sets, using ratio reward objectives and 4 reward structures r_1, r_2, r_3 and c. The right-most image shows the compositional Pareto set P'. The global target is $(\mathtt{v}_2, \mathtt{v}_3) = (\frac{3}{4}, \frac{9}{8})$, and the local targets can be seen to be consistent with $\mathtt{v}_1 = \frac{1}{4}$.

5 Architecture and Implementation

PRISM-games is implemented primarily in Java. The tool is open source, currently released under the GPL, and is available from the PRISM-games website:

`http://www.prismmodelchecker.org/games/`

which also includes documentation, examples and related publications, as well as links to resources shared with PRISM, such as the support and discussion forums. The source code can also be accessed and browsed from the PRISM project's GitHub page:

`https://github.com/prismmodelchecker`

This site also hosts resources such as the benchmark and regression test suites, which are currently being extended with SMG model checking examples.

5.1 Architecture

PRISM-games is an extension of PRISM and so shares and extends the same basic architecture as well as various components from the original tool. This includes:

- *Parsers* for the modelling language and property specification languages, which are extended to support games and the new strategy synthesis properties.
- The *discrete-event simulator*, used for manual or automatic model exploration, which now supports SMGs, as well as the new notions of two-player parallel composition for assume-guarantee strategy synthesis.

Like its parent tool, the functionality within PRISM-games can be accessed in a variety of ways:

- The *command-line* interface, which is well suited for automated verification runs, long-running tasks or remote invocation.
- The *graphical user interface* (GUI), which provides editors for models and properties, graph plotting functionality and a simulator. Notable new features in the GUI include the ability to explore synthesised game strategies in the simulator, the possibility to apply a generated strategy to a game model and then perform further model checking, and the ability to visualise Pareto sets from multi-objective strategy synthesis. Screenshots illustrating the first and third of these can be found in Figure 5.
- The *application programming interface (API)*, which provides direct programmatic access (in Java) to the underlying model checking engine used by both the command-line and graphical user interfaces.

5.2 Data Structures

We now give some details of the key data structures needed to implement model checking of SMGs and the operations needed to manipulate them. Further details can be found in [44].

Games and strategies. The current implementation of model checking within PRISM-games uses explicit-state data structures, building upon PRISM's existing "explicit" engine which is written in Java.

Stochastic multi-player games are represented using sparse matrix data structures. For the purposes of performing strategy synthesis (which is primarily based on value iteration) the sparse matrices are equipped with various matrix-vector multiplication algorithms, which form the most expensive part of the solution process.

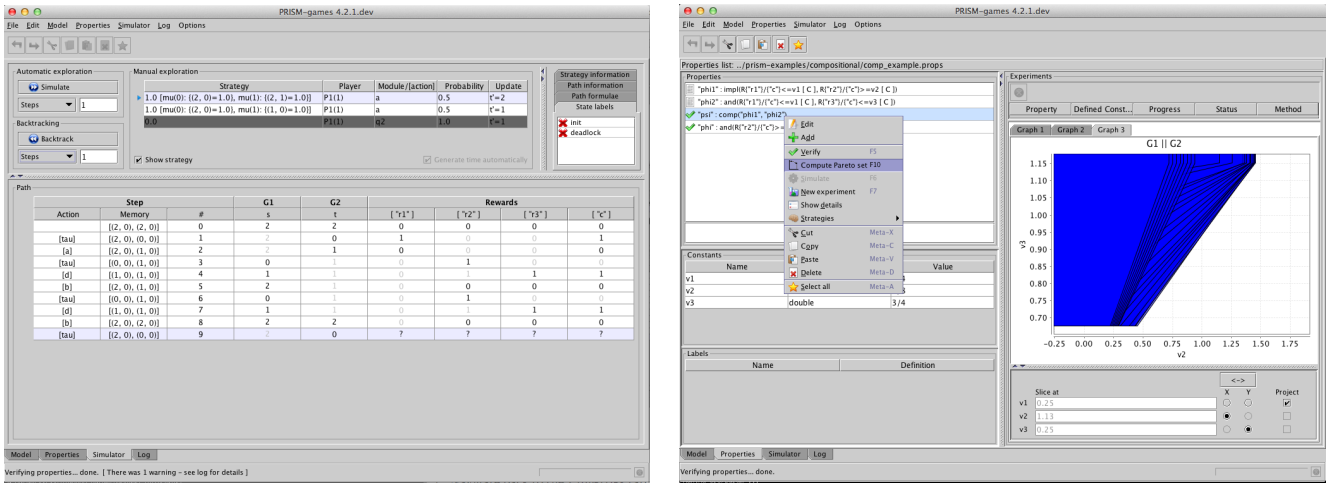

Fig. 5: Screenshots of the PRISM-games graphical user interface. Left: A synthesised strategy for an SMG being manually explored in the simulator. Right: Visualisation of a Pareto set from multi-objective strategy synthesis (here, a property with 3 objectives is displayed graphically by projecting the Pareto set onto the second and third objectives).

In order to support compositional modelling and verification, PRISM-games adds support to the data structures for combining SMGs using the notion of parallel composition from [6], described in Section 2.3. This includes various auxiliary steps, including an (optional) compatibility check and normal form transformation.

Strategies are also implemented in an explicit-state fashion. Memoryless strategies are represented as vectors and stochastic memory update strategies as maps encoding the next choice and memory update functions. For compositional verification within the assume-guarantee framework, the synthesised strategies are not explicitly composed, but the individual strategies are stored separately. When simulating a composed game under a composed strategy, the memory update is performed at each step only for the strategies corresponding to the involved components.

Polytopes. For single-objective SMG model checking, the core numerical computation techniques compute a single value (e.g., a probability or expected reward value) for each state of the SMG. These are mostly performed using fixpoint approximations over state-indexed vectors of floating point numbers. However, for multi-objective model checking, we need to compute sets of achievable objective values for each state, which can be done by associating each state with a convex, closed polytope.

Polytope representation and manipulation is done using the Parma Polyhedra Library (PPL) [4]. In particular, this provides support for performing the set-theoretic operations of convex union and intersection. The PPL library represents each polytope by both its vertices and the set of bounding hyperplanes, called the Motzkin double description method [4]. The vertex representation also allows for *rays*, which we use for downward closure operations: a ray is a vector $\mathbf{y}$ such that, for any point $\mathbf{x}$ in a polytope X, any point $\mathbf{x} + \alpha \cdot \mathbf{y}$ is also in X, for any $\alpha > 0$.

Both representations are equally expressive, but differ in how efficient operations are performed: intersection of polytopes is more efficient using hyperplanes (by taking the union of the bounding hyperplanes); convex union is more efficient using the vertices (by taking the union of the vertices). PPL automatically performs transformations between the representations, and can minimise the representation size. This representation of the polytopes is symbolic in the sense that we represent a polytope, an infinite set of points, by a finite set of vertices and hyperplanes.

Weighted Minkowski sum. One operation required by PRISM-games for manipulating polytopes, but not directly supported by the PPL library, is the *weighted Minkowski sum*. This is one of the core operations used during the iterative approximation of Pareto sets for multi-objective strategy synthesis, with the weights corresponding to the probabilities attached to outgoing transitions from each state of the SMG. Given sets $X, Y \subseteq \mathbb{R}^n$, their Minkowski sum is the set $X + Y \stackrel{\text{def}}{=} \{\mathbf{x}+\mathbf{y} \,|\, \mathbf{x} \in X \wedge \mathbf{y} \in Y\}$; given also a weight $\alpha \in [0,1]$ their weighted Minkowski sum is the set $\alpha X + (1-\alpha)Y \stackrel{\text{def}}{=} \{\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \,|\, \mathbf{x} \in X \wedge \mathbf{y} \in Y\}$.

We implement this operation using PPL's vertex representation, which we explain here for two polytopes P_1 and P_2. Let $P_i \stackrel{\text{def}}{=} \{\mathbf{x} \in \mathbb{R}^n \,|\, \exists \mathbf{y} \in \mathbb{R}^{m_i}_{\geqslant 0} \,.\, V_i\mathbf{y} = \mathbf{x} \wedge \mathbf{1}^T\mathbf{y} = 1\}$ for $i \in \{1,2\}$, where V_i is an $n \times m_i$ matrix defining the vertices of the polytope. The idea behind computing their weighted Minkowski sum is the following. First, lift the space to dimension $n+1$ and place P_1 and P_2 at

a distance of $-\frac{1}{1-\alpha}$ and $\frac{1}{\alpha}$ from the origin respectively. Their convex hull, shown with dashed lines in Figure 6, can then be computed as:

$$\{\mathbf{x} \in \mathbb{R}^{n+1} \mid \exists \mathbf{y} \in \mathbb{R}^{m_1+m_2+1} . \begin{bmatrix} V_1 & V_2 \\ \mathbf{1}^T/\alpha & -\mathbf{1}^T/(1-\alpha) \end{bmatrix} \mathbf{y} = \mathbf{x} \wedge \mathbf{1}^T\mathbf{y} = 1\}.$$

Constraining this polytope to $x_{n+1} = 0$, we get $\mathbf{1}^T\mathbf{y}_1 = \alpha$ and $\mathbf{1}^1\mathbf{y}_2 = 1 - \alpha$, and hence we define $\alpha\mathbf{z}_1 = \mathbf{y}_1$ and $(1-\alpha)\mathbf{z}_2 = \mathbf{y}_2$, so that $\mathbf{x} = \alpha V_1\mathbf{z}_1 + (1-\alpha)V_2\mathbf{z}_2$ and $\mathbf{1}^T\mathbf{z}_1 = \mathbf{1}^T\mathbf{z}_2 = 1$. This corresponds to computing the weighted Minkowski sum $\alpha \times P_1 + (1-\alpha) \times P_2$, as illustrated by the hatched polytope in Figure 6. Computing the convex hull and constraining to $x_{n+1} = 0$ are supported by PPL. This method extends to more than two polytopes in a similar fashion by introducing an extra dimension per polytope [30]. For more details and examples of the operation, see [44, Sec.s 6.2 and 7.2].

Note that the weighted Minkowski sum is the most computationally expensive operation that we have to implement, as the number of vertices of the polytope $\alpha P_1 + (1-\alpha)P_2$ is $\mathcal{O}(|P_1| \cdot |P_2|)$; see Theorem 4.1.1 of [43]. In contrast, the number of vertices of the polytopes $P_1 \cap P_2$ and $\mathsf{conv}(P_1 \cup P_2)$ is $\mathcal{O}(|P_1| + |P_2|)$ in both cases. The performance of our synthesis algorithms is therefore dependent on the outgoing branching degree of the states.

6 Experimental Results

Next, we present some experimental results for a selection of models analysed with PRISM-games, in order to give an illustration of the scalability of the tool. Although the property specification language presented in Section 3 allows multiple coalition operators $\langle\langle C \rangle\rangle\theta$ to be combined, either using Boolean combinations or nesting of subformulas, in practice, most queries are single instances of the operator so we restrict our attention to such properties. We discuss the cases of single-objective and multi-objective queries separately, since the techniques used to check them are quite different.

First, Table 1 shows statistics for a selection of models and the time taken to check various single-objective properties. The models are taken from three case studies: *team-form* (team formation protocols) [15], *mdsm* (microgrid demand-side management) [12,38] and *investor* (future markets investor) [35]. Each case study has a parameter that can be varied to yield increasingly large models (details can be found in the references cited above). The table shows the number of players and model size (number of states) for each one.

Again, we focus our attention on the kinds of objectives used most frequently in practice, which refer to either the probability of reaching a set of target states, or the expected reward accumulated before reaching it. For the latter, we show two of the three possible variants. Times are presented for performing strategy synthesis in each case using a 2.80GHz PC with 32GB RAM. Since this process reduces to the analysis of a stochastic 2-player game, the number of players in the original SMG has no impact on the solution time. Instead the model size is the key factor. We observe that, for a given example, the time for strategy synthesis increases roughly linearly with the increase in state space size. This is as expected since these properties are checked using an iterative numerical method, each iteration of which performs an operation for each state of the model.

PRISM-games is able to work with games of up to approximately 6 million states in these examples, which is comparable to the situation for similar models such as MDPs when using the "explicit" model checking engine on which PRISM-games is based. Future work will adapt PRISM's symbolic model checking engines, which can provide increased scalability in some cases, to SMGs.

Table 2 shows statistics for multi-objective strategy synthesis. The models are taken from four case studies: *uav* (human-in-the-loop UAV mission planning) [24], *driving* (autonomous urban driving) [16], *power* (aircraft power distribution) [5] and *temp* (temperature control) [44]; the first three are discussed in the next section.

The case studies do not all have parameters to scale the models, as in Table 1, but we show two variants of *driving* (using maps for two villages) and, for *power*, we vary the switch delays d. The last two case studies are used for compositional (assume-guarantee) strategy synthesis. We do not focus on that aspect here, but simply use it as a source of multi-objective queries. The table shows the number of components (subsystems) and sizes/times are given for each one, separated by slashes in the table entries. We omit the number of players since this is 2 in all cases.

The final column shows the time required for strategy synthesis, using the same hardware as above. In addition to considering different models, we also vary the accuracy with which Pareto curves are approximated. For multi-objective strategy synthesis, we observe that performance depends on multiple factors. Again, model size affects the time (see, e.g., the *power* example), but the number of objectives also has a significant impact (see, e.g., the *temp* example) and increasing the solution accuracy also results in longer runtimes.

7 Case Studies

PRISM-games has been under development since 2012 and has since then been successfully deployed to model and analyse systems across a wide range of application

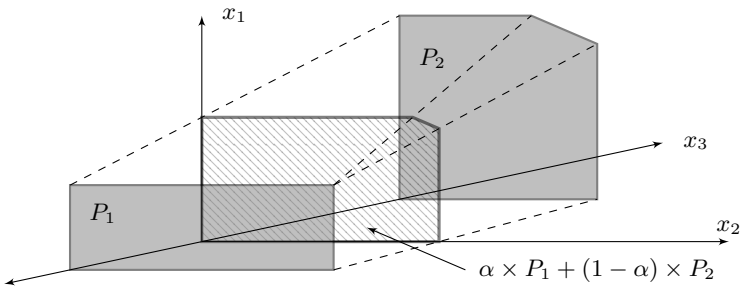

Fig. 6: Illustrating the computation of the weighted Minkowski sum. P_1 is shifted from the origin by $x_3 = -\frac{1}{1-\alpha}$, P_2 is shifted from the origin by $x_3 = \frac{1}{\alpha}$.

Case study [parameters]		SMG statistics Players	States	Strategy synthesis Property type	Time (s)
team-form [N]	3	3	12,475	$\langle\langle C \rangle\rangle \mathtt{P}_{\max=?}[\mathtt{F}\ \phi]$	0.2
	4	4	96,665		0.9
	5	5	907,993		11
mdsm [N]	5	5	743,904	$\langle\langle C \rangle\rangle \mathtt{R}_r^{\max=?}[\mathtt{F}^\infty \phi]$	62
	6	6	2,384,369		222
	7	7	6,241,312		1055
investor [$vmax$]	10	2	10,868	$\langle\langle C \rangle\rangle \mathtt{R}_r^{\max=?}[\mathtt{F}^c \phi]$	0.7
	100	2	750,893		122
	200	2	2,931,643		821

Table 1: A selection of experimental results for single-objective strategy synthesis queries.

Case study	SMG statistics Components	States	Strategy synthesis Num. obj.s	Objective types	Accuracy	Time (s)
uav	1	6,251	2	exp. total, Pareto	0.1	652
uav	1	6,251	2	exp. total, Pareto	0.01	871
driving [charlton]	1	501	3	exp. total	0.001	2603
driving [islip]	1	1,527	3	exp. total	0.001	1968
power [d=0]	2	7,296/7,296	3/3	almost-sure ratio	0.01	586/484
power [d=1]	2	24,744/24,744	3/3	almost-sure ratio	0.01	3325/2377
temp	3	1,478/1,740/1,478	3/2/3	exp. ratio	0.05	829/69/734
temp	3	1,478/1,740/1,478	3/2/3	exp. ratio	0.01	860/92/2480

Table 2: A selection of experimental results for multi-objective/compositional strategy synthesis queries.

domains. In this section, we survey some representative examples. Further details of several of the examples can be found in [38, 44, 42, 40] and an up-to-date list of case studies is maintained at the tool web site [45].

Microgrid demand-side management [38]. The example models a decentralised energy management protocol for smart grids that draw energy from a variety of sources. The system consists of a set of households, where each household follows a simple probabilistic protocol to execute a load if the current energy cost is below a pre-agreed limit, and otherwise it only executes the load with a pre-agreed probability. The energy cost to execute a load for a single time unit is the number of loads currently being executed in the grid. The analysis of the protocol with respect to the expected load per cost unit for a household, formulated as a single-objective total reward property, exposed a protocol weakness. The weakness was then corrected by disincentivising non-cooperative behaviour.

Human-in-the-loop UAV mission planning [24]. This case study concerns autonomous unmanned aerial vehicles (UAV) performing road network surveillance and reacting to inputs from a human operator. The UAV performs most of the piloting functions, such as selecting the waypoints and flying the route. The human operator performs tasks such as steering the onboard sensor to capture imagery of targets, but may also pick a road for the UAV at waypoints. The optimal UAV piloting strategy depends on mission objectives, e.g., safety, reachability, coverage, and operator characteristics, i.e., workload, proficiency, and fatigue. The main focus of the case study is on studying a multi-objective property to analyse the trade-off between the mission completion time and the number of visits to restricted operating zones, which have been investigated by computing Pareto sets.

Autonomous urban driving [16]. An SMG model of an autonomous car is developed, which considers the car driving through an urban environment and react-

ing to hazards such as pedestrians, obstacles, and traffic jams. The car not only decides on the reactions to hazards, which are adversarial, but also chooses the roads to take in order to reach a target location. The presence and probability of hazards is based on statistical information for the road. Through multi-objective strategy synthesis, strategies with optimal trade-off between the probability of reaching the target location, the probability of avoiding accidents and the overall quality of roads on the route are identified.

Aircraft power distribution [5]. An aircraft electrical power network is considered, where power is to be routed from generators to buses through controllable switches. The generators can exhibit failures and switches have delays. The system consists of several components, each containing buses and generators, and the components can deliver power to each other. The network is modelled as a composition of stochastic games, one for each component. These components are physically separated for reliability, and hence allow limited interaction and communication. Compositional strategy synthesis is applied to find strategies with good trade-off between uptime of buses and failure rate. By employing stochasticity, we can faithfully encode the reliability specifications in quantitative fashion, thus improving over previous results. The property is modelled as a conjunction of ratio reward properties.

Self-adaptive software architectures [26, 10]. Self-adaptive software automatically adapts its structure and behaviour according to changing requirements and quantitative goals. Several self-adaptive software architectures, such as adaptive industrial middleware used to monitor and manage sensor networks in renewable energy production plants, have been modelled as stochastic games and analysed. Both single- and multi-objective verification of multi-player stochastic games has been applied to evaluate their resilience properties and synthesise proactive adaptation policies.

DNS bandwidth amplification attack [22]. The Domain Name System (DNS) is an Internet-wide hierarchical naming system for assigning IP addresses to domain names, and any disruption of the service can lead to serious consequences. A notable threat to DNS, namely the bandwidth amplification attack, where an attacker attempts to flood a victim DNS server with malicious traffic, is modelled as a stochastic game. Verification and strategy synthesis is used to analyse and generate counter-measures to defend against the attack.

Attack-defence scenarios in RFID goods management system [3]. This case study considers complex attack-defence scenarios, such as an RFID goods management system, translating attack-defence trees to two-player stochastic games. Probabilistic verification is then employed to check security properties of the attack-defence scenarios and to synthesise strategies for attackers or defenders which guarantee or optimise some quantitative property. The properties considered include single-objective properties such as the probability of a successful attack or the incurred cost, as well as their multi-objective combinations.

8 Related tools

There are a variety of probabilistic model checking tools currently available. PRISM-games builds upon components of the PRISM [32] tool, as discussed in Section 5. Other general purpose probabilistic model checkers include MRMC [31], STORM [21], the Modest Toolset [29], iscasMc [27] and PAT [39]. However none of these provide support for stochastic games.

Other tools exist for the analysis of game models, but have a different focus to PRISM-games. For stochastic games, there is support for qualitative verification in GIST [11] and partial support in the general purpose game solver GAVS+ [17], but there are no tools for multi-objective or compositional analysis.

Analysis of Nash equilibria can be performed with EAGLE [41] or PRALINE [9], but only for non-stochastic games. Lastly, Uppaal Stratego [19] performs strategy synthesis against quantitative properties, but with a focus on real-time systems.

We also mention that multi-objective probabilistic verification, one of the key features of PRISM-games, is also available elsewhere for simpler models, notably Markov decision processes. This is supported by general purpose model checkers, such as PRISM [32] and STORM [21], and the more specialised tool MultiGain [8].

9 Conclusions and Future Work

PRISM-games is a tool for verification and strategy synthesis of stochastic multi-player games. It incorporates an array of techniques to support the generation of strategies specified by a wide range of formally specified quantitative properties, including single-objective and multi-objective variants. In this paper, we have provided an overview of the tool, from both a user perspective and in terms of the underlying implementation.

Various extensions are under development or planned for the future. Firstly, support for a wider range of temporal properties, specified in linear temporal logic (LTL), is in progress. Secondly, symbolic implementations of model checking are being added, to complement the current explicit-state version. Initially, this builds upon the

existing symbolic techniques implemented in PRISM for other probabilistic models using binary decision diagrams (BDDs) and multi-terminal BDDs.

Future work will investigate verification and strategy synthesis techniques for alternative game-theoretic solution concepts such as Nash equilibria, and wider classes of stochastic games, such as concurrent variants and games operating under partial observability.

Acknowledgements. This work has been supported by the ERC Advanced Grant VERIWARE and the EPSRC Mobile Autonomy Programme Grant. The authors also gratefully acknowledge helpful feedback from the anonymous reviewers.

References

1. R. Alur and T. Henzinger. Reactive modules. In *Proc. 11th Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, pages 207–218. IEEE Computer Society Press, July 1996.
2. R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.
3. Z. Aslanyan, F. Nielson, and D. Parker. Quantitative verification and synthesis of attack-defence scenarios. In *Proc. 29th IEEE Computer Security Foundations Symposium (CSF'16)*, pages 105–119. IEEE, 2016.
4. R. Bagnara, P. Hill, and E. Zaffanella. The Parma Polyhedra Library. *Science of Computer Programming*, 72(1–2):3–21, 2008.
5. N. Basset, M. Kwiatkowska, U. Topcu, and C. Wiltsche. Strategy synthesis for stochastic games with multiple long-run objectives. In *Proc. 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15)*, volume 9035 of *LNCS*, pages 256–271. Springer, 2015.
6. N. Basset, M. Kwiatkowska, and C. Wiltsche. Compositional controller synthesis for stochastic games. In *Proc. 25th International Conference on Concurrency Theory (CONCUR'14)*, volume 8704 of *LNCS*, pages 173–187. Springer, 2014.
7. N. Basset, M. Kwiatkowska, and C. Wiltsche. Compositional strategy synthesis for stochastic games with multiple objectives. *Information and Computation*, 2017. To appear.
8. T. Brázdil, K. Chatterjee, V. Forejt, and A. Kučera. MultiGain: A controller synthesis tool for MDPs with multiple mean-payoff objectives. In *Proc. 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15)*, volume 9035 of *LNCS*, pages 181–187. Springer, 2015.
9. R. Brenguier. PRALINE: A tool for computing Nash equilibria in concurrent games. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 890–895. Springer, 2013.
10. J. Cámara, G. A. Moreno, and D. Garlan. Stochastic game analysis and latency awareness for proactive self-adaptation. In *Proc. Software Engineering for Adaptive and Self-Managing Systems (SEAMS'14)*, pages 155–164, 2014.
11. K. Chatterjee, T. Henzinger, B. Jobstmann, and A. Radhakrishna. Gist: A solver for probabilistic games. In *Proc. 22nd International Conference on Computer Aided Verification (CAV'10)*, LNCS, pages 665–669. Springer, 2010.
12. T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.
13. T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. PRISM-games: A model checker for stochastic multi-player games. In *Proc. 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, volume 7795 of *LNCS*, pages 185–191. Springer, 2013.
14. T. Chen, V. Forejt, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. On stochastic games with multiple objectives. In *Proc. 38th International Symposium on Mathematical Foundations of Computer Science (MFCS'13)*, volume 8087 of *LNCS*, pages 266–277. Springer, 2013.
15. T. Chen, M. Kwiatkowska, D. Parker, and A. Simaitis. Verifying team formation protocols with probabilistic model checking. In *Proc. 12th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA XII 2011)*, volume 6814 of *LNCS*, pages 190–297. Springer, 2011.
16. T. Chen, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *Proc. 10th International Conference on Quantitative Evaluation of SysTems (QEST'13)*, volume 8054 of *LNCS*, pages 322–337. Springer, 2013.
17. C. Cheng, A. Knoll, M. Luttenberger, and C. Buckl. GAVS+: An open platform for the research of algorithmic game solving. In *Proc. 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)*, pages 258–261. Springer, 2011.
18. A. Condon. On algorithms for simple stochastic games. *Advances in computational complexity theory, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 13:51–73, 1993.
19. A. David, P. G. Jensen, K. G. Larsen, M. Mikucionis, and J. H. Taankvist. Uppaal stratego. In *Proc. 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15)*, volume 9035 of *LNCS*, pages 206–211. Springer, 2015.
20. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.
21. C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk. A Storm is coming: A modern probabilistic model checker. In *Proc. 29th International Conference on Computer Aided Verification (CAV'17)*, 2017.
22. T. Deshpande, P. Katsaros, S. Smolka, and S. Stoller. Stochastic game-based analysis of the DNS bandwidth amplification attack using probabilistic model checking. In *Proc. European Dependable Computing Conference (EDCC'14)*, pages 226–237, 2014.

23. K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. *Logical Methods in Computer Science*, 4(4):1–21, 2008.
24. L. Feng, C. Wiltsche, L. Humphrey, and U. Topcu. Controller synthesis for autonomous systems interacting with human operators. In *Proc. IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS'15)*, pages 70–79, 2015.
25. V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.
26. T. Glazier, J. Camara, B. Schmerl, and D. Garlan. Analyzing resilience properties of different topologies of collective adaptive systems. In *Proc. Self-Adaptive and Self-Organizing Systems Workshops (SASOW'15)*, pages 55–60, 2015.
27. E. M. Hahn, Y. Li, S. Schewe, A. Turrini, and L. Zhang. iscasMc: A web-based probabilistic model checker. In *Proc. 19th International Symposium on Formal Methods (FM'14)*, pages 312–317, 2014.
28. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
29. A. Hartmanns and H. Hermanns. The Modest toolset: An integrated environment for quantitative modelling and verification. In *Proc. 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, volume 8413 of *LNCS*, pages 593–598. Springer, 2014.
30. B. Huber, J. Rambau, and F. Santos. The Cayley trick, lifting subdivisions and the Bohne-Dress theorem on zonotopal tilings. *JEMS*, 2:179–198, 1999.
31. J.-P. Katoen, I. Zapreev, E. M. Hahn, H. Hermanns, and D. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104, 2011.
32. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
33. M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Compositional probabilistic verification through multi-objective model checking. *Information and Computation*, 232:38–65, 2013.
34. M. Kwiatkowska, D. Parker, and C. Wiltsche. PRISM-games 2.0: A tool for multi-objective strategy synthesis for stochastic games. In *Proc. 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'16)*, LNCS. Springer, 2016.
35. A. McIver and C. Morgan. Results on the quantitative mu-calculus qMu. *ACM Transactions on Computational Logic*, 8(1), 2007.
36. R. Segala. *Modelling and Verification of Randomized Distributed Real Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.
37. L. S. Shapley. Stochastic games. *PNAS*, 39(10):1095, 1953.
38. A. Simaitis. *Automatic Verification of Competitive Stochastic Systems*. PhD thesis, Department of Computer Science, University of Oxford, 2014.
39. J. Sun, Y. Liu, J. S. Dong, and J. Pang. PAT: Towards flexible verification under fairness. In *Proc. 21st International Conference on Computer Aided Verification (CAV'09)*, volume 5643 of *LNCS*, pages 709–714. Springer, 2009.
40. M. Svorenova and M. Kwiatkowska. Quantitative verification and strategy synthesis for stochastic games. *European Journal of Control*, 2016.
41. A. Toumi, J. Gutierrez, and M. Wooldridge. A tool for the automated verification of Nash equilibria in concurrent games. In *Proc. 12th International Colloquium on Theoretical Aspects of Computing (ICTAC'15)*, volume 9399 of *LNCS*, pages 583–594. Springer, 2015.
42. M. Ujma. *On Verification and Controller Synthesis for Probabilistic Systems at Runtime*. PhD thesis, University of Oxford, 2015.
43. C. Weibel. *Minkowski Sums of Polytopes: Combinatorics and Computation*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2007.
44. C. Wiltsche. *Assume-Guarantee Strategy Synthesis for Stochastic Games*. PhD thesis, University of Oxford, 2015.
45. PRISM-games website. www.prismmodelchecker.org/games/.