Symbolic Verification and Strategy Synthesis for Turn-based Stochastic Games

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

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

Abstract. Stochastic games are a convenient formalism for modelling systems that comprise rational agents competing or collaborating within uncertain environments. Probabilistic model checking techniques for this class of models allow us to formally specify quantitative specifications of either collective or individual behaviour and then automatically synthesise strategies for the agents under which these specifications are guaranteed to be satisfied. Although good progress has been made on algorithms and tool support, efficiency and scalability remain a challenge. In this paper, we investigate a symbolic implementation based on multi-terminal binary decision diagrams. We describe how to build and verify turn-based stochastic games against either zero-sum or Nash equilibrium based temporal logic specifications. We collate a set of benchmarks for this class of games, and evaluate the performance of our approach, showing that it is superior in a number of cases and that strategies synthesised in a symbolic fashion can be considerably more compact.

1 Introduction

Games have long been used as an underlying modelling formalism for the design and verification of computerised systems. For example, they naturally model the interaction between a system, whose behaviour can be controlled, and its environment whose actions cannot. Another example is the interplay between the defender and attacker in a computer security scenario.

In the context of *model checking*, where the required behaviour of a system is specified using temporal logic, we can use, for example, alternating-time temporal logic (ATL) [4] to formalise the capabilities of a player (or a coalition of players) acting in the context of another, adversarial player (or coalition) in a game model. Yet further expressive logics such as strategy logic [19] can also reason about the existence of, for instance, Nash equilibria.

Another important tool for modelling and verification is *stochasticity*. Probability is often essential to effectively quantify uncertain aspects of systems, from the presence of hardware failures to the unreliability of physical sensors. *Stochastic games* [49,24,28] are a well studied model for the dynamic execution of multiple players in a probabilistic setting. Results and algorithms for many verification problems on such models have also been presented, e.g., [17,18].

Building on these foundations, progress has since been made on the practical applicability of probabilistic model checking using stochastic games. This includes logics and algorithms for both *turn-based* stochastic games (TSGs) [21] and *concurrent* stochastic games (CSGs) [39]. The logic rPATL, a quantitative extension of ATL, allows specification of zero-sum properties for stochastic games, and extensions [39] also permit reasoning about the existence of Nash equilibria. A modelling formalism and tool support for TSGs and CSGs have been developed, in the form of PRISM-games [38], and this framework has been successfully applied to the analysis of, for example, human-robot collaborations [27,29], self-adaptive software systems [14] and computer security [6].

However, as usual for model checking approaches, efficiency and scalability are key challenges. So, in this paper, we consider *symbolic* implementations, in particular using binary decision diagrams (BDDs) and multi-terminal BDDs (MTBDDs), previously deployed for the compact representation and efficient manipulation of various models. Well known tools for verifying multi-agent systems such as MOCHA [3] and MCMAS [41] incorporate symbolic implementations of model checking, and probabilistic model checkers such as PRISM [34] and STORM [26] support symbolic techniques for simpler classes of stochastic models, such as Markov chains and Markov decision processes (MDPs).

As a first step in this direction, we consider a symbolic implementation of model checking and strategy synthesis for *turn-based* stochastic games. This also provides symbolic verification of (turn-based) probabilistic timed games, via the digital clocks translation [36]. We describe how to encode TSGs as MTBDDs and how to perform verification symbolically, in particular using value iteration. We also describe how to perform strategy synthesis, and how to extend this approach to compute Nash equilibria for TSGs.

In order to evaluate this, we collate a set of TSG model checking benchmarks of varying sizes, and add them to the PRISM benchmark suite [35]. We show that the symbolic approach offers significant gains in terms of the time required for model construction, for qualitative (graph-based) verification and, in some cases, for numerical solution of TSGs. We also show that optimal strategies can be represented more compactly symbolically rather than explicitly.

Related work. Various methods have been proposed for solving stochastic games [49,24,28] and for verifying them against logical specifications, e.g., [17,18]. GIST [16] implements qualitative verification against ω-regular specifications and PRISM-games [38] supports various quantitative properties. In [33], a wider range of methods for solving TSGs are implemented and explored, offering significant speed-ups. However, none of these provide symbolic implementations.

Multiple MTBDD-based implementations of probabilistic model checking for simpler (non-game) stochastic models have been developed. Originally, this focused on PCTL model checking for Markov chains or MDPs [9,8,48]. A so-called *hybrid* approach [48] improves performance through a combination of symbolic model storage and explicit-state algorithms, and is the default model checking engine in PRISM. Enhancements by others include automatic variable reordering and model checking of quantile-based properties [32]. MTBDDs and BDDs have

also been applied to the solution of energy games [5], which are *non-probabilistic* games with integer weights. Extensions of MTBDDs (XADDs) have been used for symbolic analysis of continuous-state MDPs [50].

Interpreting "symbolic" verification more widely, i.e., beyond BDD-based approaches, [42] considers symbolic methods for stochastic parity games, and [10] presents a probabilistic variant of the well known IC3 approach to model checking. Also relevant are methods to use, and learn, decision trees to succinctly represent strategies for probabilistic models [11].

2   Preliminaries

We begin with some background material, first on model checking of turn-based stochastic games, and then on (multi-terminal) binary decision diagrams.

Notation. We use $Dist(X)$ to denote the set of probability distributions over a set X, and we use $\mathbb{B} = \{0, 1\}$ for the set of Boolean values, with 0 denoting `false` and 1 denoting `true`.

2.1   Model Checking for Stochastic Games

Several variants of stochastic games exist. In this paper, we focus on (finite, multi-player) turn-based stochastic games.

Definition 1 (Turn-based stochastic game). *A* turn-based stochastic game *(TSG) is a tuple* $\mathcal{G} = (N, S, (S_i)_{i \in N}, \bar{s}, A, \delta, L)$ *where:*

- *N is a finite set of players;*
- *S is a finite set of states;*
- *$(S_i)_{i \in N}$ is a partition of S;*
- *$\bar{s} \in S$ is an initial state;*
- *A is a finite set of actions;*
- *$\delta : S \times A \to Dist(S)$ is a (partial) transition probability function;*
- *$L : S \to 2^{AP}$ is a labelling function.*

We now fix an n-player TSG $\mathcal{G}$ for the remainder of the section. The TSG $\mathcal{G}$ starts in its initial state $\bar{s} \in S$. In each state s, a player $i \in N$ selects an action from the set of *available* actions, which is denoted by $A(s) = \{a \in A \mid \delta(s, a) \text{ is defined}\}$. We assume that $A(s) \neq \varnothing$ for all states s. The choice of action to take in each state s is under the control of exactly one player, namely the player $i \in N$ for which $s \in S_i$. Once action $a \in A(s)$ is selected, the successor state is chosen according to the probability distribution $\delta(s, a)$, i.e., the game moves to state s' with probability $\delta(s, a)(s')$. We augment $\mathcal{G}$ with *reward structures*, which are tuples of the form $r = (r_A, r_S)$ where $r_A : S \times A \to \mathbb{R}$ and $r_S : S \to \mathbb{R}$ are action and state reward functions, respectively.

A *path* through $\mathcal{G}$ is a sequence $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ such that $s_i \in S$, $a_i \in A(s_i)$ and $\delta(s_i, a_i)(s_{i+1}) > 0$ for all $i \geqslant 0$. The sets of finite and infinite

paths (starting in state s) of $\mathcal{G}$ are given by $FPaths_\mathcal{G}$ and $IPaths_\mathcal{G}$ ($FPaths_{\mathcal{G},s}$ and $IPaths_{\mathcal{G},s}$).

Strategies of $\mathcal{G}$ are used to resolve the choices of the players. Formally, a strategy for player i is a function $\sigma_i\colon FPaths_\mathcal{G} \to Dist(A)$ such that, if $\sigma_i(\pi)(a_i) > 0$, then $a_i \in A(last(\pi))$ where $last(\pi)$ is the final state of path π. A *strategy profile* is a tuple $\sigma = (\sigma_1, \ldots, \sigma_n)$ of strategies for all players. The set of strategies for player i and set of profiles are denoted $\Sigma_\mathcal{G}^i$ and $\Sigma_\mathcal{G}$. Given a profile σ and state s, let $IPaths_{\mathcal{G},s}^\sigma$ denote the infinite paths with initial state s corresponding to σ. We can then define, using standard techniques [31], a probability measure $Prob_{\mathcal{G},s}^\sigma$ over $IPaths_{\mathcal{G},s}^\sigma$ and, for a random variable $X\colon IPaths_\mathcal{G} \to \mathbb{R}$, the expected value $\mathbb{E}_{\mathcal{G},s}^\sigma(X)$ of X from s under σ.

In $\mathcal{G}$, the utility or *objective* of player i is represented by a random variable $X_i\colon IPaths_\mathcal{G} \to \mathbb{R}$. Such variables can encode, for example, the probability of reaching a target or the expected cumulative reward before reaching a target.

We now introduce the notion of Nash equilibrium (NE) [46] for $\mathcal{G}$ given objectives $(X_i)_{i=1}^n$ for the players. We restrict our attention to subgame-perfect NE [47], which are NE in every state of $\mathcal{G}$. For profile $\sigma = (\sigma_1, \ldots, \sigma_n)$ and player i strategy σ_i', we define the sequence $\sigma_{-i} \stackrel{\text{def}}{=} (\sigma_1, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n)$ and profile $\sigma_{-i}[\sigma_i'] \stackrel{\text{def}}{=} (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$.

Definition 2 (Best response). *For objectives $(X_i)_{i=1}^n$, player i, strategy sequence σ_{-i} and state s, a* best response *for player i to σ_{-i} in state s is a strategy $\sigma_i^\star$ for player i such that $\mathbb{E}_{\mathcal{G},s}^{\sigma_{-i}[\sigma_i^\star]}(X_i) \geqslant \mathbb{E}_{\mathcal{G},s}^{\sigma_{-i}[\sigma_i]}(X_i)$ for all $\sigma_i \in \Sigma_\mathcal{G}^i$.*

Definition 3 (Nash equilibrium). *For objectives $(X_i)_{i=1}^n$, a strategy profile $\sigma^\star = (\sigma_1^\star, \ldots, \sigma_n^\star)$ of $\mathcal{G}$ is a subgame-perfect Nash equilibrium (NE) if $\sigma_i^\star$ is a best response to $\sigma_{-i}^\star$ for all $i \in N$ and $s \in S$. Furthermore, a NE $\sigma^\star$ of $\mathcal{G}$ is a* social welfare optimal NE *(SWNE) for objectives $X_1, \ldots, X_n$ if $\mathbb{E}_{\mathcal{G},s}^{\sigma^\star}(X_1) + \cdots + \mathbb{E}_{\mathcal{G},s}^{\sigma^\star}(X_n) \geqslant \mathbb{E}_{\mathcal{G},s}^\sigma(X_1) + \cdots + \mathbb{E}_{\mathcal{G},s}^\sigma(X_n)$ for all NE σ of $\mathcal{G}$.*

We can also define the dual concept of *social cost optimal NE* (SCNE) [39], for which the players of $\mathcal{G}$ try to minimise, rather than maximise, their expected utilities by considering equilibria for the objectives $-X_1, \ldots, -X_n$.

To formally specify properties of TSGs, we use the PRISM-games logic presented in [39], which extends the logic rPATL previously defined for zero-sum properties of TSGs [21]. The logic uses the *coalition* operator $\langle\!\langle C \rangle\!\rangle$ from alternating temporal logic (ATL) [4] to define *zero-sum* formulae and allows *nonzero-sum* properties, using (social welfare or social cost) NE.

Definition 4 (PRISM-games logic syntax). *The syntax of the PRISM-games logic is given by the grammar:*

$$\phi \;:=\; \texttt{true} \mid \texttt{a} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\!\langle C \rangle\!\rangle \texttt{P}_{\bowtie p}[\,\psi\,] \mid \langle\!\langle C \rangle\!\rangle \texttt{R}_{\bowtie q}^r[\,\rho\,] \mid \langle\!\langle C_1 \!:\! \cdots \!:\! C_m \rangle\!\rangle_{\mathrm{opt}\bowtie q}(\theta)$$

$$\psi \;:=\; \texttt{X}\,\phi \;\mid\; \phi\,\texttt{U}^{\leqslant k}\,\phi \;\mid\; \phi\,\texttt{U}\,\phi$$

$$\rho \;:=\; \texttt{I}^{=k} \;\mid\; \texttt{C}^{\leqslant k} \;\mid\; \texttt{F}\,\phi$$

$$\theta \;:=\; \texttt{P}[\,\psi\,] + \cdots + \texttt{P}[\,\psi\,] \;\mid\; \texttt{R}^r[\,\rho\,] + \cdots + \texttt{R}^r[\,\rho\,]$$

where C and $C_1, , \ldots, C_m$ are coalitions of players such that $C_i \cap C_j = \varnothing$ for all $1 \leqslant i \neq j \leqslant m$ and $\cup_{i=1}^{m} C_i = N$, $\bowtie \in \{<, \leqslant, \geqslant, >\}$, $p \in [0,1] \cap \mathbb{R}$, r is a reward structure, $q \in \mathbb{R}_{\geqslant 0}$, $\mathrm{opt} \in \{\min, \max\}$, $\mathtt{a}$ is an atomic proposition and $k \in \mathbb{N}$.

The syntax of the PRISM-games logic distinguishes between state (ϕ), path (ψ), reward (ρ) and nonzero-sum (θ) formulae. State formulae are evaluated over states of a TSG, while path, reward and nonzero-sum formulae are evaluated over paths. *Zero-sum* state formula have the following meaning:

- $\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\bowtie q}[\psi]$ is satisfied in a state if the coalition of players C can ensure that the probability of the path formula ψ being satisfied is $\bowtie q$, regardless of the actions of the other players;
- $\langle\!\langle C \rangle\!\rangle \mathtt{R}^r_{\bowtie x}[\rho]$ is satisfied in a state if the players in C can ensure that the expected value of the reward formula ρ for reward structure r is $\bowtie x$, regardless of the actions of the other players.

On the other hand, for a *nonzero-sum* state formula:

- $\langle\!\langle C_1 : \cdots : C_m \rangle\!\rangle_{\max \bowtie x}(\theta)$ is satisfied if there exists a subgame-perfect SWNE profile between coalitions $C_1, \ldots, C_m$ under which the *sum* of the objectives of $C_1, \ldots, C_m$ in θ is $\bowtie x$;
- $\langle\!\langle C_1 : \cdots : C_m \rangle\!\rangle_{\min \bowtie x}(\theta)$ is satisfied if there exists a subgame-perfect SCNE profile between coalitions $C_1, \ldots, C_m$ under which the *sum* of the objectives of $C_1, \ldots, C_m$ in θ is $\bowtie x$.

For all of the above formulae, we also allow *numerical* variants, which directly yield an optimal value, rather than checking whether a threshold can be met. For example, $\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\max=?}[\psi]$ gives the maximum probability with which the players in C can guarantee that that ψ is satisfied.

Both zero-sum and nonzero-sum formulae are composed of *path* (ψ) and *reward* (ρ) formulae, used in the probabilistic and reward objectives included within $\mathtt{P}$ and $\mathtt{R}$ operators, respectively. The path formulae include: *next* ($\mathtt{X}\,\phi$), *bounded until* ($\phi\,\mathtt{U}^{\leqslant k}\,\phi$) and *unbounded until* ($\phi\,\mathtt{U}\,\phi$). There are also the standard equivalences including: *probabilistic reachability* ($\mathtt{F}\,\phi \equiv \mathtt{true}\,\mathtt{U}\,\phi$) and *bounded probabilistic reachability* ($\mathtt{F}^{\leqslant k}\,\phi \equiv \mathtt{true}\,\mathtt{U}^{\leqslant k}\,\phi$).

The reward formulae include: *instantaneous (state) reward* at the kth step ($\mathtt{I}^{=k}$), *bounded cumulative reward* over k steps ($\mathtt{C}^{\leqslant k}$), and *reachability reward* until a formula ϕ is satisfied ($\mathtt{F}\,\phi$). In the case of reachability reward formulae, several variants have previously been introduced [21], differing in how they treat paths that do not reach a state satisfying ϕ. We restrict our attention to the most common one, the default in PRISM, which assigns the reward value infinity to paths that never reach a state satisfying ϕ.

We next define the semantics of the PRISM-games logic for TSGs. However, we first need to define the concept of a coalition game.

Definition 5 (Coalition game). *For TSG $\mathcal{G}$ and a partition of its players into m coalitions $\mathcal{C} = \{C_1, \ldots, C_m\}$, we define the coalition game $\mathcal{G}^\mathcal{C} = (\{1, \ldots, m\}, S, (S_i^\mathcal{C})_{i \in M}, \bar{s}, A, \delta, L)$ as an m-player TSG where $S_i^\mathcal{C} = \cup_{j \in C_i} S_j$.*

To simplify notation, for any coalition C of $\mathcal{G}$, we use the notation $\mathcal{G}^C$ to represent the 2-player coalition game $\mathcal{G}^{\mathcal{C}}$ where $\mathcal{C} = \{C, N \backslash C\}$.

Definition 6 (PRISM-games logic semantics). *For a TSG $\mathcal{G}$ and formula ϕ, we define the satisfaction relation $\models$ inductively over the structure of ϕ. The propositional logic fragment (true, a, $\neg$, $\wedge$) is defined in the usual way. For a PRISM-games logic formula and state $s \in S$ of TSG $\mathcal{G}$, we have:*

$$s \models \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\bowtie q}[\,\psi\,] \;\Leftrightarrow\; \exists \sigma_1 \in \Sigma^1. \forall \sigma_2 \in \Sigma^2. \mathbb{E}^{\sigma_1;\sigma_2}_{\mathcal{G}^C,s}(X^\psi) \bowtie q$$

$$s \models \langle\!\langle C \rangle\!\rangle \mathtt{R}^r_{\bowtie x}[\,\rho\,] \;\Leftrightarrow\; \exists \sigma_1 \in \Sigma^1. \forall \sigma_2 \in \Sigma^2. \mathbb{E}^{\sigma_1;\sigma_2}_{\mathcal{G}^C,s}(X^{r,\rho}) \bowtie x$$

$$s \models \langle\!\langle C_1 : \cdots : C_m \rangle\!\rangle_{\mathrm{opt}\bowtie q}(\theta) \;\Leftrightarrow\; \exists \sigma^\star \in \Sigma_{\mathcal{G}^{\mathcal{C}}}. \big(\textstyle\sum_{i=1}^m \mathbb{E}^{\sigma^\star}_{\mathcal{G}^{\mathcal{C}}}(X_i^\theta) \big) \bowtie q$$

and $\sigma^\star$ is an SWNE if $\mathrm{opt} = \max$, and an SCNE if $\mathrm{opt} = \min$, for the objectives $(X_i^\theta)_{i=1}^m$ in the coalition game $\mathcal{G}^{\mathcal{C}}$.

For an objective X^ψ, $X^{r,\rho}$ and path $\pi \in IPaths_{\mathcal{G}^C,s}$:

$$X^\psi(\pi) \;=\; 1 \text{ if } \pi \models \psi \text{ and } 0 \text{ otherwise}$$
$$X^{r,\rho}(\pi) \;=\; rew(r,\rho)(\pi).$$

The semantics for satisfaction of path formulae ($\pi \models \psi$) and the random variable $rew(r,\rho)(\pi)$ for a reward formula can be found in, e.g., [39].

As the zero-sum objectives appearing in the logic are either finite-horizon or infinite-horizon and correspond to either probabilistic until or expected reachability formulae, we have that TSGs are *determined* with respect to these objectives [43], which yields the following equivalences:

$$\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\max=?}[\,\psi\,] \equiv \langle\!\langle N \backslash C \rangle\!\rangle \mathtt{P}_{\min=?}[\,\psi\,]$$
$$\langle\!\langle C \rangle\!\rangle \mathtt{R}^r_{\max=?}[\,\rho\,] \equiv \langle\!\langle N \backslash C \rangle\!\rangle \mathtt{R}^r_{\min=?}[\,\rho\,].$$

Also, as for other probabilistic temporal logics, we can represent negated path formulae by inverting the probability threshold, e.g.:

$$\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\geqslant q}[\,\neg\psi\,] \equiv \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\leqslant 1-q}[\,\psi\,]$$

notably allowing the 'globally' operator $\mathtt{G}\,\phi \equiv \neg(\mathtt{F}\,\neg\phi)$ to be defined.

Since the logic is branching-time, the model checking algorithm for the logic works by recursively computing the set $Sat(\phi)$ of states satisfying formula ϕ over the structure of ϕ. The main step in the algorithm requires computation of values for zero-sum and nonzero-sum formulae. The standard approach is to use *value iteration* [15], which we discuss below.

Note that the PRISM logic used here, which includes non-zero sum formulae, was first considered for CSGs [39,37]. Our focus here is on TSGs, where the computation of values is simpler: because only one coalition has a choice in each state, value iteration need only take the minimum or maximum over actions, whereas for CSGs matrix games need to be solved in each state.

For the case of zero-sum formulae, efficiency and accuracy can be improved through the use of graph-based *precomputation algorithms* [4], which identify the

states that have values 0 and 1 in the case of probabilistic properties and value ∞ in the case of expected reward properties.

Value iteration. Below, we illustrate value iteration for the zero-sum formula $\phi = \langle\langle C \rangle\rangle \mathsf{P}_{\max=?}[\mathsf{F}\ \phi']$; the remaining cases have a similar structure. The value of ϕ in state s is given by the limit $val(s, \phi) = \lim_{k \to \infty} x_s^k$, where for any $k \in \mathbb{N}$:

$$
x_s^k = \begin{cases}
1 & \text{if } s \in Sat(\phi') \\
0 & \text{else if } k = 0 \\
\max_{a \in A(s)} \sum_{s' \in S} \delta(s, a)(s') \cdot x_{s'}^{k-1} & \text{else if } s \in \cup_{i \in C} S_i \\
\min_{a \in A(s)} \sum_{s' \in S} \delta(s, a)(s') \cdot x_{s'}^{k-1} & \text{otherwise}
\end{cases}
$$

In practice, a suitable convergence criterion needs to be chosen to terminate the computation. Here, we use the simple but common approach of checking the maximum relative difference between values for states in successive iterations, but more sophisticated approaches have been devised for TSGs [30].

2.2 Binary Decision Diagrams

A *binary decision diagram* (BDD) [12] is a rooted, directed acyclic graph used to provide a compact representation of a Boolean function over a particular set of Boolean variables. A BDD b over n Boolean variables $\underline{\mathsf{x}} = (\mathsf{x}_1, \ldots, \mathsf{x}_n)$ represents a function $f_{\mathsf{b}} : \mathbb{B}^n \to \mathbb{B}$. BDDs have two types of nodes: (i) *non-terminal* nodes, which are labelled with a variable x_i, and whose outgoing edges are labelled 1 ("then") and 0 ("else"); and (ii) *terminal* (leaf) nodes, labelled with 0 or 1. For a valuation $\underline{v} = (v_1, \ldots, v_n) \in \mathbb{B}^n$ of $\underline{\mathsf{x}}$, the value of $f_{\mathsf{b}}(\underline{v})$ can be found by traversing the BDD b from its root to a terminal node, taking at each non-terminal node the edge matching the value v_i for its variable x_i. The value of $f_{\mathsf{b}}(\underline{v})$ is taken as the value of the terminal node that is reached.

By requiring that variables are ordered, from the root node downwards, and by storing the graph in reduced form (merging isomorphic subgraphs, and removing redundant nodes), BDDs can represent structured Boolean functions very compactly and can be manipulated efficiently, i.e., with operations whose complexity is proportional to the number of nodes in the graph rather than the size of the function. This includes all standard Boolean operators, for example, $\text{OR}(\mathsf{b}_1, \mathsf{b}_2)$, which returns the BDD representing the function $f_{\mathsf{b}_1} \vee f_{\mathsf{b}_2}$. We also use $\text{AND}(\mathsf{b}_1, \mathsf{b}_2)$ and $\text{NOT}(\mathsf{b})$, defined analogously.

Multi-terminal BDDs (MTBDDs) [23], which are also sometimes known as algebraic decision diagrams (ADDs) [7], generalise BDDs by allowing terminal nodes to be labelled with values from an arbitrary set D. Hence, they represent functions of the form $f : \mathbb{B}^n \to D$. Typically, we are interested in real-valued functions and so an MTBDD m over n Boolean variables $\underline{\mathsf{x}} = (\mathsf{x}_1, \ldots, \mathsf{x}_n)$ represents a function $f_{\mathsf{m}} : \mathbb{B}^n \to \mathbb{R}$. Like for BDDs, a variety of useful operators for MTBDDs can be implemented. In particular, we use:

- APPLY($op, \mathsf{m}_1, \mathsf{m}_2$), where op is a binary operation over the reals: returns the MTBDD representing the function $f_{\mathsf{m}_1}\, op\, f_{\mathsf{m}_2}$.
- IFTHENELSE($\mathsf{b}, \mathsf{m}_1, \mathsf{m}_2$), where b is a BDD and $\mathsf{m}_1, \mathsf{m}_2$ are MTBDDs: returns the MTBDD for the function with value f_{m_1} if f_{b} is true and f_{m_2} otherwise.
- CONST(c), where $c \in \mathbb{R}$: returns the MTBDD representing the constant function with value c.
- ABSTRACT($op, \underline{\mathsf{y}}, \mathsf{m}$), where op is a commutative and associative binary operation over the reals (here, we often use min or max) and $\underline{\mathsf{y}} \subset \underline{\mathsf{x}}$ is a subset of the variables of m: returns an MTBDD over variables $\underline{\mathsf{x}}\backslash\underline{\mathsf{y}}$ representing the result of abstracting all the variables in $\underline{\mathsf{y}}$ from m by applying op over all possible values taken by the variables in $\underline{\mathsf{y}}$.

BDDs were popularised thanks to the success of *symbolic model checking* [13,45], which uses them to provide an efficient and scalable implementation of model checking, for example of the temporal logic CTL on labelled transition systems. Assume that we have an encoding $\mathsf{enc}_S : S \to \mathbb{B}^k$ of the state space S of a transition system into k Boolean variables. We can represent a subset $S' \subseteq S$ as a BDD, by using it to encode the characteristic function $\chi_{S'} : S \to \mathbb{B}$. A transition relation $\to\, \subseteq S \times S$ can be represented similarly as a BDD over 2 sets of k Boolean variables, i.e., by a BDD b where $f_{\mathsf{b}}(\mathsf{enc}_S(s), \mathsf{enc}_S(s')) = 1$ if and only if $(s, s') \in\, \to$. The key operations for model checking such as (pre or post) image computation can be performed efficiently on these BDD representations.

Symbolic implementations of probabilistic model checking [9,48] build on the fact that real-value vectors and matrices can be represented as MTBDDs in similar fashion. A key operation used in the numerical computation required for probabilistic model checking (i.e., for value iteration) is matrix-vector multiplication, which can be performed symbolically [23,22]:

- MVMULT(m, v), where m is an MTBDD over variables $\underline{\mathsf{x}}, \underline{\mathsf{y}}$ representing a matrix $\mathbf{M}$ and v is an MTBDD over variables $\underline{\mathsf{x}}$ representing a vector $\mathbf{v}$: returns the MTBDD over variables $\underline{\mathsf{x}}$ representing the vector $\mathbf{Mv}$.

3   Symbolic Model Checking for Stochastic Games

We now describe a symbolic implementation for the representation, construction and verification of TSGs.

3.1   Symbolic Representation and Construction of TSGs

We begin by discussing how to represent TSGs symbolically, as MTBDDs. The key components of a TSG, as required to perform model checking, are the transition probability function $\delta : S \times Act \to Dist(S)$ and the partition $(S_i)_{i \in N}$ of the state space amongst players. We consider two different symbolic encodings, one which represents δ and $(S_i)_i$ separately, and one which uses a single MTBDD.

For the first, we can use the standard approach for MDPs [8,1], which considers $\delta : S \times Act \to Dist(S)$ as a function $\delta' : S \times Act \times S \to [0, 1]$ in the obvious

way, i.e., for states s, s' and action a, we have $\delta'(s, a, s') = \delta(s, a)(s')$. Then, given an encoding $\mathsf{enc}_S : S \to \mathbb{B}^k$ of the state space into k Boolean variables, and an encoding $\mathsf{enc}_{Act} : Act \to \mathbb{B}^l$ of the action set into l Boolean variables, δ can be represented by an MTBDD over $2k + l$ variables. Reusing the same encoding enc_S, each set S_i is represented by a BDD over k variables.

For the second encoding, we assume that the TSG is represented by a single function $\delta'' : N \times S \times Act \times S \to [0, 1]$ such that, for player i, states s, s' and action a, $\delta''(i, s, a, s')$ equals $\delta(s, a)(s')$ if $s \in S_i$ and 0 otherwise. Given encodings $\mathsf{enc}_S : S \to \mathbb{B}^k$ and $\mathsf{enc}_{Act} : Act \to \mathbb{B}^l$ as above, plus an encoding $\mathsf{enc}_N : N \to \mathbb{B}^m$ of the player set, we can represent the TSG as an MTBDD over $2k + l + m$ Boolean variables. In our experiments, we found minimal difference between the two encodings, in terms of the size of storage for δ, but the first option incurs some additional overhead relating to the representation of the sets S_i. Hence, in this paper, we focus on the second, single-MTBDD encoding.

From now on, we will assume the use of variables $\underline{x} = (x_1, \ldots, x_k)$ and $\underline{y} = (y_1, \ldots, y_k)$ to encode the state space S (both x and y are used when representing the transition function; only one, usually $\underline{x}$, is needed when representing a subset of S or a real-valued vector indexed over S). We will use variables $\underline{z} = (z_1, \ldots, z_l)$ to encode actions and variables $\underline{w} = (w_1, \ldots, w_m)$ to encode players.

Example 1. Figure 1 shows a simple TSG with 2 players and its symbolic representation, using the second (single MTBDD) encoding described above. Top left is the TSG, in which player 1 states are drawn as diamonds and player 2 states as squares. Below that is a table explaining the representation: the details of each transition in the TSG and how it is encoded into Boolean variables.

For players, we use a one-hot encoding to two Boolean variables (w_1, w_2), i.e., $\mathsf{enc}_N(1) = (1, 0)$ and $\mathsf{enc}_N(2) = (0, 1)$. For the (two) actions, we use just a single variable z_1, where $\mathsf{enc}_{Act}(a) = (0)$ and $\mathsf{enc}_{Act}(b) = (1)$. The state space S is encoded with 2 variables using the usual binary encoding of the integer index i of each state s_i. In a transition, variables (x_1, x_2) and (y_1, y_2) represent the source and destination states, respectively.

To the right of the figure is the MTBDD representation. The 1 ("then") and 0 ("else") edges from each non-terminal node are drawn as solid and dashed lines, respectively. The zero terminal and edges to it are omitted for clarity. Each row of the table corresponds to a unique path through the MTBDD. The variable order used places $\underline{w}$ and $\underline{z}$ first, followed by $\underline{x}$ and $\underline{y}$, where, as usual in symbolic model checking, the variables in the latter two are interleaved. ∎

In order to be effective in practice, symbolic representations of TSGs need to be *constructed* in an efficient manner. In the context of this paper, we work with games that are described in the modelling language of the PRISM-games tool [38], which is inspired by the Reactive Modules formalism [2], proposed for specifying concurrent, multi-component systems.

We omit full details here, but note that this can be done by extending the existing approach used for the symbolic implementation of model checking for simpler probabilistic models in PRISM [34]. The basics for model construction
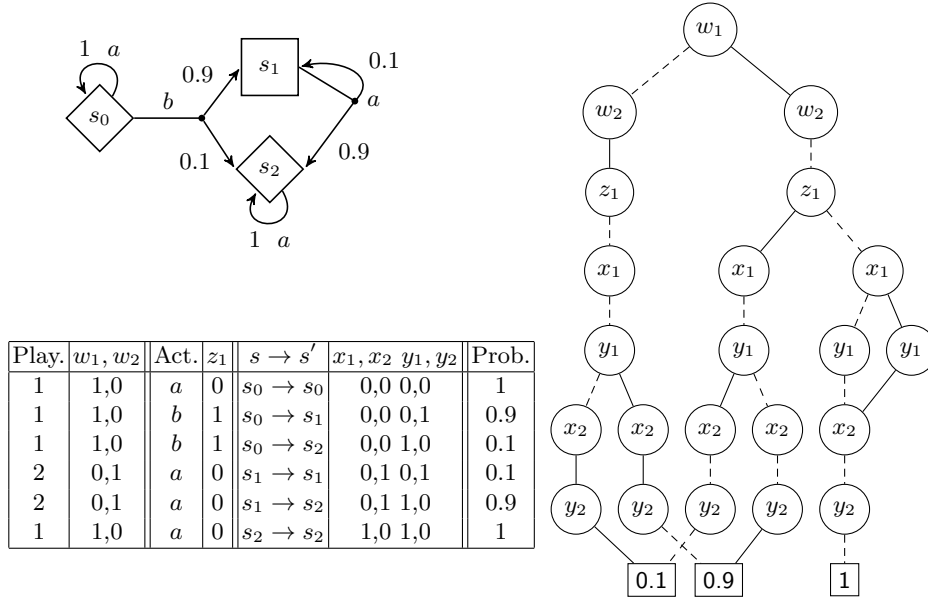
Play.	w_1, w_2	Act.	z_1	$s \to s'$	$x_1, x_2 \ y_1, y_2$	Prob.
1	1,0	a	0	$s_0 \to s_0$	0,0 0,0	1
1	1,0	b	1	$s_0 \to s_1$	0,0 0,1	0.9
1	1,0	b	1	$s_0 \to s_2$	0,0 1,0	0.1
2	0,1	a	0	$s_1 \to s_1$	0,1 0,1	0.1
2	0,1	a	0	$s_1 \to s_2$	0,1 1,0	0.9
1	1,0	a	0	$s_2 \to s_2$	1,0 1,0	1

Fig. 1. A TSG with its MTBDD representation and an explanation of the encoding.

from the PRISM modelling language can be found in [48]. The key idea is to construct the MTBDD in a compositional fashion, based on the structure of the model description. We also note that the second MTBDD encoding, building a single MTBDD, is better suited for this task, since it facilitates the detection of modelling errors (such as multiple players controlling actions in the same state).

3.2   Symbolic Model Checking of TSGs

Next, we describe a symbolic approach to performing probabilistic model checking of TSGs. We focus here on the PRISM-games logic described in Section 2.1. Essentially, since this is a branching-time logic, the model checking problem for a TSG $\mathcal{G}$ and a formula ϕ amounts to determining the set $Sat(\phi) = \{s \in S \mid s \models \phi\}$. Furthermore, this set is computed in a recursive fashion, following the structure of the parse tree of the formula ϕ.

In a symbolic setting, the set $Sat(\phi)$ will be represented as a BDD. The propositional fragment of the logic is treated in the usual way for symbolic model checking [13,45], using standard BDD implementations of Boolean operators. The key parts of the model checking algorithm are those for the `P` and `R` operators. In particular, we need to compute an MTBDD representing the real-valued vector of probability or expected reward values for each state s.

Computing these values can be done in a variety of ways. Here, we use *value iteration*, since iterative methods are known to be typically better suited to symbolic implementation [7,9,48]. This is because it requires minimal changes to

---

Algorithm 1 Value iteration and strategy synthesis for reachability probabilities

---

Input: BDD target (over variables $\underline{x}$) for set of target states
Output: MTBDD sol (over variables $\underline{x}$) giving the probability from each state, and a BDD strat (over variables $\underline{x}, \underline{z}'$) representing an optimal strategy

 1: **procedure** PROBREACH(target)
 2:   $S_0 \leftarrow$ PROB0(target)
 3:   $S_1 \leftarrow$ PROB1(target)
 4:   $S_? \leftarrow$ NOT(APPLY($\vee, S_1, S_0$))
 5:   trans$_? \leftarrow$ APPLY($\times$, trans, $S_?$)
 6:   sol $\leftarrow S_1$; *done* $\leftarrow$ **false**
 7: **while** $\neg done$ **do**
 8:     tmp $\leftarrow$ sol
 9:     sol $\leftarrow$ MVMULT(trans$_?$, sol)
10:     $p_1 \leftarrow$ OR($\{$CUBE(enc$_N(i), \underline{w}) : i \in N\}$)
11:     $p_2 \leftarrow$ OR($\{$CUBE(enc$_N(i), \underline{w}) : i \in N\backslash C\}$)
12:     sol$_1 \leftarrow$ ABSTRACT(max, ABSTRACT(+, APPLY($\times$, sol, p_1), $\underline{w}$), $\underline{z}$)
13:     sol$_2 \leftarrow$ ABSTRACT(min, ABSTRACT(+, APPLY($\times$, sol, p_2), $\underline{w}$), $\underline{z}$)
14:     sol $\leftarrow$ APPLY(+, sol$_1$, sol$_2$)
15:     sol $\leftarrow$ IFTHENELSE(S_1, CONST(1), sol)
16:     *done* $\leftarrow$ SUPNORM(sol, tmp) $< \varepsilon$
17: **end while**
18:   strat $\leftarrow$ APPLY($\approx$, MVMULT(trans$_?$, sol), sol)
19:   strat $\leftarrow$ ABSTRACT($\vee$, strat, $\underline{w}$)
20:   strat $\leftarrow$ REPLACEVARS(strat, $\underline{z}, \underline{z}'$)
21: **return** sol, strat
22: **end procedure**

---

be made to the representation of the model during solution, which could cause a blow-up in storage size due to the introduction of irregularities. This means that some alternative methods for solving stochastic games such as quadratic programming are unlikely to be well suited to a symbolic implementation.

To simplify presentation, we restrict our attention to computing reachability probabilities, assuming that they are maximised by a coalition of players C (and minimised by $N\backslash C$). The process for other computations, such as expected reward values, is similar. In other words, we consider the construction of an MTBDD sol representing a vector $\underline{sol}$ indexed over S with $sol(s) = val(s, \langle\!\langle C \rangle\!\rangle P_{\max=?}[\text{F } target])$ for some atomic proposition *target* labelling the states to be reached (see Section 2.1).

Algorithm 1 shows an MTBDD implementation that performs both the numerical solution, using value iteration, and synthesis of an optimal strategy. The input is a BDD target representing the target set $Sat(target)$, and we assume that MTBDD trans encodes the TSG. PROB0 and PROB1 are BDD-based implementations of the precomputation algorithms [4] for finding states with probability 0 and 1; we omit the details and focus on the numerical part.

The key part of value iteration can be done using matrix multiplication, treating the TSG as a non-square matrix with rows over $N \times Act \times S$ and

columns over S, followed by maximising and minimising over action choices for players in C and $N\backslash C$, respectively. Function CUBE($\underline{v}, \underline{\mathsf{w}}$) builds a *cube*, i.e., a BDD b over variables $\underline{\mathsf{w}}$ such that $f_{\mathsf{b}} = 1$ for precisely one valuation $\underline{v}$ of $\underline{\mathsf{w}}$. We check termination of value iteration using a function SUPNORM which performs a pointwise calculation of the relative difference for pairs of elements in two vectors represented as MTBDDs and returns the maximum difference. This is compared against a pre-specified convergence criterion threshold $\varepsilon \in \mathbb{R}_{>0}$.

Strategy synthesis. Lines 18-20 compute an optimal strategy, where $\approx$ represents an approximate equality check to the same level of accuracy as the convergence check (i.e., relative difference less than ε), and $\underline{\mathsf{z}}'$ is a fresh copy of the variables $\underline{\mathsf{z}}$ that encode actions, but appearing after $\underline{\mathsf{x}}$ and $\underline{\mathsf{y}}$ in the variable ordering. The result is a BDD strat over variables $\underline{\mathsf{x}}$ and $\underline{\mathsf{z}}$, representing an optimal strategy: for any state s, we traverse the top part of the BDD by following valuation $\mathsf{enc}_S(s)$. Any path (we allow multiple) from that node to the 1 terminal represents an optimal action a in that state (read from its encoding $\mathsf{enc}_{Act}(a)$). We leave as future work the possibility of selecting single optimal actions for states in a way that further reduces the size of the strategy representation.

Nash equilibria. Lastly, we briefly sketch how our symbolic model checking implementation also extends to nonzero-sum formulas, i.e., the synthesis of (social welfare) Nash equilibria. The process is again based on value iteration but, as mentioned in Section 2.1, this is simpler for TSGs than the CSG-based algorithm of [39]. Essentially we adapt Algorithm 1, first maximising for individual coalitions, as in the existing value iteration loop, then selecting all actions that are optimal, as in the strategy synthesis part, and then further maximising those choices over the sum of values for all players. The latter part means that we maintain a solution vector for each player as MTBDDs during the process. For the 2-coalition case, part of the computation reduces to symbolic model checking for MDPs, where we can reuse existing implementations.

4   Case Studies and Experimental Results

We have developed a symbolic implementation of model checking for TSGs within PRISM-games [38], leveraging parts of PRISM's existing symbolic engines for other models (Markov chains and Markov decision processes). This builds upon the CUDD decision diagram library by Fabio Somenzi, which supports both BDDs and MTBDDs, and a Java wrapper contained within PRISM which extends this library. Our experiments were carried out using a 2.10GHz Intel Xeon Gold with 16 GB maximum heap space for Java.

4.1   TSG Benchmarks

In order to evaluate the approach, we first present a set of benchmark TSG models. We have collated these and added them to the PRISM Benchmark Suite [35], which provides a selection of probabilistic models and associated properties for

performing model checking. To facilitate benchmarking, most models and properties are parameterised, allowing a wide range of model checking instances to be considered. Python scripts are also included to automate the process of selecting and executing instances, and for extracting information from tool logs.

The benchmarks are listed below:

- *avoid*: a TSG example from [20] modelling a game between an intruder and an observer in a grid-world (also used in [33]);
- *dice*: a simple 2-player dice game TSG distributed with PRISM-games;
- *hallway_human*: a TSG variant (from [20]) of a standard benchmark from the AI literature [40] modelling a robot moving through a hallway environment which is both probabilistic and adversarial (also used in [33]);
- *investors*: the futures market investor TSG example from [44], adapted and extended to more investors;
- *safe_nav*: a TSG modelling safe navigation in a human-robot system, from [29].
- *task_graph*: an extended version of the task-graph scheduling problem with faulty processors from [36], converted from a (turn-based) probabilistic timed game to a TSG using the digital clocks translation of [36].

4.2 Experimental Results

Table 1 shows statistics for a selection of TSG model instances that we use for our evaluation (see [51,35] for more details). We also give the time required to build a representation of the TSG, from its PRISM-games modelling language description, either symbolically, as an MTBDD, or explicitly, as a sparse matrix, as done in the existing implementation of PRISM-games. The faster time is in bold. For the symbolic case, we also show the MTBDD size. We see that the symbolic approach is considerably faster. The explicit implementation of model construction (in Java) is not highly optimised but the difference in performance is clear nonetheless. For some instances, where the explicit engine took several hours, the symbolic one requires no more than a few seconds.

Secondly, Table 2 shows the performance of model checking for the symbolic and explicit implementations on a range of example properties for the benchmarks (again, see [51,35] for full details). We break down the time required into qualitative analysis (graph-based precomputation) and quantitative analysis (numerical solution with value iteration). Again, the faster time is highlighted in bold. We also show the total memory required to store the resulting optimal strategy in each case (these are omitted for qualitative probabilistic reachability and expected reward, since they are not yet included in the implementation).

Pre-computation has shown to be more efficient for all model/property combinations in the table, and in some cases this is a decisive factor in terms of the overall model checking time. Results for value iteration generally favour the explicit engine, although there are instances where the symbolic one performs better. In terms of representing optimal strategies, we see that the symbolic one is more compact in all cases.

Case study [parameters]	Param. values	Players	States	MTBDD nodes	Constr. time(s) Symbolic	Explicit
avoid [X_MAX, Y_MAX]	10,10		106,524	19,298	**0.2**	1.6
	15,15	2	480,464	36,178	**0.4**	6.4
	20,20		1,436,404	69,407	**1.0**	18.8
dice [N]	10		5,755	1,717	**0.02**	0.2
	25	2	34,645	4,046	**0.04**	0.5
	50		136,795	7,958	**0.09**	1.5
hallway_human [X_MAX, Y_MAX]	5,5		25,000	1,334	**0.03**	0.6
	8,8	2	163,840	1,234	**0.04**	2.8
	10,10		400,000	1,752	**0.07**	6.7
investors [N, vmax]	2,10		172,240	5,846	**0.04**	2.0
	2,20	3	568,790	11,325	**0.06**	6.5
	2,40		2,041,690	22,191	**0.1**	23.4
	3,10		1,229,001	7,434	**0.06**	13.7
	3,20	4	4,058,751	12,913	**0.1**	48.5
	3,40		14,569,251	23,779	**0.2**	memout
safe_nav [N, feat]	8,D		2,592,845	28,008	**1.0**	1,602
	8,C	2	5,078,029	44,973	**1.7**	4,588
	8,B		8,732,493	67,735	**2.7**	10,010
	8,A		17,052,941	118,262	**4.8**	memout
task_graph [N, k1, k2]	6,10,10		467,638	19,881	**0.6**	6.7
	6,15,15	2	1,010,318	22,350	**1.0**	13.8
	6,20,20		1,759,348	22,350	**1.8**	25.1
	9,10,10		2,567,638	36,014	**1.4**	46.5
	9,15,15	2	5,533,288	36,745	**2.8**	100.0
	9,20,20		9,6231,38	39,349	**4.6**	169.2

Table 1. Model building statistics for the TSG case studies.

5 Conclusions

We have presented a symbolic version of probabilistic model checking for turn-based stochastic games, using BDDs and MTBDDs to implement model construction, model checking (via value iteration) and optimal strategy synthesis. There are some significant gains to be had, particularly in terms of model construction, but also further improvements to be made.

Future work includes studying different encodings more thoroughly, as well as variable orderings. There is also scope to investigate more efficient symbolic strategy representations. Another possibly interesting extension is providing support for Büchi, co-Büchi and Rabin-chain objectives [25], where a symbolic implementation could also allow for better scalability.

Case study [parameters] Property (type)	Param. values	Verification time and strategy memory							
		Symbolic				Explicit			
		Qual. (s)	Quant. (s)	Total (s)	Strat. (MB)	Qual. (s)	Quant. (s)	Total (s)	Strat. (MB)
avoid [X_MAX, Y_MAX] *exit* (P[F])	10,10	**4.4**	4.0	**8.4**	**0.1**	33.2	**0.9**	34.2	0.4
	15,15	**20.4**	23.0	**43.6**	**0.2**	407.2	**9.1**	416.3	1.8
	20,20	**77.7**	82.8	**161.2**	**0.4**	1,544	**14.5**	1,558	5.5
avoid [X_MAX, Y_MAX] *find* (P[F])	10,10	**8.3**	4.0	**12.3**	**0.2**	17.4	**0.5**	18.0	0.4
	15,15	**37.8**	21.9	**60.0**	**0.4**	224.9	**3.7**	228.7	1.8
	20,20	**152.7**	66.7	**220.0**	**0.7**	1,145	**9.2**	1,155	5.5
dice [N] *p1wins* (P[F])	10	**0.02**	**0.02**	**0.02**	**0.02**	0.07	0.04	0.1	0.02
	25	**0.2**	**0.2**	**0.4**	**0.05**	0.5	0.2	0.8	0.1
	50	**0.6**	**0.5**	**1.1**	**0.1**	5.3	2.6	7.9	0.5
hallway_human [X_MAX, Y_MAX] *save* (P[F])	5,5	**0.06**	-	**0.06**	-	0.2	-	0.2	-
	8,8	**0.2**	-	**0.2**	-	2.0	-	2.0	-
	10,10	**0.6**	-	**0.6**	-	6.9	-	7.0	-
investors [N, vmax] *greater* (P[F])	2,10	**0.04**	**1.0**	**1.1**	**0.06**	3.1	3.1	6.4	0.7
	2,20	**0.04**	**5.4**	**5.6**	**0.1**	19.7	22.3	42.2	2.2
	2,40	**0.05**	**22.6**	**22.8**	**0.2**	27.8	97.1	125.3	7.8
	3,10	**0.1**	**3.4**	**3.6**	**0.2**	27.6	30.9	58.9	4.7
	3,20	**0.1**	**16.0**	**16.2**	**0.3**	83.8	169.8	255.5	15.5
	3,40	**0.2**	**62.5**	**62.9**	**0.4**	-	-	memout	-
safe_nav [N, feat] *reach* (P[F])	8,D	**12.7**	4.1	**17.1**	**1.8**	134.9	**2.5**	138.0	9.9
	8,C	**26.1**	7.6	**34.4**	**2.9**	145.8	**3.6**	150.4	19.4
	8,B	**48.8**	12.0	**62.1**	**4.6**	313.8	**7.6**	323.1	33.3
	8,A	**138.7**	**27.1**	**169.3**	**8.7**	-	-	memout	-
task_graph [N, k1, k2] *time* (R[F])	6,10,10	**0.8**	116.8	117.7	-	14.3	**34.2**	**48.9**	-
	6,15,15	**1.1**	346.6	348.1	-	27.9	**63.4**	**91.8**	-
	6,20,20	**1.4**	826.5	828.6	-	52.9	**116.8**	**170.8**	-
	9,10,10	**4.1**	1,117	1,122	-	90.9	**179.3**	**271.5**	-
	9,15,15	**5.7**	3,304	3,312	-	250.9	**515.9**	**769.9**	-
	9,20,20	**7.8**	6,624	6,636	-	660.2	**1,268**	**1,934**	-

Table 2. Statistics for TSG verification instances.

References

1. de Alfaro, L., Kwiatkowska, M., Norman, G., Parker, D., Segala, R.: Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In: Proc. 6th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00). LNCS, vol. 1785, pp. 395–410. Springer (2000)
2. Alur, R., Henzinger, T.: Reactive modules. Formal Methods in System Design **15**(1), 7–48 (1999)
3. Alur, R., Henzinger, T., Mang, F., Qadeer, S., Rajamani, S., Tasiran, S.: MOCHA: Modularity in model checking. In: Proc. 10th Int. Conf. Computer Aided Verification (CAV'98). LNCS, vol. 1427, pp. 521–525. Springer, Vancouver (1998)

4. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49**(5), 672–713 (2002)
5. Amram, G., Maoz, S., Pistiner, O., Ringert, J.O.: Efficient algorithms for omega-regular energy games. In: Proc. 24th International Symposium on Formal Methods (FM'21). LNCS, vol. 13047, pp. 163–181. Springer (2019)
6. Aslanyan, Z., Nielson, F., Parker, D.: Quantitative verification and synthesis of attack-defence scenarios. In: Proc. 29th IEEE Computer Security Foundations Symposium (CSF'16). pp. 105–119. IEEE (2016)
7. Bahar, I., Frohm, E., Gaona, C., Hachtel, G., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. Formal Methods in System Design **10**(2/3), 171–206 (1997)
8. Baier, C.: On algorithmic verification methods for probabilistic systems (1998), habilitation thesis, Fakultät für Mathematik & Informatik, Universität Mannheim
9. Baier, C., Clarke, E., Hartonas-Garmhausen, V., Kwiatkowska, M., Ryan, M.: Symbolic model checking for probabilistic processes. In: Proc. 24th Int. Coll. Automata, Languages and Programming (ICALP'97). LNCS, vol. 1256, pp. 430–440. Springer (1997)
10. Batz, K., Junges, S., Kaminski, B.L., Katoen, J.P., Matheja, C., Schröer, P.: Pric3: Property directed reachability for MDPs. In: Computer Aided Verification. pp. 512–538. Springer International Publishing, Cham (2020)
11. Brázdil, T., Chatterjee, K., Chmelik, M., Fellner, A., Kretínský, J.: Counterexample explanation by learning small strategies in markov decision processes. In: Proc. 27th Int. Conf. Computer Aided Verification (CAV'15). LNCS, vol. 9206, pp. 158–177. Springer (2015)
12. Bryant, R.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers **C-35**(8), 677–691 (1986)
13. Burch, J., Clarke, E., McMillan, K., Dill, D., Hwang, J.: Symbolic model checking: 10^{20} states and beyond. In: Proc. 5th Annual IEEE Symposium on Logic in Computer Science (LICS'90). pp. 428–439. IEEE Computer Society Press (1990)
14. Caámara, J., Garlan, D., Schmerl, B., Pandey, A.: Optimal planning for architecture-based self-adaptation via model checking of stochastic games. In: Proc. 30th ACM Symposium on Applied Computing (SAC'15) (2015)
15. Chatterjee, K., Henzinger, T.: Value iteration. In: 25 Years of Model Checking. LNCS, vol. 5000, pp. 107–138. Springer (2008)
16. Chatterjee, K., Henzinger, T., Jobstmann, B., Radhakrishna, A.: Gist: A solver for probabilistic games. In: Proc. 22nd Int. Conf. Computer Aided Verification (CAV'10). pp. 665–669. LNCS, Springer (2010)
17. Chatterjee, K., Jurdzinski, M., Henzinger, T.: Quantitative stochastic parity games. In: Munro, J.I. (ed.) Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04). pp. 121–130. SIAM (2004)
18. Chatterjee, K., Henzinger, T.A.: A survey of stochastic ω-regular games. Journal of Computer and System Sciences **78**(2), 394–413 (2012)
19. Chatterjee, K., Henzingera, T.A., Piterman, N.: Strategy logic. Information and Computation **208**(6), 677–693 (2010)
20. Chatterjee, K., Katoen, J.P., Weininger, M., Winkler, T.: Stochastic games with lexicographic reachability-safety objectives. In: Proc. 32nd Int. Conf. Computer Aided Verification (CAV'20). LNCS, vol. 12225, pp. 398–420. Springer (2020)
21. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Formal Methods in System Design **43**(1), 61–92 (2013)

22. Clarke, E., Fujita, M., McGeer, P., McMillan, K., Yang, J., Zhao, X.: Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In: Proc. Int. Workshop Logic Synthesis (IWLS'93). pp. 1–15 (1993), also available in *Formal Methods in System Design*, 10(2/3):149–169, 1997

23. Clarke, E., McMillan, K., Zhao, X., Fujita, M., Yang, J.: Spectral transforms for large Boolean functions with applications to technology mapping. In: Proc. 30th Design Automation Conference (DAC'93). pp. 54–60. ACM Press (1993), also available in *Formal Methods in System Design*, 10(2/3):137–148, 1997

24. Condon, A.: The complexity of stochastic games. Information and Computation **96**(2), 203–224 (1992)

25. de Alfaro, L., Majumdar, R.: Quantitative solution of omega-regular games. Journal of Computer and System Sciences **68**(2), 374–397 (2004)

26. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A Storm is coming: A modern probabilistic model checker. In: Proc. CAV'17. LNCS, vol. 10427, pp. 592–600 (2017)

27. Feng, L., Wiltsche, C., Humphrey, L., Topcu, U.: Synthesis of human-in-the-loop control protocols for autonomous systems. IEEE Trans. Autom. Sci. Eng. **13**(2), 450–462 (2016)

28. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer (1997)

29. Junges, S., Jansen, N., Katoen, J.P., Topcu, U., Zhang, R., Hayhoe, M.: Model checking for safe navigation among humans. In: Proc. 15th International Conference on Quantitative Evaluation of SysTems (QEST'18). LNCS, vol. 11024, pp. 207–222. Springer (2018)

30. Kelmendi, E., Krämer, J., Kretínský, J., Weininger, M.: Value iteration for simple stochastic games: Stopping criterion and learning algorithm. In: Proc. 30th International Conference on Computer Aided Verification (CAV'18). LNCS, vol. 10981, pp. 623–642. Springer (2018)

31. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. Springer (1976)

32. Klein, J., Baier, C., Chrszon, P., Daum, M., Dubslaff, C., Klüppelholz, S., Märcker, S., Müller, D.: Advances in symbolic probabilistic model checking with PRISM. In: Proc. 22nd Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'16). LNCS, vol. 9636, pp. 349–366. Springer (2016)

33. Kretínský, J., Ramneantu, E., Slivinskiy, A., Weininger, M.: Comparison of algorithms for simple stochastic games. In: Proc. 11th Int. Symp. Games, Automata, Logics, and Formal Verification (GandALF'20). pp. 131–148. EPTCS (2020)

34. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. 23rd Int. Conf. Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591. Springer (2011)

35. Kwiatkowska, M., Norman, G., Parker, D.: The PRISM benchmark suite. In: Proc. 9th Int. Conf. Quantitative Evaluation of SysTems (QEST'12). pp. 203–204. IEEE CS Press (2012), https://www.prismmodelchecker.org/benchmarks/

36. Kwiatkowska, M., Norman, G., Parker, D.: Verification and control of turn-based probabilistic real-time games. In: The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy (Essays Dedicated to Catuscia Palamidessi on the Occasion of Her 60th Birthday). LNCS, vol. 11760, pp. 379–396. Springer (2019)

37. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: Multi-player equilibria verification for concurrent stochastic games. In: Proc. 17th Int. Conf. Quantitative Evaluation of SysTems (QEST'20). LNCS, vol. 12289, pp. 74–95. Springer (2020)

38. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: PRISM-games 3.0: Stochastic game verification with concurrency, equilibria and time. In: Proc. 32nd Int. Conf. Computer Aided Verification (CAV'20). LNCS, vol. 12225, pp. 475–487. Springer (2020)
39. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: Automatic verification of concurrent stochastic systems. Formal Methods in System Design pp. 1–63 (2021)
40. Littman, M., Cassandra, A., Kaelbling, L.: Learning policies for partially observable environments: Scaling up. In: Proc. 12th Int. Conf. Machine Learning (ICML'95). pp. 362–370 (1995)
41. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A model checker for the verification of multi-agent systems. In: Proc. 21st Int. Conf. Computer Aided Verification (CAV'09). LNCS, vol. 5643, pp. 682–688. Springer (2009)
42. Majumdar, R., Mallik, K., Schmuck, A.K., Soudjani, S.: Symbolic qualitative control for stochastic systems via finite parity games. IFAC **54**(5), 127–132 (2021)
43. Martin, D.: The determinacy of Blackwell games. Journal of Symbolic Logic **63**(4), 1565–1581 (1998)
44. McIver, A., Morgan, C.: Results on the quantitative mu-calculus qMu. ACM Trans. Computational Logic **8**(1) (2007)
45. McMillan, K.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
46. von Neumann, J., Morgenstern, O., Kuhn, H., Rubinstein, A.: Theory of Games and Economic Behavior. Princeton University Press (1944)
47. Osborne, M., Rubinstein, A.: An Introduction to Game Theory. Oxford University Press (2004)
48. Parker, D.: Implementation of Symbolic Model Checking for Probabilistic Systems. PhD thesis, University of Birmingham (2002)
49. Shapley, L.: Stochastic games. In: Proc. National Academy of Science. vol. 39, pp. 1095–1100 (1953)
50. Zamani, Z., Sanner, S., Fang, C.: Symbolic dynamic programming for continuous state and action MDPs. In: Proc. AAAI'12. pp. 1839–1845. AAAI Press (2012)
51. Supporting material, https://www.prismmodelchecker.org/files/pgsym/