Automated Verification of Concurrent Stochastic Games

Marta Kwiatkowska[1], Gethin Norman[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. We present automatic verification techniques for concurrent stochastic multi-player games (CSGs) with rewards. To express properties of such models, we adapt the temporal logic rPATL (probabilistic alternating-time temporal logic with rewards), originally introduced for the simpler model of turn-based games, which enables quantitative reasoning about the ability of coalitions of players to achieve goals related to the probability of an event or reward measures. We propose and implement a modelling approach and model checking algorithms for property verification and strategy synthesis of CSGs, as an extension of PRISM-games. We evaluate the performance, scalability and applicability of our techniques on case studies from domains such as security, networks and finance, showing that we can analyse systems with probabilistic, cooperative and competitive behaviour between concurrent components, including many scenarios that cannot be analysed with turn-based models.

1 Introduction

Stochastic multi-player games are a versatile modelling framework for systems that exhibit cooperative and competitive behaviour in the presence of adversarial or uncertain environments. They can be viewed as a collection of players (agents) with strategies for determining their actions based on the execution so far. These models combine nondeterminism, representing the adversarial, cooperative and competitive choices, stochasticity, modelling uncertainty due to noise, failures or randomness, and concurrency, representing simultaneous execution of interacting agents. Examples of such systems appear in many domains, from robotics and autonomous transport, to security and computer networks.

Formal verification for stochastic games provide a means of producing quantitative guarantees on the correctness of a system (e.g. "the control software can always safely stop the vehicle with probability at least 0.99, regardless of the actions of other road users"), where the required behavioural properties are specified precisely in quantitative extensions of temporal logic. The closely related problem of *strategy synthesis* constructs an optimal strategy for a player, or coalition of players, that guarantees a property is satisfied.

Automatic verification and strategy synthesis for models exhibiting nondeterminism and stochasticity are well established and implemented, e.g., in tools

such as PRISM [14] and STORM [11]. Recently, these techniques have also been formulated and implemented in PRISM-games [16], an extension of PRISM, for *turn-based* stochastic multi-player games (TSGs), which can be viewed as Markov decision processes whose states are partitioned among a set of players, with exactly one player taking control of each state. Properties are specified in the logic rPATL (probabilistic alternating-time temporal logic with rewards) [9], a quantitative extension of the game logic ATL [4]. This allows us to specify that a coalition of players achieve a high-level objective, regarding the probability of an event's occurrence or the expectation of a cumulative reward measure, irrespective of the strategies of the other players.

Concurrent stochastic multi-player games (CSGs) generalise TSGs by permitting players to choose their actions concurrently in each state of the model. This can provide a more realistic model of interactive agents operating concurrently, and making action choices without already knowing the actions that are being taken by other agents. However, although algorithms for verification and strategy synthesis of CSGs have been known for some time (e.g., [2,3,6]), their implementation and application to real-world examples is lacking.

This paper develops the first approach to modelling, verification and strategy synthesis for CSGs that is implemented in software and applied to a selection of in-depth case studies. We first adapt the logic rPATL to CSGs and provide a formal semantics. Then, we propose a model checking algorithm, building upon the existing techniques for TSGs and adapting to CSGs by integrating techniques for solving matrix games. Next, we develop an approach to modelling CSGs as an extension of the PRISM-games model checking tool and implement algorithms for construction, verification and strategy synthesis.

Finally, we investigate the performance, scalability and applicability of our implementation using a selection of real-life case studies. We demonstrate that our CSG modelling and verification techniques facilitate insightful analysis of quantitative aspects of systems taken from diverse set of application domains: finance, computer security, computer networks and communication systems. These illustrate examples of systems whose modelling and analysis requires stochasticity *and* competitive or adversarial behaviour between concurrent components or agents, as provided by CSGs; in several cases, we explicitly highlight the differences between our use of CSGs and existing models verified using TSGs.

Related work. Various verification algorithms have been proposed for CSGs, e.g. [2,3,6], but without implementations, tool support or case studies. PRISM-games [16], upon which we build in this work, provides modelling and verification for a wide range of properties of stochastic multi-player games, including those in the logic rPATL, and multi-objective extensions of it, but focuses purely on the turn-based variant of the model (TSGs). GIST [8] allows the analysis of ω-regular properties on probabilistic games, but again focuses on turn-based, not concurrent, games. GAVS+ [10] is a general-purpose tool for algorithmic game solving, supporting TSGs and (non-stochastic) concurrent games, but not CSGs. Two further tools, PRALINE [5] and EAGLE [25], allow the computation of Nash equilibria [20] for the restricted class of (non-stochastic) concurrent games.

2 Preliminaries

We begin with some basic background from game theory, and then describe concurrent stochastic games. For any finite set X, we will write $Dist(X)$ for the set of probability distributions over X. We first introduce *normal form games*, which are simple one-shot games where players make their choices concurrently.

Definition 1 (Normal form game). *A (finite, n-person)* normal form game *(also known as strategic form) is a tuple* $\mathsf{N} = (N, A, u)$ *where:*

- $N = \{1, \ldots, n\}$ *is a finite set of players;*
- $A = A_1 \times \cdots \times A_n$ *and* A_i *is a finite set of actions available to player* $i \in N$*;*
- $u = (u_1, \ldots, u_n)$ *and* $u_i : A \to \mathbb{R}$ *is a utility function for player* $i \in N$*.*

In a game N, players select actions simultaneously, with player $i \in N$ choosing from the action set A_i. If each player i selects action a_i, then player j receives the utility $u_j(a_1, \ldots, a_n)$. A (mixed) strategy σ_i for player i is a distribution over its actions, i.e. $\sigma_i \in Dist(A_i)$. Let Σ^i_{N} denote the set of strategies for player i. A *strategy profile* is a tuple of strategies for each player. Under a strategy profile $\sigma{=}(\sigma_1, \ldots, \sigma_n)$, the expected utility of player i is defined as follows:

$$u_i(\sigma) \stackrel{\text{def}}{=} \sum_{(a_1,\ldots,a_n)\in A} u_i(a_1, \ldots, a_n) \cdot \left(\prod_{j=1}^{n} \sigma_j(a_j)\right) .$$

A two-player normal form game $\mathsf{N} = (N, A, u)$ is *zero-sum* if for each action tuple $\alpha \in A$ we have $u_1(\alpha) + u_2(\alpha) = 0$. Such a game is often called a *matrix game* as it can be represented by a matrix $\mathsf{Z} \in \mathbb{R}^{l\times m}$ where $A_1 = \{a_1, \ldots, a_l\}$, $A_2 = \{b_1, \ldots, b_m\}$ and $z_{ij} = u_1(a_i, b_j) = -u_2(a_i, b_j)$.

We require the following classical result concerning matrix games.

Theorem 1 (Minimax theorem [21,22]). *For any matrix game* $\mathsf{Z} \in \mathbb{R}^{l\times m}$*, there exists* $v^\star \in \mathbb{R}$*, called the* value *of the game and denoted* $val(\mathsf{Z})$*, such that:*

- *there is a strategy* $\sigma_1^\star$ *for player 1, called an optimal strategy of player 1, such that under this strategy the player's expected utility is at least* $v^\star$ *regardless of the strategy of player 2, i.e.* $\inf_{\sigma_2\in\Sigma^2_{\mathsf{N}}} u_1(\sigma_1^\star, \sigma_2) \geqslant v^\star$*;*
- *there is a strategy* $\sigma_2^\star$ *for 2, called an optimal strategy of player 2, such that under this strategy the player's expected utility is at least* $v^\star$ *regardless of the strategy of player 1, i.e.* $\inf_{\sigma_1\in\Sigma^1_{\mathsf{N}}} u_2(\sigma_1, \sigma_2^\star) \geqslant v^\star$*.*

The value of a matrix game $\mathsf{Z} \in \mathbb{R}^{l\times m}$ can be found by solving the following linear programming (LP) problem [21,22]. Maximise v subject to the constraints:

$$v \leqslant p_1{\cdot}z_{1j} + \cdots + p_l{\cdot}z_{lj} \;\text{ for } 1 \leqslant j \leqslant m$$
$$p_i \geqslant 0 \;\text{ for } 1 \leqslant i \leqslant l \;\text{ and }\; p_1 + \cdots + p_l = 1$$

In addition, the solution for $(p_1, \ldots, p_l)$ yields an optimal strategy for player 1. The value of the game can also be calculated as the solution of the following dual LP problem. Minimise v subject to the constraints:

$$v \geqslant q_1{\cdot}z_{i1} + \cdots + q_m{\cdot}z_{im} \;\text{ for } 1 \leqslant i \leqslant l$$
$$q_j \geqslant 0 \;\text{ for } 1 \leqslant j \leqslant m \;\text{ and }\; q_1 + \cdots + q_m = 1$$

and the solution $(q_1, \ldots, q_m)$ yields an optimal strategy for player 2.

We next introduce *concurrent stochastic games*, in which players repeatedly make choices simultaneously to determine the next state of the game.

Definition 2 (Concurrent stochastic game). *A* concurrent stochastic multi-player game *(CSG) is a tuple* $\mathsf{G} = (N, S, \bar{s}, A, \Delta, \delta, AP, L)$ *where:*

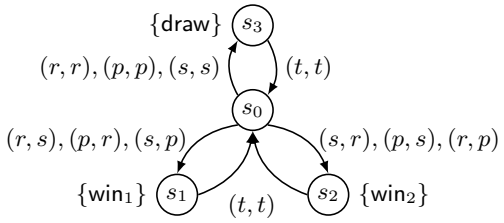
- $N = \{1, \ldots, n\}$ *is a finite set of players;*
- S *is a finite set of states and* $\bar{s} \in S$ *is an initial state;*
- $A = (A_1 \cup \{\bot\}) \times \cdots \times (A_n \cup \{\bot\})$ *where* A_i *is a finite set of actions available to player* $i \in N$ *and* $\bot$ *is an idle action disjoint from the set* $\cup_{i=1}^{n} A_i$*;*
- $\Delta : S \to \cup_{i=1}^{n} A_i$ *is an action assignment function;*
- $\delta : S \times A \to \mathit{Dist}(S)$ *is a probabilistic transition function;*
- AP *is a set of atomic propositions and* $L : S \to 2^{AP}$ *is a labelling function.*

In any state s of a CSG G, each player $i \in N$ chooses an action from the set $A_i(s) \stackrel{\text{def}}{=} \Delta(s) \cap A_i$ if this set is non-empty and $\{\bot\}$ otherwise. If each player i selects action a_i, then the next state of the game is chosen according to the probability distribution $\delta(s, (a_1, \ldots, a_n))$. TSGs are a restricted class of CSGs for which in any state s there is a unique player i such that $A_i(s) \neq \{\bot\}$.

A path π of a CSG G is a sequence $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \cdots$ where $s_i \in S$, $\alpha_i = (a_1^i, \ldots, a_n^i) \in A$, $a_j^i \in A_j(s_i)$ for $j \in N$ and $\delta(s_i, \alpha_i)(s_{i+1}) > 0$ for all $i \geqslant 0$. We denote by $\pi(i)$ the $(i{+}1)$th state of π, $\pi[i]$ the action associated with the $(i{+}1)$th transition and, if π is finite, $\mathit{last}(\pi)$ the final state. The length of a path π, denoted $|\pi|$, is the number of transitions appearing in the path. Let $\mathit{FPaths}_{\mathsf{G}}$ and $\mathit{IPaths}_{\mathsf{G}}$ ($\mathit{FPaths}_{\mathsf{G},s}$ and $\mathit{IPaths}_{\mathsf{G},s}$) be the sets of finite and infinite paths (starting in state s). A *strategy* σ_i for player i of G is a way of resolving the choices of player i based on the execution so far. Formally, a strategy for player i is a function $\sigma_i : \mathit{FPaths}_{\mathsf{G}} \to \mathit{Dist}(A_i)$ such that if $\sigma_i(\pi)(a_i) > 0$, then $a_i \in A_i(\mathit{last}(\pi))$. The set of all strategies of player i is denoted by Σ_{G}^i. A strategy for player i is deterministic if it always selects actions with probability 1 and memoryless if it makes the same choice for paths that end in the same state.

A strategy profile for CSG G is a tuple $\sigma = (\sigma_1, \ldots, \sigma_n) \in \Sigma_{\mathsf{G}}^1 \times \cdots \times \Sigma_{\mathsf{G}}^n$ yielding a strategy for each player of the game. We use $\mathit{FPaths}_{\mathsf{G},s}^{\sigma}$ and $\mathit{IPaths}_{\mathsf{G},s}^{\sigma}$ for the sets of finite and infinite paths corresponding to the choices made by the strategy profile σ when starting in state s. For a given strategy profile σ and starting state s, the behaviour of G is fully probabilistic and we can define a probability measure $\mathit{Prob}_{\mathsf{G},s}^{\sigma}$ over the set of infinite paths $\mathit{IPaths}_{\mathsf{G},s}^{\sigma}$ [13]. Given a random variable $X : \mathit{IPaths}_{\mathsf{G}} \to \mathbb{R}$, the expected value of X with respect to profile σ when starting in state s is given by $\mathbb{E}_{\mathsf{G},s}^{\sigma}(X) \stackrel{\text{def}}{=} \int_{\pi \in \mathit{IPaths}_{\mathsf{G},s}^{\sigma}} X(\pi) \, \mathrm{d}\mathit{Prob}_{\mathsf{G},s}^{\sigma}$.

We augment CSGs with *reward structures* of the form $r = (r_A, r_S)$ where $r_A : S \times A \to \mathbb{R}$ is an action reward function (which maps each state and action tuple pair to a real value that is accumulated when the action tuple is selected in the state) and $r_S : S \to \mathbb{R}$ is a state reward function (which maps each state to a real value that is accumulated when the state is reached). We allow both



(a) Rock-paper-scissors CSG.

$$\begin{array}{c} \\ r \\ p \\ s \end{array} \begin{array}{c} \begin{array}{ccc} r & p & s \end{array} \\ \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \end{array}$$

(b) Rock-paper-scissors matrix game.

positive and negative rewards; however, we will later require certain restrictions to ensure the correctness of the presented model checking algorithm.

Example 1. Consider the CSG shown in Figure 1(a) corresponding to two players repeatedly playing the rock-paper-scissors game. Transitions are labelled with action-pairs where $A_1 = A_2 = \{r, p, s, t\}$, with r, p and s representing playing rock, paper and scissors, respectively, and t restarting the game. The CSG starts in state s_0 and states s_1, s_2 and s_3 are labelled with atomic propositions corresponding to when a player wins or there is a draw in a round of the rock-paper-scissors game. The matrix game representation of the rock-paper-scissors game is presented in Figure 1(b). The optimal value for the matrix game is the solution to the following LP problem. Maximise v subject to the constraints:

$$v \leqslant p_2 - p_3,\ v \leqslant p_3 - p_1,\ v \leqslant p_1 - p_2,\ p_1 + p_2 + p_3 = 1,\ p_1, p_2, p_3 \geqslant 0$$

which yields the solution $v^\star = 0$ with corresponding optimal strategy $\sigma_1^\star = (1/3, 1/3, 1/3)$ for player 1 (the optimal strategy for player 2 is the same).

3 rPATL for Concurrent Stochastic Games

In this section, we discuss the temporal logic rPATL, previously proposed for specifying properties of TSGs [9], and adapt it to CSGs.

Definition 3 (rPATL syntax). *The syntax of* rPATL *is given by the grammar:*

$$\begin{array}{rcl} \phi & \coloneqq & \mathtt{true} \mid \mathsf{a} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,] \mid \langle\!\langle C \rangle\!\rangle \mathtt{R}^r_{\sim x}[\,\rho\,] \\ \psi & \coloneqq & \mathtt{X}\,\phi \mid \phi\ \mathtt{U}^{\leqslant k}\ \phi \mid \phi\ \mathtt{U}\ \phi \\ \rho & \coloneqq & \mathtt{I}^{=k} \mid \mathtt{C}^{\leqslant k} \mid \mathtt{C} \mid \mathtt{F}^c\ \phi \end{array}$$

where $\mathsf{a} \in AP$ *is an atomic proposition,* $C \subseteq N$ *is a coalition of players,* $\sim \in \{<, \leqslant, \geqslant, >\}$, $q \in [0,1]$, $x \in \mathbb{R}$, r *is a reward structure and* $k \in \mathbb{N}$.

The rPATL syntax distinguishes between state (ϕ), path (ψ) and reward (ρ) formulae. State formulae are evaluated over states of a CSG, while path and reward formulae are both evaluated over paths. A state satisfies a formula $\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,]$ if the *coalition* of players C can ensure the probability of the path formula ψ being satisfied is $\sim q$, and satisfies a formula $\langle\!\langle C \rangle\!\rangle \mathtt{R}^r_{\sim x}[\,\rho\,]$ if the players in C can ensure the expected value of the reward formula ρ for rewards structure r is

$\sim x$. For path formulae, we allow next ($\mathtt{X}\,\phi$), time-bounded until ($\phi\ \mathtt{U}^{\leqslant k}\ \phi$) and unbounded until ($\phi\ \mathtt{U}\ \phi$). For reward formulae, we allow instantaneous (state) reward at the kth step ($\mathtt{I}^{=k}$), reward accumulated over k steps ($\mathtt{C}^{\leqslant k}$), total cumulative reward ($\mathtt{C}$) and reward accumulated until a formula is satisfied ($\mathtt{F}^{c}\ \phi$).

There are two differences from the rPATL syntax of [9]. First, we add several reward operators ($\mathtt{I}^{=k}$, $\mathtt{C}^{\leqslant k}$, $\mathtt{C}$), adapted from the property specification language of PRISM [14]. These proved to be useful for the case studies we present later in Section 6. On the other hand, for simplicity, we restricted our attention to a single variant (c) of the reward operator $\mathtt{F}^{\star}\ \phi$: two other variants are included in [9], labelled 0 and ∞, which define the accumulated reward to be 0 or infinity, respectively, if no state satisfying ϕ is reached along a path. We intend to add these variants to our CSG verification implementation at a later date.

To introduce the semantics of rPATL, for any CSG G and coalition of players $C \subseteq N$, we require the following definition of a two-player *coalition game*. Without loss of generality, we assume the coalition is of the form $C = \{1, \ldots, n'\}$.

Definition 4 (Coalition game). *For CSG $\mathsf{G} = (N, S, \bar{s}, A, \Delta, \delta, AP, L)$ and coalition $C = \{1, \ldots, n'\} \subseteq N$, the* coalition game *induced by C of G is the two-player game $\mathsf{G}^C = (\{1,2\}, S, \bar{s}, A^C, \Delta, \delta^C, AP, L)$ where:*

- *$A^C = A_1^C \times A_2^C$, $A_1^C = A_1 \times \cdots \times A_{n'}$ and $A_2^C = A_{n'+1} \times \cdots \times A_n$;*
- *for any $s \in S$, $a_1^C = (a_1, \ldots, a_{n'}) \in A_1^C$ and $a_2^C = (a_{n'+1}, \ldots, a_n) \in A_2^C$ we have $\delta^C(s, (a_1^C, a_2^C)) = \delta(s, a_1, \ldots, a_n)$.*

Furthermore, for any reward structure $r = (r_A, r_S)$ of G, the corresponding reward structure $r^C = (r_A^C, r_S^C)$ of G^C is constructed in a similar manner.

Definition 5 (rPATL semantics). *For CSG G the satisfaction relation $\models$ for* rPATL *is defined inductively on the structure of the formula. The cases of* `true`*, atomic propositions, negations and conjunction of formulae are defined in the usual way. For temporal operators and $s \in S$ we have:*

$$s \models \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,] \;\Leftrightarrow\; \exists \sigma_1 \in \Sigma^1_{\mathsf{G}^C}.\, \forall \sigma_2 \in \Sigma^2_{\mathsf{G}^C}.\, \mathbb{P}^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s}(\psi) \sim q$$
$$s \models \langle\!\langle C \rangle\!\rangle \mathtt{R}^{r}_{\sim x}[\,\rho\,] \;\Leftrightarrow\; \exists \sigma_1 \in \Sigma^1_{\mathsf{G}^C}.\, \forall \sigma_2 \in \Sigma^2_{\mathsf{G}^C}.\, \mathbb{E}^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s}[rew(r^C, \rho)] \sim x$$

where $\mathbb{P}^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s}(\psi) = Prob^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s}\{\pi \in IPaths^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s} \mid \pi \models \psi\}$ and for $\pi \in IPaths_{\mathsf{G}^C,s}$:

$$\begin{aligned}
\pi \models \mathtt{X}\,\phi &\;\Leftrightarrow\; \pi(1) \models \phi \\
\pi \models \phi_1\ \mathtt{U}^{\leqslant k}\ \phi_2 &\;\Leftrightarrow\; \exists i \leqslant k.\, (\pi(i) \models \phi_2 \wedge \forall j < i.\, \pi(j) \models \phi_1) \\
\pi \models \phi_1\ \mathtt{U}\ \phi_2 &\;\Leftrightarrow\; \exists i \in \mathbb{N}.\, (\pi(i) \models \phi_2 \wedge \forall j < i.\, \pi(j) \models \phi_1) \\
rew(r^C, \mathtt{I}^{=k})(\pi) &= r_S(\pi(k)) \\
rew(r^C, \mathtt{C}^{\leqslant k})(\pi) &= \textstyle\sum_{i=0}^{k-1} \big(r_A(\pi(i), \pi[i]) + r_S(\pi(i))\big) \\
rew(r^C, \mathtt{C})(\pi) &= \textstyle\sum_{i=0}^{\infty} \big(r_A(\pi(i), \pi[i]) + r_S(\pi(i))\big) \\
rew(r^C, \mathtt{F}^{c}\ \phi)(\pi) &= \begin{cases} \sum_{i=0}^{\infty} \big(r_A(\pi(i), \pi[i]) + r_S(\pi(i))\big) & \text{if } \forall j \in \mathbb{N}.\, \pi(j) \not\models \phi \\ \sum_{i=0}^{k_\phi} \big(r_A(\pi(i), \pi[i]) + r_S(\pi(i))\big) & \text{otherwise} \end{cases}
\end{aligned}$$

and $k_\phi = \min\{k-1 \mid \pi(k) \models \phi\}$.

For CSG G, coalition C, state s, path formula ψ, reward structure r and reward formula ρ, we define the following optimal values of G^C:

$$\begin{array}{rcl} \mathbb{P}^C_{\mathsf{G},s}(\psi) & \stackrel{\text{def}}{=} & \sup_{\sigma_1 \in \Sigma^1_{\mathsf{G}^C}} \inf_{\sigma_2 \in \Sigma^2_{\mathsf{G}^C}} \mathbb{P}^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s}(\psi) \\ \mathbb{E}^C_{\mathsf{G},s}(r,\rho) & \stackrel{\text{def}}{=} & \sup_{\sigma_1 \in \Sigma^1_{\mathsf{G}^C}} \inf_{\sigma_2 \in \Sigma^2_{\mathsf{G}^C}} \mathbb{E}^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s}(r,\rho)\,. \end{array} \tag{1}$$

As in the case of TSGs [9], negated path formulae can be represented 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 to be specified: $\mathtt{G}\ \phi \equiv \neg(\mathtt{F}\ \neg\phi)$.

As for other probabilistic temporal logics, it is useful to consider *numerical* state formulae of the form $\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\min=?}[\,\psi\,]$, $\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\max=?}[\,\psi\,]$, $\langle\!\langle C \rangle\!\rangle \mathtt{R}^r_{\min=?}[\,\rho\,]$ and $\langle\!\langle C \rangle\!\rangle \mathtt{R}^r_{\max=?}[\,\rho\,]$. For example, for state s we have:

$$\begin{array}{rcl} \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\min=?}[\,\psi\,] & \stackrel{\text{def}}{=} & \inf_{\sigma_1 \in \Sigma^1_{\mathsf{G}^C}} \sup_{\sigma_2 \in \Sigma^2_{\mathsf{G}^C}} \mathbb{P}^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s}(\psi) \\ \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\max=?}[\,\psi\,] & \stackrel{\text{def}}{=} & \sup_{\sigma_1 \in \Sigma^1_{\mathsf{G}^C}} \inf_{\sigma_2 \in \Sigma^2_{\mathsf{G}^C}} \mathbb{P}^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s}(\psi)\,. \end{array}$$

As CSGs are *determined* with respect to the properties we consider [18], i.e. the maximum value coalition C can ensure equals the minimum value coalition $N\backslash C$ can ensure, the above values are the *optimal values* for the respective properties in G. The determinacy result also yields the following equivalences:

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

Example 2. Returning to Example 1, we can use rPATL to specify the following properties of the rock-paper-scissors CSG:

- $\langle\!\langle \{1\} \rangle\!\rangle \mathtt{P}_{\geqslant 1}[\,\mathtt{F}\ \mathsf{win}_1\,]$ player 1 can ensure it eventually wins a round of the game with probability 1;
- $\langle\!\langle \{2\} \rangle\!\rangle \mathtt{P}_{\max=?}[\,\neg\mathsf{win}_1\ \mathtt{U}\ \mathsf{win}_2\,]$ the maximum probability with which player 2 can ensure it wins a round of the game before player 1;
- $\langle\!\langle \{2\} \rangle\!\rangle \mathtt{R}^{utility}_{\max=?}[\,\mathtt{C}^{\leqslant 2\cdot K}\,]$ the maximum expected utility player 2 can ensure over K rounds (*utility* is the reward structure that assigns rewards 1, 0 and -1 to the states labelled win_2, draw and win_1, respectively).

4 Model Checking CSGs against rPATL

Next, we present an algorithm for model checking an rPATL formula ϕ against a CSG G. The overall approach is the standard one for branching-time logics, i.e., determining the set $Sat(\phi)$ recursively. The computation of this set for atomic propositions and logical connectives is straightforward, and therefore we concentrate on formulae of the form $\langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,]$ and $\langle\!\langle C \rangle\!\rangle \mathtt{R}^r_{\sim x}[\,\rho\,]$. For these, the problem reduces to computing optimal values for the coalition game G^C, see (1). In particular, for $\sim \in \{\geqslant, >\}$ and $s \in S$ we have:

$$s \models \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,] \ \Leftrightarrow\ \mathbb{P}^C_{\mathsf{G},s}(\psi) \sim q \quad \text{and} \quad s \models \langle\!\langle C \rangle\!\rangle \mathtt{R}^r_{\sim x}[\,\rho\,] \ \Leftrightarrow\ \mathbb{E}^C_{\mathsf{G},s}(\rho) \sim x\,.$$

and, since CSGs are determined for rPATL properties, for $\sim \in \{<, \leqslant\}$ we have:

$$s \models \langle\!\langle C \rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,] \;\Leftrightarrow\; \mathbb{P}^{N\setminus C}_{\mathsf{G},s}(\psi) \sim q \quad \text{and} \quad s \models \langle\!\langle C \rangle\!\rangle \mathtt{R}^{r}_{\sim x}[\,\rho\,] \;\Leftrightarrow\; \mathbb{E}^{N\setminus C}_{\mathsf{G},s}(\rho) \sim x\,.$$

Therefore, for the remainder of this section, we focus on computing $\mathbb{P}^{C}_{\mathsf{G},s}(\psi)$ and $\mathbb{E}^{C}_{\mathsf{G},s}(\rho)$ for a fixed CSG G, coalition C and state s. We assume that the available actions of players 1 and 2 of the (two-player) CSG G^C in state s are $\{a_1, \ldots, a_l\}$ and $\{b_1, \ldots, b_m\}$, respectively.

Matrix games. The computation of optimal probabilities and expected reward values requires finding values of matrix games. These values can be found through the solution to an LP problem as presented in Section 2. Solution methods for such problems include Simplex and branch-and-bound.

Probabilistic formulae. We now show how to compute the optimal probability $\mathbb{P}^{C}_{\mathsf{G},s}(\psi)$ for each state s and path formula ψ. If $\psi = \mathtt{X}\,\phi$, then for any state s we have that $\mathbb{P}^{C}_{\mathsf{G},s}(\mathtt{X}\,\phi) = \mathit{val}(\mathsf{Z})$ where $\mathsf{Z} \in \mathbb{R}^{l\times m}$ is the matrix game with:

$$z_{i,j} = \textstyle\sum_{s' \in Sat(\phi)} \delta(s,(a_i,b_j))(s')\,.$$

If $\psi = \phi_1\ \mathtt{U}^{\leqslant k}\ \phi_2$, the values can be computed recursively as follows.

- $\mathbb{P}^{C}_{\mathsf{G},s}(\phi_1\ \mathtt{U}^{\leqslant 0}\ \phi_2) = 1$ if $s \in \mathit{Sat}(\phi_2)$ and 0 otherwise;
- $\mathbb{P}^{C}_{\mathsf{G},s}(\phi_1\ \mathtt{U}^{\leqslant k+1}\ \phi_2)$ equals 1 if $s \in \mathit{Sat}(\phi_2)$, else equals 0 if $s \notin \mathit{Sat}(\phi_1)$ and otherwise equals $\mathit{val}(\mathsf{Z})$ where $\mathsf{Z} \in \mathbb{R}^{l\times m}$ is the matrix game with:

$$z_{i,j} = \textstyle\sum_{s' \in S} \delta(s,(a_i,b_j))(s') \cdot \mathbb{P}^{C}_{\mathsf{G},s'}(\phi_1\ \mathtt{U}^{\leqslant k}\ \phi_2)\,.$$

If $\psi = \phi_1\ \mathtt{U}\ \phi_2$, the probability values can be computed through *value iteration* [23], i.e. using the fact that $\langle \mathbb{P}^{C}_{\mathsf{G},s}(\phi_1\ \mathtt{U}^{\leqslant k}\ \phi_2)\rangle_{k\in\mathbb{N}}$ is a non-decreasing sequence converging to $\mathbb{P}^{C}_{\mathsf{G},s}(\phi_1\ \mathtt{U}\ \phi_2)$ and computing $\mathbb{P}^{C}_{\mathsf{G},s}(\phi_1\ \mathtt{U}^{\leqslant k}\ \phi_2)$ for sufficiently large k, i.e. terminating the computation when the difference between the values for k and $k{+}1$ are less than some threshold ε. Alternatively, *policy iteration* could be used, e.g., see [6].

Reward formulae. We next show how to compute the expected reward values $\mathbb{E}^{C}_{\mathsf{G},s}(r,\rho)$ for each state s and path formula ρ. If $\rho = \mathtt{I}^{=k}$, the values can be computed recursively as follows.

- $\mathbb{E}^{C}_{\mathsf{G},s}(r,\mathtt{I}^{=0}) = \mathit{val}(\mathsf{Z})$ where $\mathsf{Z} \in \mathbb{R}^{l\times m}$ is the matrix game with $z_{i,j}{=}r_S(s)$;
- $\mathbb{E}^{C}_{\mathsf{G},s}(r,\mathtt{I}^{=k+1})$ equals $\mathit{val}(\mathsf{Z})$ where $\mathsf{Z} \in \mathbb{R}^{l\times m}$ is the matrix game with:

$$z_{i,j} = \textstyle\sum_{s' \in S} \delta(s,(a_i,b_j))(s') \cdot \mathbb{E}^{C}_{\mathsf{G},s'}(r,\mathtt{I}^{=k})\,.$$

If $\rho = \mathtt{C}^{\leqslant k}$, then the values can be computed recursively as follows.

- $\mathbb{E}^{C}_{\mathsf{G},s}(r,\mathtt{C}^{\leqslant 0}) = 0$;
- $\mathbb{E}^{C}_{\mathsf{G},s}(r,\mathtt{C}^{\leqslant k+1})$ equals $\mathit{val}(\mathsf{Z})$ where $\mathsf{Z} \in \mathbb{R}^{l\times m}$ is the matrix game with:

$$z_{i,j} = r_A(s,(a_i,b_j)) + r_S(s) + \textstyle\sum_{s' \in S} \delta(s,(a_i,b_j))(s') \cdot \mathbb{E}^{C}_{\mathsf{G},s'}(r,\mathtt{C}^{\leqslant k})\,.$$

For the remaining reward formulae we restrict the use of negative rewards to ensure correctness. For total rewards ($\rho = \mathtt{C}$) we require that all negative rewards are associated with state-action pairs that reach an absorbing state (a state where all rewards are zero and which cannot be left) with probability 1. For reachability rewards ($\rho = \mathtt{F}^c\ \phi$) we require all negative rewards are associated with state-action pairs that reach a target or absorbing state with probability 1.

If $\rho = \mathtt{C}$, we first find the states for which the optimal expected reward values are infinite. Similarly to [9], we can use the qualitative algorithms of [1] to find these states as they can also be defined as those states for which the coalition C can ensure the probability of reaching states with positive reward infinitely often is greater than 0. After removing these states from G^C, the remaining values can be computed as the limit of the (non-decreasing sequence of) bounded cumulative reward values:

$$\mathbb{E}^C_{\mathsf{G},s}(r, \mathtt{C}) = \lim_{k\to\infty} \mathbb{E}^C_{\mathsf{G},s}(r, \mathtt{C}^{\leqslant k})\,.$$

Finally, if $\rho = \mathtt{F}^c\ \phi$, then we first make all states of G^C satisfying ϕ absorbing. Next, as in the case above, we find the states of G^C for which the optimal expected reward values are infinite and remove them from the game. The values for the remaining states can then be computed through value iteration where $\mathbb{E}^C_{\mathsf{G},s}(\mathtt{F}^c\ \phi) = \lim_{k\to\infty} \mathbb{E}^C_{\mathsf{G},s}(\mathtt{F}^c\ \phi)_k$, $\mathbb{E}^C_{\mathsf{G},s}(\mathtt{F}^c\ \phi)_0 = 1$ if $s \in \mathit{Sat}(\phi)$ and 0 otherwise and $\mathbb{E}^C_{\mathsf{G},s}(\mathtt{F}^c\ \phi)_{k+1}$ equals 1 if $s \in \mathit{Sat}(\phi_2)$ and otherwise equals $\mathit{val}(\mathsf{Z})$, and $\mathsf{Z} \in \mathbb{R}^{l\times m}$ is the matrix game with:

$$z_{i,j} = r_A(s,(a_i,b_j)) + r_S(s) + \textstyle\sum_{s'\in S} \delta(s,(a_i,b_j))(s') \cdot \mathbb{E}^C_{\mathsf{G},s'}(\mathtt{F}^c\ \phi)_k\,.$$

Strategy synthesis. In addition to verifying rPATL formulae, it is typically also very useful to perform *strategy synthesis*, i.e., to construct a witness to the satisfaction of a property. For example, in the case of the probabilistic property $\langle\!\langle C\rangle\!\rangle \mathtt{P}_{\sim q}[\,\psi\,]$, this means finding a strategy $\sigma_1^\star$ for the coalition C such that:

$$\mathbb{P}^{\sigma_1^\star,\sigma_2'}_{\mathsf{G}^C,s}(\psi) \sim q \quad \text{for all } \sigma_2' \in \Sigma^2_{\mathsf{G}^C}$$

and, in the case of a quantitative probabilistic property $\langle\!\langle C\rangle\!\rangle \mathtt{P}_{\max=?}[\,\psi\,]$, means finding an optimal strategy $\sigma_1^\star$ for the coalition C, i.e. a strategy such that:

$$\mathbb{P}^{\sigma_1^\star,\sigma_2'}_{\mathsf{G}^C,s}(\psi) \geqslant \mathbb{P}^C_{\mathsf{G},s}(\psi) = \sup_{\sigma_1\in\Sigma^1_{\mathsf{G}^C}} \inf_{\sigma_2\in\Sigma^2_{\mathsf{G}^C}} \mathbb{P}^{\sigma_1,\sigma_2}_{\mathsf{G}^C,s}(\psi) \quad \text{for all } \sigma_2' \in \Sigma^2_{\mathsf{G}^C}\,.$$

For unbounded properties, we can synthesise a strategy which is memoryless, but which needs randomisation; bounded properties require both finite-memory and randomisation. This differs from TSGs where deterministic strategies are sufficient in both cases. We can synthesise such strategies using the approach above for computing optimal values and keeping track of the optimal strategy of player 1 for the matrix game solved in each state. Since we use value iteration, only ε-optimal strategies can by synthesised for unbounded properties, where ε is the convergence criterion used when performing value iteration [24].

Correctness and complexity. We conclude this section with a discussion of correctness and complexity. The overall (recursive) approach and the reduction to solution of a two-player game is essentially the same as for TSGs [9], and therefore the same correctness arguments apply. The correctness of value iteration for unbounded properties follows from [23] and for bounded properties from Definition 5 and the solution of matrix games (see Section 2). Regarding complexity, due to the recursive nature of the algorithm, it is linear in the size of the formula ϕ, while in the worst case finding the optimal values of a 2-player CSG is PSPACE [7]. In practice, we use value iteration, which solves an LP problem of size $|A|$ for each state at each iteration, with the number of iterations depending on the convergence criterion. The efficiency in practice is reported in Section 6.

5 Implementation and Tool Support

We have implemented support for modelling and automated verification of CSGs as an extension of PRISM-games [16], which only handled TSGs. The tool, and the files for the case studies described in the next section, are available from [28].

Modelling. CSGs are specified using the same language as for TSGs, itself an extension of the original PRISM modelling language (Figure 2 shows an example). It allows multiple parallel components, called *modules*, operating both asynchronously and synchronously. Each module's state is defined by a number of finite-valued variables, and its behaviour by a set of probabilistic guarded commands $[a]\ g \to u$, comprising an action label a, guard g and probabilistic update u. If the guard (a predicate over the variables of all modules) is satisfied, then the module can (probabilistically) update its variables according to u. Modules interact by either reading the values of each other's variables, or synchronising (moving simultaneously) on commands labelled with the same action.

To specify a CSG, the model description must also define a list of players and the (disjoint) sets of actions each controls. We adopt the syntax from PRISM-games, but the semantics differs from TSGs. In a state of a CSG, each player chooses to perform one of the commands that is enabled (its guard is true) and is labelled by an action under its control (if no such command is enabled, the player idles, i.e. chooses $\bot$). Unlike standard PRISM models, the commands for all players then execute synchronously, despite being labelled with distinct actions. Currently, to remain consistent with PRISM's conventions, we require that each variable is updated by at most one player and each player's updates are independent of the other players' choices. This has not proven restrictive (see, e.g., the range of examples modelled in Section 6), but we plan to relax these constraints in the future.

Example 3. Figure 2 shows a model description for the rock-paper-scissors CSG of Example 1. Player 1 is represented by module *player1*, with variable *m1*, and its commands are labelled with the actions *r1*, *p1*, *s1*, *t1* (corresponding to r, p, s, t in Figure 1(a)). In this example, the updates are all non-probabilistic. Player 2 is identical in structure to player 1 and is defined using PRISM's module

```
csg

player player1 [r1], [p1], [s1], [t1] endplayer
player player2 [r2], [p2], [s2], [t2] endplayer

module player1
   m1 : [0..3];
   [r1] m1=0 → (m1'=1); // rock
   [p1] m1=0 → (m1'=2); // paper
   [s1] m1=0 → (m1'=3); // scissors
   [t1] m1>0 → (m1'=0); // restart
endmodule

// second player constructed through renaming
module player2 = player1[m1=m2, r1=r2, p1=p2, s1=s2, t1=t2] endmodule

label "win1" = (m1=1 & m2=3) | (m1=2 & m2=1) | (m1=3 & m2=2); // player 1 wins round
label "win2" = (m2=1 & m1=3) | (m2=2 & m1=1) | (m2=3 & m1=2); // player 2 wins round
label "draw" = (m2=1 & m1=1) | (m2=2 & m1=2) | (m2=3 & m1=3); // draw

rewards "utility" // utility for player 1
   [t1] (m1=1 & m2=3) | (m1=2 & m2=1) | (m1=3 & m2=2) : 1; // player 1 wins
   [t1] (m1=1 & m2=2) | (m1=2 & m2=3) | (m1=3 & m2=1) : -1; // player 2 wins
endrewards
```

Fig. 2: PRISM language specification of CSG from Example 1.

renaming feature. Labels (defining the atomic propositions from Figure 1(a)) and reward structures are also defined in the standard way for PRISM models.

Implementation. Our tool constructs a CSG from a given model specification and implements the rPATL model checking and strategy synthesis algorithms from Section 4. We adapt the existing modelling/property language parsers and various other pieces of basic model checking functionality from PRISM-games. We store and verify CSGs using an extension of PRISM's explicit-state (sparse matrix based) model checking engine, which is implemented in Java. A notable addition to this is the solution of values of matrix games, which is performed via linear programming (see Section 2) using the LPsolve library [17].

6 Case Studies and Experimental Results

To demonstrate the applicability of our techniques and tool, and to evaluate their performance, we now present results from a variety of case studies. This also illustrates the utility of CSGs over TSGs. As mentioned earlier, the tool and examples (models and properties) are available from [28].

Efficiency and scalability. We begin by presenting a selection of results regarding the performance of our implementation. The models on which these are based are described in more detail in subsequent sections. Table 1 shows results for a representative selection of models and rPATL properties, verified using a 2.8 GHz Intel X5660 with 32GB RAM. We give model statistics (number of players, states and transitions) and the times for model construction and verification (just the numerical part, i.e. value iteration, for the latter). Our tool is able to analyse models with up to approximately 3 million states. Models with

Case study & rPATL property [parameters]	Param. values	CSG statistics			Time (s)	
		Players	States	Transitions	Const.	Verif.
Future markets investors $\langle\langle i1 \rangle\rangle \mathtt{R}^{profit1}_{\mathtt{max}=?}[\,\mathtt{F}^{c}\ cashed\_in_1\,]$ [*months*]	1	5	61	92	0.1	0.1
	3	5	3,664	13,482	0.4	2.0
	5	5	18,671	82,494	1.1	16.1
	7	5	53,799	247,807	2.3	55.5
	9	5	116,838	561,538	4.7	145.4
User-centric networks $\langle\langle user \rangle\rangle \mathtt{R}^{unpaid}_{\mathtt{max}=?}[\,\mathtt{F}^{c}\ services{=}K\,]$ [td, K]	1,1	7	1,029	2,386	0.4	0.9
	1,3	7	18,218	50,181	1.9	17.5
	1,5	7	145,561	458,169	11.0	227.6
	1,7	7	755,531	2,651,829	61.6	1,611.0
	1,9	7	2,993,308	11,461,723	269.0	7,967.0
Intrusion detection system $\langle\langle policy \rangle\rangle \mathtt{R}^{damage}_{\mathtt{min}=?}[\,\mathtt{C}\,]$ [*rounds*]	25	4	581	1,616	0.2	2.9
	50	4	1,181	3,316	0.3	10.2
	100	4	2,381	6,716	0.3	37.1
	200	4	4,781	13,516	0.4	139.2
Jamming radio systems $\langle\langle user \rangle\rangle \mathtt{P}_{\mathtt{max}=?}[\,\mathtt{F}\ sent \geqslant slots/2\,]$ [*slots*]	10	3	7,921	1,038,384	3.8	13.6
	15	3	17,281	2,421,504	9.9	43.3
	20	3	30,241	4,380,624	13.3	95.7
	25	3	46,801	6,915,744	22.8	195.8

Table 1: Statistics for a representative set of CSG verification instances.

10,000 states can be built and verified in under a minute; the largest takes under 2.5 hours. The most significant cost in terms of time is the need to (repeatedly) solve a matrix game in each state, but verification times are not quite linear in the model size due to variations in the number of iterations needed.

Futures market investors. The first of our case studies is a futures market investor model [19], which represents the interactions between investors and a stock market. For the TSG model of [19], in successive months, a single investor chooses whether to invest, the market decides whether to bar the investor, and then the values of shares and a cap on values are updated probabilistically. We have built and analysed several CSGs variants of the model, analysing optimal strategies for investors under adversarial conditions. First, we made the investor and market take their decisions concurrently, and verified that this yielded no additional gain for the investor (see [28]). This is because the market and investor have the same information, and so the market knows when it is optimal for the investor to invest without needing to see its decision.

We next modelled two competing investors who simultaneously decide whether to invest (and, as above, the market simultaneously decides whether to bar each of them). If the two investors cash in their shares in the same month, then their profits are reduced. We also consider several distinct profit models: 'normal market', 'later cash-ins' and 'later cash-ins with fluctuation'. The first is from [19] and the latter two reward postponing cashing in shares (see [28] for details). The CSG has 5 players: one for each investor, one deciding on the barring of investors, one controlling share values and one updating the month. We study both the maximum profit of one investor and the maximum combined profit of both investors. For comparison, we also build a TSG model in which the investors first take turns to decide whether to invest (the ordering decided by the market) and then the market decides on whether to bar any of the investors.

Figure 3 shows the maximum expected value over a fixed number of months under the 'normal market' for both the profit of first investor and the combined

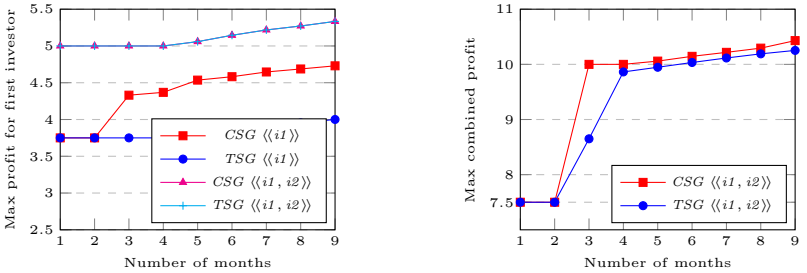


Fig. 3: Futures market investors: normal market.

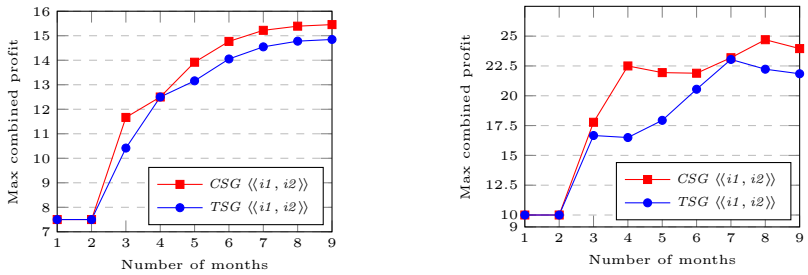


Fig. 4: Futures market: later cash-ins without (left) and with (right) fluctuations.

profit of the two investors. For the former, we show results for the first investor acting alone ($\langle\langle i1\rangle\rangle$) and when in a coalition with the second investor ($\langle\langle i1, i2\rangle\rangle$). We plot the corresponding results from the TSG model for comparison. Figure 4 shows the maximum expected combined profit for the other two profit models.

When investors cooperate to maximise the profit of the first, results for the CSG and TSG models coincide. This follows from the discussion above since all the second investor can do is make sure it does not invest at the same time as the first. For the remaining cases and given sufficient months, there is always a strategy in the concurrent setting that outperforms all turn-based strategies. The increase in profit for a single investor in the CSG model is due to the fact that, as the investors decisions are concurrent, the second cannot ensure it invests at the same time as the first, and hence decrease the profit of the first. In the case of combined profit, the difference arises because, although the market knows when it is optimal for one investor to invest, in the CSG model the market does not know which one will, and therefore may choose the wrong investor to bar.

We performed strategy synthesis to study the optimal actions of investors. By way of example, consider $\langle\langle i1\rangle\rangle \mathtt{R}^{profit1}_{\max=?}[\,\mathtt{F}^{c}\ cashed\_in_1\,]$ over three months and for a normal market (see Figure 3 left). The optimal TSG strategy for the first investor is to invest in the first month (which the market cannot bar) ensuring an expected profit of 3.75. The optimal (randomised) CSG strategy is to invest:

- in the first month with probability 0.494949;
- in the second month with probability 1, if the second investor has cashed in;
- in the second month with probability 0.964912, if the second investor did not cash in at the end of the first month and the shares went up;

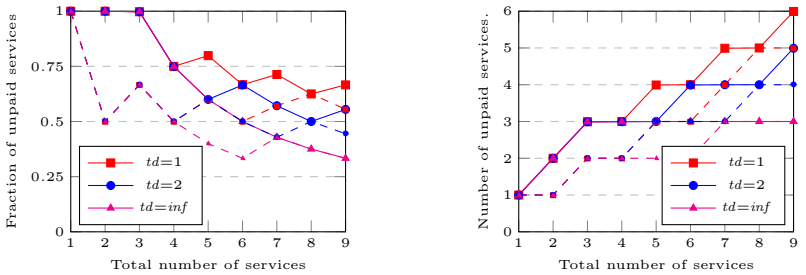


Fig. 5: User-centric network results (CSG/TSG values as solid/dashed lines).

– in the second month with probability 0.954023, if the second investor did not cash in at the end of the first month and the shares went down;
– in the third month with probability 1 (this is the last month to invest).

Following this strategy, the first investor ensures an expected profit of ∼4.33.

Trust models for user-centric networks. Trust models for user-centric networks were analysed previously using TSGs in [15]. The analysis considered the impact of different parameters on the effectiveness of cooperation mechanisms between service providers. The providers share information on the measure of *trust* for users in a *reputation*-based setting. Each measure of trust is based on the service's previous interactions with the user (which services they paid for). In the original TSG model, a single user can either make a request to one of three service providers or buy the service directly by paying maximum price. If the user makes a request to a service provider, then the provider decides to accept or deny the request based on the user's trust measure. If the request was accepted, the provider would next decide on the price again based on the trust measure, and the user would then decide whether to pay for the service and finally the provider would update its trust measure based on whether there was a payment. This sequence of steps would have to take place before any other interactions occurred between the user and other providers.

Here we consider CSG models allowing the user to make requests and pay different service providers simultaneously and for the different providers to execute requests concurrently. There are 7 players: one for the user's interaction with each service provider, one for the user buying services directly and one for each of the 3 service providers. Three trust models were considered. In the first, the trust level was decremented by 1 (td=1) when the user does not pay, decremented by 2 in the second (td=2) and reset to 0 in the third (td=inf).

Figure 5 presents results for the maximum fraction and number of unpaid services the user can ensure for each trust model. The results for the orignal TSG model are included as dashed lines. The results demonstrate that the user can take advantage of the fact that in the CSG model it can request multiple services at the same time, and obtain more services without paying before the different providers get a chance to inform each other about non-payment. In addition, the results show that having a more severe penalty on the trust measure for non-payment decreases the unpaid services the user can obtain.

Intrusion detection policies. In [26], CSGs are used to model the interaction between an intrusion detection policy and attacker. The policy has a number of libraries it can use to detect attacks and the attacker has a number of different attacks which can incur different levels of damage if not detected. Furthermore, each library can only detect certain attacks. In the model, in each round the policy chooses a library to deploy and the attacker chooses an attack. A reward structure is specified representing the level of damage when an attack is not detected. The goal is to find optimal intrusion detection policies which correspond to finding a strategy for the policy that minimises damage. We have constructed CSG models with 4 players (representing the policy, attacker, system and time) for the two scenarios outlined in [26]. We have synthesised optimal policies which ensure the minimum cumulative damage over a fixed number of rounds and damage in a specific round. Here concurrency is required for the game to be meaningful, otherwise it is easy for the player whose turn follows the other player's to 'win'. For example, if the attacker knows what library is being deployed, then it can simply choose an attack the library cannot detect.

Jamming multi-channel radio systems. A CSG model for jamming multi-channel cognitive radio systems is presented in [27]. The system consists of a number of channels which can be an occupied or idle state. The state of each channel remains fixed within a time slot and between slots is Markovian (i.e. the state changes randomly based only on the state of the channel in the previous slot). A secondary user has a subset of available channels and at each time-slot must decide which to use. There is a single attacker which again has a subset of available channels and at each time slot decides to send a jamming signal over one of them. The CSG has 3 players representing the secondary user, attacker and environment. We synthesise strategies for the secondary user which maximise the probability of ensuring at least half the messages are sent correctly. Again concurrency is required as otherwise, e.g., the attacker can observe the user and then jam the chosen channel.

7 Conclusion

We have designed and implemented an approach for the automatic verification of CSGs. We have extended the semantics of the temporal logic rPATL to CSGs and presented a new modelling approach based on the PRISM language to specify such games. We have proposed and implemented algorithms for property verification and strategy synthesis as an extension of the PRISM-games model checker. Finally, we have evaluated the approach on a range of case studies.

There are a number of directions for future work. First, we plan to consider additional properties (e.g. Nash equilibria and multi-objective queries). We are also working on extending the implementation to consider alternative solution methods (e.g. policy iteration and using CPLEX [12] to solve matrix games) and a symbolic (binary decision diagram based) implementation. Lastly, we are considering extending the approach to partially observable strategies.

References

1. de Alfaro, L., Henzinger, T.: Concurrent omega-regular games. In: LICS'00 (2000)
2. de Alfaro, L., Henzinger, T., Kupferman, O.: Concurrent reachability games. TCS 386(3) (2007)
3. de Alfaro, L., Majumdar, R.: Quantitative solution of omega-regular games. JCSS 68(2) (2004)
4. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49(5) (2002)
5. Brenguier, R.: PRALINE: A tool for computing Nash equilibria in concurrent games. In: Proc. CAV'13. LNCS 8044, Springer (2013)
6. Chatterjee, K., de Alfaro, L., Henzinger, T.: Strategy improvement for concurrent reachability and turn-based stochastic safety games. JCSS 79(5) (2013)
7. Chatterjee, K., Henzinger, T.: A survey of stochastic ω-regular games. JCSS 78(2) (Mar 2012)
8. Chatterjee, K., Henzinger, T., Jobstmann, B., Radhakrishna, A.: Gist: A solver for probabilistic games. In: Proc. CAV'10. LNCS 6174, Springer (2010)
9. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. FMSD 43(1) (2013)
10. Cheng, C., Knoll, A., Luttenberger, M., Buckl, C.: GAVS+: An open platform for the research of algorithmic game solving. In: Proc. TACAS'11. LNCS 6605, Springer (2011)
11. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A storm is coming: A modern probabilistic model checker. In: Proc. CAV'17. LNCS 10427, Springer (2017)
12. ILOG CPLEX, ibm.com/products/ilog-cplex-optimization-studio
13. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. Springer (1976)
14. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc CAV'11. LNCS 6806, Springer (2011)
15. Kwiatkowska, M., Parker, D., Simaitis, A.: Strategic analysis of trust models for user-centric networks. In: Proc. SR'13. EPTCS 112 (2013)
16. Kwiatkowska, M., Parker, D., Wiltsche, C.: PRISM-games: Verification and strategy synthesis for stochastic multi-player games with multiple objectives. STTT 20(2) (2018)
17. LPSolve (version 5.5), lpsolve.sourceforge.net/5.5/
18. Martin, D.: The determinacy of Blackwell games. J. Symb. Log 63(4) (1998)
19. McIver, A., Morgan, C.: Results on the quantitative mu-calculus qMu. ACM Trans. Computational Logic 8(1) (2007)
20. Nash, J.: Equilibrium points in n-person games. Proc. Natl. Acad. Sci 36 (1950)
21. von Neumann, J.: Zur theorie der gesellschaftsspiele. Mathematische Annalen 100 (1928)
22. von Neumann, J., Morgenstern, O., Kuhn, H., Rubinstein, A.: Theory of Games and Economic Behavior. Princeton University Press (1944)
23. Raghavan, T., Filar, J.: Algorithms for stochastic games — a survey. Zeitschrift für Operations Research 35(6), 437–472 (Nov 1991)
24. Svorenova, M., Kwiatkowska, M.: Quantitative verification and strategy synthesis for stochastic games. European Journal of Control 30 (2016)
25. Toumi, A., Gutierrez, J., Wooldridge, M.: A tool for the automated verification of Nash equilibria in concurrent games. In: Proc. ICTAC'15. Springer (2015)
26. Zhu, Q., Baar, T.: Dynamic policy-based IDS configuration. In: CDC'09 (2009)
27. Zhu, Q., Li, H., Han, Z., Basar, T.: A stochastic game model for jamming in multi-channel cognitive radio systems. In: Proc. ICC'10. IEEE (2010)
28. Supporting material, www.prismmodelchecker.org/subm/qest18/