Verification of Multiplayer Stochastic Games via Abstract Dependency Graphs

Søren Enevoldsen, Mathias Claus Jensen, Kim Guldstrand Larsen, Anders Mariegaard, and Jiří Srba

Department of Computer Science, Aalborg University
Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark
{senevoldsen,mcje,kgl,am,srba}@cs.aau.dk

Abstract. We design and implement an efficient model checking algorithm for alternating-time temporal logic (ATL) on turn-based multiplayer stochastic games with weighted transitions. This logic allows us to query about the existence of multiplayer strategies that aim to maximize the probability of game runs satisfying resource-bounded next and until logical operators, while requiring that the accumulated weight along the successful runs does not exceed a given upper bound. Our method relies on a recently introduced formalism of abstract dependency graphs (ADG) and we provide an efficient reduction of our model checking problem to finding the minimum fixed-point assignment on an ADG over the domain of unit intervals extended with certain-zero optimization. As the fixed-point computation on ADGs is performed in an on-the-fly manner without the need of a priori generating the whole graph, we achieve a performance that is comparable with state-of-the-art model checker PRISM-games for finding the exact solutions and sometimes an order of magnitude faster for queries that ask about approximate probability bounds. We document this on a series of scalable experiments from the PRISM-games benchmark that we annotate with weight information.

1 Introduction

Advances in model checking over the last decades allow us to verify larger systems using less resources. More recently, addition of quantitative aspects to model checking techniques became an important research topic. In order to model real-world applications, modelling formalisms must reflect both *probabilistic choices* [5] that model the uncertainties in system behaviour and at the same time be able to reason about quantitative aspects such as *cost* [22]. Moreover, in order to take into account the unpredictable environment, we need to verify that the desirable properties hold for all possible environmental behaviours. These aspects are usually modelled as *games*—in our case multiplayer games [39] where the players form coalitions in order to enforce a given property.

In order to reason about the probabilistic, cost and game aspects, we study the model of turn-based multiplayer stochastic games [40] where transitions contain multidimensional cost (weight) vectors, representing different cost quanti-

ties. Multidimensional verification is necessary in applications where the system must respect bounds on several dependent quantities simultaneously (see e.g [26,12,27]), such as consumption of energy and the discrete progression of time. We assume that any play of a game eventually accumulates some weight, which is natural for many models that include quantities such as time and energy, as executing an infinite number of actions without progressing time or consuming energy, is in many cases unrealistic. Our model can be seen as a weight extension of PRISM-games [32], where we consider properties formulated in an extension of alternating-time temporal logic (ATL) [1] that contains operators that specify existence of strategies for player coalitions ensuring cost- and probability bounded next or until properties. Hence we can ask questions like "is the probability that player 1 and 3 can form a coalition such that they enforce that a certain state is reachable within a total cost of c_1 time units and c_2 units of energy, greater than 0.8"? . We can thus reason about strategies that enforce strict bounds on multiple accumulating quantities *simultaneously*. This has many practical applications for systems that e.g have to complete a number of tasks within a given time-limit, but must at the same time also stay within an energy budget, no matter how the environment behaves.

Our verification approach is based on a novel reduction to the problem of finding fixed points on *abstract dependency graphs* (ADG) [25,23], a recently introduced formalism that extends classical dependency graphs by Liu and Smolka [35]. Dependency graphs allow us to assign Boolean values to nodes in the graph, whereas ADGs assign to nodes values from a more abstract domain. In our case, we use the domain of the unit interval, representing probabilities, extended with a special value called "certain-zero" [20] that allows for an early termination of the on-the-fly computation of the fixed point on the ADG. We formally prove the correctness of our encoding and provide an efficient implementation that allows us to take as input the models described in PRISM-games and perform model checking in an on-the-fly manner. On three different PRISM-games case studies (annotated with the cost information), we demonstrate that our implementation is performance-wise comparable to the state-of-the-art model checker PRISM-games on queries that include exact probability bounds. However, once we lower the probability threshold from the exact probability bound, our on-the-fly algorithm demonstrates the potential of significantly outperforming PRISM-games.

Related Work Since the introduction of stochastic games in the seminal work by Shapley [39], a number of variations and extensions of the classical formalism have been studied by the verification community. From a theoretical perspective, Condon [18,19] studies the complexity and algorithms for (simple) stochastic two-player games where the objective is to determine the winning probability for a given player. More recently, [4,10] consider controller synthesis for turn-based stochastic two-player games with PCTL winning objectives. Compared to our work, these papers consider controller synthesis instead of model-checking, and do not consider quantitative games and offer no implementation.

For *quantitative* verification of turn-based stochastic multiplayer games, [13] presents the logic rPATL (Probabilistic Alternating-Time Temporal Logic with

Rewards) that naturally extends the logic Probabilistic Alternating-Time Temporal Logic [16] (PATL) with reward-operators. PATL is itself a probabilistic extension of ATL. A similar logic is introduced in [36], interpreted on concurrent games. The logic rPATL allows one to state that a *coalition* of players has a strategy such that either the probability of an event happening or an expected reward measure, is within a given threshold. Verifying rPATL properties on stochastic multiplayer games has been implemented in PRISM-games [32]. PRISM-games supports analysis of various types of games, verification of multi-objective properties [14] and has been applied to several case-studies (see e.g [15,13]). Compared to our approach, PRISM-games does not directly support multidimensional reward-bounded properties and the current implementation offers no on-the-fly verification techniques that we demonstrate can yield a considerable speedup. Recently, a number of papers [6,38,28] have improved *value iteration*, the underlying technique of PRISM-GAMES, to deal with inaccuracies in the computed results stemming from certain termination criteria based on lower bound approximations. The approach has been applied to simple stochastic games [31,3] but has yet to be incorporated into PRISM-GAMES. Although our approach also computes lower bounds, we prove that we always terminate and compute the exact answer, relying on the fact that any formula is weight-constrained and any path of any game eventually accumulates weight. Another approach to computing measures on probabilistic models with multi-dimensional rewards and non-determinism (MDPs) is presented in [27]. A performance comparison is left for the future work.

Lastly, our work is a continuation of the work done in [30], where a special-purpose algorithm is developed for PCTL model-checking on models with multi-dimensional weights.We lift the approach to games by showing how to formally treat the game features in ADGs and we consider a new set of domain values that treat the probabilities symbolically while the weights are encoded explicitly; our novel encoding outperforms the pure symbolic implementation provided in [30] by an order of magnitude. Finally, our approach is more generic as it relies on the notion of ADGs and variations of the logic and/or the model can often be dealt with by minor modifications of the ADG construction, without the need of changing the underlying fixed-point algorithm. A related abstract approach is presented in [7,8], for solving systems of fixed-point equations over (continuous) lattices via a game-theoretic approach. An example application is (lattice-valued) μ-calculus model-checking [8] that deals with systems of fixed-point equations over infinite lattices (e.g the reals), which in turn can be applied to model-checking probabilistic CTL or probabilistic μ-calculi.

2 Turn-Based Stochastic Games

Before introducing turn-based stochastic games, we present some preliminaries. For any set X, X^n is the set of all n-dimensional vectors with elements from X and x^n denotes the n-dimensional vector where $x \in X$ is at all coordinates. Thus, $\mathbb{N}^n$ is the set of all n-dimensional vectors of natural numbers and 0^n is the

(a) Turn-based stochastic game

(b) Markov reward model

Fig. 1: Two simple models

0-vector. We assume a fixed dimensionality $n > 0$ and any vector is written in boldface e.g. $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ are vectors. For any such two vectors, we let $\mathbf{x} \geq \mathbf{y}$ if and only if $x_i \geq y_i$ for all $1 \leq i \leq n$. For any countable non-empty set X, we let $\mathcal{D}(X) = \{\mu \colon X \to [0,1] \mid \sum_{x \in X} \mu(x) = 1\}$ denote the set of probability distribution on X. For any distribution $\mu \in \mathcal{D}(X)$, the *support* of μ is defined as $\texttt{support}(\mu) = \{x \in X \mid \mu(x) > 0\}$. By $\mathcal{D}_{\mathsf{fin}}(X) \subseteq \mathcal{D}(X)$ we denote the set of all distributions on X with finite support. For any two sets X and Y we denote by $f \colon X \rightharpoonup Y$ that f is a partial function from domain $dom(f) = X$ to range $ran(f) = Y$. For a set X, let X^* be the set of all finite strings over X and for any string $w = a_1 a_2 a_3 \cdots a_n \in X^*$, let $|w| = n$ denote the length of w and for all $1 \leq i \leq |w|$, let $w[i] = a_i$ be the i'th symbol of w. The empty string is denoted by ε.

2.1 Definition of Stochastic Games

We now present turn-based stochastic multiplayer games [39], where the states are partitioned into a number of sets, each set owned by a player of the game. The game begins in a state owned by one of the players and proceeds in turns, by letting the owner of the current state play one of the available actions after which the game then transitions to the next state by a probabilistic choice. Each such transition has an associated cost vector, that can naturally be interpreted as the cost of the transition. Hence, given a *strategy* for each player in the game, any non-determinism is resolved and the induced model is what is known as a Markov reward model with impulse rewards [2,17]. It is a folklore result that deterministic strategies are sufficient (see e.g. [37]). We assume a fixed finite set of atomic propositions AP.

Definition 1. *A Markov reward model (MRM) is a tuple $\mathcal{M} = (M, \to, \ell)$ where M is a finite set of states, $\to \colon M \to \mathcal{D}_{\mathsf{fin}}(\mathbb{N}^n \times M)$ is the transition function and $\ell \colon M \to 2^{\mathsf{AP}}$ is the labelling function.*

For any state $m \in M$, the probability of transitioning to another state m' with cost $\mathbf{w}$ is given by $\to(m)(\mathbf{w}, m')$. A $\mathbf{w}$-successor of a state m is any state m' such that $\to(m)(\mathbf{w}, m') > 0$. A path is an infinite sequence of transitions $\pi = (m_1, \mathbf{w_1}, m_2), (m_2, \mathbf{w_2}, m_3) \cdots$ where s_{i+1} is a $\mathbf{w_i}$-successor of s_i for all $i \geq 1$.

We let $\mathsf{Paths}(m)$ denote the set of all paths starting in m and for any path $\pi \in \mathsf{Paths}(m)$ we let $\pi[i]$ denote the i'th state of π and by π_n denote the finite prefix of π ending in state $\pi[n]$. We let $\mathcal{W}(\pi)(j) = \sum_{i=1}^{j-1} \mathbf{w_i}$ denote the accumulated cost up until the state $\pi[j]$. Finally, we let $\mathsf{Paths}(M)$ be the set of all paths of M. An example of an MRM can be seen in Figure 1b.

In order to measure events of any MRM $\mathcal{M} = (M, \to, \ell)$, we introduce the classical cylinder set construction from [5, Chapter 10]. For any finite sequence $w = (m_1, \mathbf{w_1}, m_2), (m_2, \mathbf{w_2}, m_3) \cdots (m_{n-1}, \mathbf{w_{n-1}}, m_n)$, the cylinder set of w, $C(w)$ is the set of all paths having w as a prefix, i.e., $C(w) = \{\pi \in \mathsf{Paths}(M) \mid \pi_n = w\}$ and the measure associated to the cylinder of w is given by $\mathbb{P}_M(C(w)) = \prod_{i=1}^{n-1} \to(m_i)(\mathbf{w_i}, m_{i+1})$. We can now define the probability space $(M^\omega, \Sigma, \mathbb{P}_M)$ where Σ is the smallest σ-algebra that contains the cylinder sets of all finite alternating sequences of states and costs.

We now lift MRMs to stochastic games. Let Act be a fixed finite set of actions.

Definition 2. *A* turn-based stochastic multiplayer game *is a structure* $\mathcal{G} = (\Pi, M, \{M_i\}_{i \in \Pi}, \to, \ell)$ *where* Π *is a finite set of players,* M *is a finite set of state,* $\{M_i\}_{i \in \Pi}$ *is a partition of* M *such that for any* $i \in \Pi$, M_i *is a finite set of states controlled by player* i, $\to \colon M \times \mathsf{Act} \rightharpoonup \mathcal{D}_{\mathsf{fin}}(\mathbb{N}^n \times M)$ *is the finite (partial) transition function and* $\ell \colon S \to 2^{\mathsf{AP}}$ *is a labelling function.*

For any state $m \in M$ we let $\mathsf{Act}(m) = \{\alpha \in \mathsf{Act} \mid (m, \alpha) \in dom(\to)\}$ denote the set of *enabled* actions in state m and assume any game to be *non-blocking* by requiring all states to have at least one enabled action, i.e $\mathsf{Act}(m) \neq \emptyset$. An α-successor of a state m is any state m' such that the probability of transitioning from m by playing the α action is strictly positive for some cost vector $\mathbf{w} \in \mathbb{N}^n$, i.e $\to(m, \alpha)(\mathbf{w}, m') > 0$. We let $\mathtt{succ}(m)_\alpha$ be the set of all α-successors of m. A path is an infinite sequence of transitions $\pi = (m_1, \alpha_1, \mathbf{w_1}, m_2), (m_2, \alpha_2, \mathbf{w_2}, m_3), \ldots$ where m_{i+1} is an α_i-successor of m_i with cost vector $\mathbf{w_i}$ for all $i \geq 1$. For any action $\alpha \in \mathsf{Act}(m)$ we let $\mathbf{k} = \min\{\mathbf{w} \mid \to(m, \alpha)(\mathbf{w}, m') > 0\}$ be the smallest possible transition cost when playing action α in m and say that α is $\mathbf{k}'$-enabled in m whenever $\mathbf{k}' \geq \mathbf{k}$ with $\mathsf{Act}_{\mathbf{k}'}(m) \subseteq \mathsf{Act}(m)$ being the set of all $\mathbf{k}'$-enabled actions in m. Thus, the set $\mathsf{Act}_{\mathbf{k}'}(m)$ contains the actions available to the player owning state m, if only transitions with a cost at most $\mathbf{k}'$ are permitted. We extend the path notation introduced for MRMs by letting Paths_i^* be the set of all finite paths that end in a state owned by player $i \in \Pi$ and for any such finite path $\pi \in \mathsf{Paths}_i^*$, the last state is given by $\mathsf{last}(\pi)$.

Remark 1. Notice that if $|\Pi| = 1$, the resulting model is a Markov decision process (MDP) [37] with impulse rewards and if furthermore $|\mathsf{Act}| = 1$, the model is an MRM. Hence, turn-based stochastic multiplayer games subsume both MDPs and MRMs.

In the rest of the paper, we restrict the class of games, by assuming that the accumulated cost of any loop of any game is of strictly positive magnitude. Formally, for any state $m \in M$, it is the case that for all paths $\pi \in \mathsf{Paths}(m)$ such that $\pi[j] = m$ for some $j \in \mathbb{N}$ (a loop), we have that $\mathcal{W}(\pi)(j) \neq 0^n$.

Example 1. Figure 1a depicts a simple turn-based stochastic game $\mathcal{G}$ with two players $\Pi = \{\bigcirc, \diamondsuit\}$. The states depicted as circles, m_1 and m_3 belong to player $\bigcirc$ while the state m_3 belongs to player $\diamondsuit$. The transition function is depicted by edges labelled by a given enabled action, followed by the cost of the transition and probabilities to successor states. The labelling of each state is given next to the state. In case the probability distribution assigns probability 1 to a single state, there is no branching and we simply label the edge with the action, probability 1 and the associated weight.

Starting from the state m_1, player $\bigcirc$ is in control and may choose either of the actions β and α. For β, there is a small probability, $\frac{1}{10}$, of transitioning to state m_3 whereas for action α, the game transitions to m_2 with probability $\frac{1}{2}$. In m_2, player $\diamondsuit$ may choose to let the game stay in state m_2 by the self-loop, or decide to transition to m_3.

If the two players are considered opponents and the goal of player $\bigcirc$ is to maximize the probability of reaching a state labelled b (m_3) within a given bound on the accumulated cost of reaching b, the only safe option is to always choose the action β in state m_1 as player $\diamondsuit$ can force the game to stay in state m_2 if it is ever reached. On the other hand, if the two players work together, player $\diamondsuit$ always plays the action β in m_2 to ensure that state m_3 is reached.

2.2 Strategies

As indicated by Example 1, any game unfolds by applying concrete *strategies* for each player, specifying which action to play in a given state. We now formally define strategies by first fixing a game $\mathcal{G} = (\Pi, M, \{M_i\}_{i\in\Pi}, \to, \ell)$. Given a player, $i \in \Pi$, a (history-dependent deterministic) strategy for player i in $\mathcal{G}$ is a function $\sigma : \mathsf{Paths}_i^* \to \mathsf{Act}$, that associates an action with each finite path ending in a state owned by player i. Thus, a strategy prescribes which action a player should play in a given state, given the full history of the game. For a strategy to be *sound*, only actions enabled in the given state must be played. Formally, a strategy σ for player i is sound if for any finite path $\pi \in \mathsf{Paths}_i^*$ with $\mathsf{last}(\pi) = m_i \in M_i$, it holds that $\sigma(\pi) \in \mathsf{Act}(m_i)$. We let $\mathfrak{S}_i$ denote the set of all sound strategies for player i in $\mathcal{G}$.

Remark 2. If $\sigma(\pi_1) = \sigma(\pi_2)$ for all $\pi_1, \pi_2 \in \mathsf{Paths}_i^*$ with $\mathsf{last}(\pi_1) = \mathsf{last}(\pi_2)$, we say that σ is a *memoryless strategy* for player i, as the action prescribed depends only on the last state of the game.

Strategies naturally extend to sets of players by considering what is commonly known as a *coalition* of players. A *coalition strategy* for any coalition $C \subseteq \Pi$ in $\mathcal{G}$, is a set of sound strategies, $\{\sigma_i\}_{i\in C}$, such that $\sigma_i \in \mathfrak{S}_i$ for all $i \in C$. We let $\mathfrak{S}_C$ denote the set of all coalition strategies for the coalition C, use σ_C to range over elements of $\mathfrak{S}_C$ and let $\overline{C} = \Pi \setminus C$ be the coalition containing the players in the complement of C. Given a state $m \in M$, coalition strategies σ_C and $\sigma_{\overline{C}}$, a unique MRM is induced from $\mathcal{G}$ by resolving the non-deterministic choices as prescribed by σ_C and $\sigma_{\overline{C}}$. We let $\mathbb{P}_{\mathcal{G}}^{\sigma_C,\sigma_{\overline{C}}}$ denote the probability measure on the induced MRM.

Example 2. Consider again the game from Figure 1a and the memoryless strategies $\sigma_{\bigcirc}^{\alpha}$ and $\sigma_{\diamond}^{\beta}$, respectively defined for any $\pi_{\bigcirc} \in \mathsf{Paths}_{\bigcirc}^{*}$ and $\pi_{\diamond} \in \mathsf{Paths}_{\diamond}^{*}$ as $\sigma_{\bigcirc}^{\alpha}(\pi_{\bigcirc}) = \alpha$ and $\sigma_{\diamond}^{\beta}(\pi_{\diamond}) = \beta$. The induced MRM is the one depicted in Figure 1b.

3 Probabilistic Weighted ATL

As a specification language, we employ an extension of Alternating-time Temporal Logic (ATL [1]) to reason about whether or not a given *coalition* of players can together enforce the game to enjoy a given property, regardless of the strategy of the remaining players of the game. Hence, a witness of satisfaction is a coalition-strategy. Our logic is syntactically similar to probabilistic resource-bounded ATL proposed by Nguyen and Rakib [36], but interpreted on turn-based games instead of concurrent games. It is also similar to rPATL [13] employed by PRISM-games, except that we do no support expected reward measures but we allow instead for multi-cost bounded path formulae. We restrict negation to atomic propositions and therefore include conjunction and disjunction explicitly.

Definition 3 (Syntax). *The set of PWATL formulae is given by the grammar:*

$$\phi ::= \mathsf{a} \mid \neg \mathsf{a} \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle\!\langle C \rangle\!\rangle_{\triangleright \lambda}[\psi] \qquad \text{(State Formulae)}$$

$$\psi ::= X_{\leq \mathbf{k}}\, \phi \mid \phi\, U_{\leq \mathbf{k}}\, \phi \qquad \text{(Path Formulae)}$$

where $\mathsf{a} \in \mathsf{AP}$, $C \subseteq \Pi$, $\lambda \in [0,1]$, $\mathbf{k} \in \mathbb{N}^n$ *and* $\triangleright = \{>, \geq\}$.

The set of PWATL state-formulae is denoted by $\mathcal{L}_{\mathsf{ATL}}$. A formula $\langle\!\langle C \rangle\!\rangle_{\triangleright \lambda}[\psi] \in \mathcal{L}_{\mathsf{ATL}}$ is satisfied by a state $m \in M$ of a game $\mathcal{G} = (\Pi, M, \{M_i\}_{i \in \Pi}, \rightarrow, \ell)$, if there exists a coalition strategy σ_C for the players in $C \subseteq \Pi$ such that, no matter which coalition strategy $\sigma_{\overline{C}}$ is assigned to the remaining players in $\overline{C}$, measuring paths that satisfy ψ in the MRM induces from $\mathcal{G}$ by σ_C and $\sigma_{\overline{C}}$, yields a probability p such that $p \triangleright \lambda$.

Definition 4 (Semantics). *For a game* $\mathcal{G} = (\Pi, M, \{M_i\}_{i \in \Pi}, \rightarrow, \ell)$, *state* $m \in M$, *and path* $\pi \in \mathsf{Paths}$, *PWATL satisfiability is defined inductively:*

$$\begin{aligned}
&\mathcal{G}, m \models \mathsf{a} && \textit{iff} \quad \mathsf{a} \in \ell(m)\\
&\mathcal{G}, m \models \neg \mathsf{a} && \textit{iff} \quad \mathsf{a} \notin \ell(m)\\
&\mathcal{G}, m \models \phi_1 \wedge \phi_2 && \textit{iff} \quad \mathcal{G}, m \models \phi_1 \textit{ and } \mathcal{G}, m \models \phi_2\\
&\mathcal{G}, m \models \phi_1 \vee \phi_2 && \textit{iff} \quad \mathcal{G}, m \models \phi_1 \textit{ or } \mathcal{G}, m \models \phi_2\\
&\mathcal{G}, m \models \langle\!\langle C \rangle\!\rangle_{\triangleright \lambda}[\psi] && \textit{iff} \quad \exists \sigma_C \in \mathfrak{S}_C . \forall \sigma_{\overline{C}} \in \mathfrak{S}_{\overline{C}} .\\
& && \qquad \mathbb{P}_{\mathcal{G}}^{\sigma_C, \sigma_{\overline{C}}}(\{\pi \in \mathsf{Paths}(m) \mid \mathcal{G}, \pi \models \psi\}) \triangleright \lambda\\
&\mathcal{G}, \pi \models \phi_1 U_{\leq \mathbf{k}} \phi_2 && \textit{iff} \quad \exists j \in \mathbb{N} . \mathcal{G}, \pi[j] \models \phi_2, \mathcal{W}(\pi)(j) \leq \mathbf{k}\\
& && \qquad \textit{and } \mathcal{G}, \pi[i] \models \phi_1 \textit{ for all } i < j\\
&\mathcal{G}, \pi \models X_{\leq \mathbf{k}} \phi && \textit{iff} \quad \mathcal{G}, \pi[2] \models \phi \textit{ and } \mathcal{W}(\pi)(1) \leq \mathbf{k}
\end{aligned}$$

Example 3. Consider once again the game in Figure 1a and the formula $\phi = \langle\!\langle C \rangle\!\rangle_{>\frac{1}{2}}[\mathsf{a}\, U_{\leq 8} \mathsf{b}]$ with $C = \{\bigcirc, \diamondsuit\}$ By the memoryless strategies from Example 2,

$$\mathbb{P}_{\mathcal{G}}^{\sigma_C, \emptyset}(\{\pi \in \mathsf{Paths}(m_1) \mid \mathcal{G}, \pi \models \mathsf{a}\, U_{\leq 8} \mathsf{b}\}) = \frac{1}{2}$$

where $\sigma_C = \{\sigma_{\bigcirc}^{\alpha}, \sigma_{\diamondsuit}^{\beta}\}$. This is easily verified by inspecting the induced MRM in Figure 1b. Hence, the two memoryless strategies do not prove $\mathcal{G}, m_1 \models \phi$.

To construct a strategy for $\mathcal{G}, m_1 \models \phi$, we modify the player $\bigcirc$ strategy. Instead of always playing action α, the action will depend on the accumulated cost of the game history: for any finite path $\pi_{\bigcirc} \in \mathsf{Paths}_{\bigcirc}^{*}$ of length at least j,

$$\sigma_{\bigcirc}^{*}(\pi_{\bigcirc}) = \begin{cases} \beta & \text{if } \mathcal{W}(\pi_{\bigcirc})(j) \leq 4 \\ \alpha & \text{otherwise} \end{cases} .$$

4 Model Checking Through Dependency Graphs

In this section we demonstrate how the PWATL model-checking problem for turn-based stochastic multiplayer games can be reduced to computing fixed points on so-called *abstract dependency graphs* [25]. For a model-checking problem $\mathcal{G}, m \models \phi$, the corresponding abstract dependency graph represents the decomposition of the problem into sub-problems (*dependencies*) given by the inductive definition of PWATL semantics.

4.1 Abstract Dependency Graphs

An abstract dependency graph [25] is a (directed) graph consisting of a collection of vertices V, together with a function that to each $v \in V$ assigns a set of vertices being the *dependencies* of v and a function for computing the value of v, given the value of all its dependencies. The vertex values are drawn from a triple $\mathfrak{D} = (D, \sqsubseteq, \bot)$ where $(D, \sqsubseteq)$ is a partial order, $\bot \in D$ the least element of D and $\sqsubseteq$ must satisfy the *ascending chain condition*: for any infinite chain $d^1 \sqsubseteq d^2 \sqsubseteq d^3 \ldots$ of elements $d^i \in \mathfrak{D}$, there exists an integer k such that $d^k = d^{k+j}$ for all $j > 0$. This kind of ordering is referred to in [25] as a Noetherian ordering relation with least element (NOR). For any NOR we assume the elements are finitely representable, meaning that elements can be represented by finite strings.

For the computation of the value of each vertex we consider the application of *monotone* functions to the values of all its dependencies. Formally, for any $n \in \mathbb{N}$, $\mathcal{F}(\mathfrak{D}, n)$ on a NOR $(\mathfrak{D}, \sqsubseteq, \bot)$ is the set of all monotone functions $f : \mathfrak{D}^n \to \mathfrak{D}$ of arity n, where f is monotone if $d_i \sqsubseteq d_i'$ for all i, $1 \leq i \leq n$, implies $f(d_1, \ldots, d_n) \sqsubseteq f(d_1', \ldots, d_n')$ for any $d_1, \ldots, d_n, d_1', \ldots d_n' \in D$, and we let $\mathcal{F}(\mathfrak{D}) = \bigcup_{n \geq 0} \mathcal{F}(\mathfrak{D}, n)$ be the collection of all such functions. We assume all functions $f \in \mathcal{F}(\mathfrak{D}, n)$ for any $n \in \mathbb{N}$ to be *effectively computable*, meaning that for any $f \in \mathcal{F}(\mathfrak{D}, n)$ and $d_1, \ldots, d_n \in D$, there exists an algorithm that terminates and computes the finite representation of $f(d_1, \ldots, d_n) \in D$.

We are now ready to define abstract dependency graphs.

Definition 5 (Abstract Dependency Graph [25]). *An* abstract dependency graph *(ADG) is a tuple* $G = (V, E, \mathfrak{D}, \mathcal{E})$ *where*

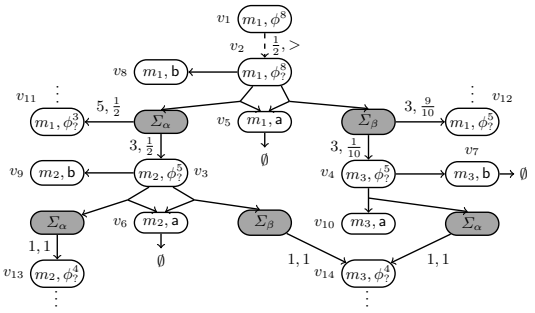
- V *is a finite set of vertices,*
- $E : V \to V^*$ *is an edge function from vertices to sequences of vertices such that* $E(v)[i] \neq E(v)[j]$ *for every* $v \in V$ *and every* $1 \leq i < j \leq |E(v)|$*, i.e. the co-domain of* E *contains only strings over* V *where no symbol appears more than once,*
- $\mathfrak{D}$ *is NOR with finitely representable elements, and*
- $\mathcal{E}$ *is a labelling function* $\mathcal{E} : V \to \mathcal{F}(\mathfrak{D})$ *such that* $\mathcal{E}(v) \in \mathcal{F}(\mathfrak{D}, |E(v)|)$ *for each* $v \in V$*, i.e. each edge* $E(v)$ *is labelled by an effectively computable monotone function* f *of arity that corresponds to the length of* $E(v)$*.*

In the following, we assume a fixed ADG $G = (V, E, \mathfrak{D}, \mathcal{E})$. For each vertex $v \in V$, $E(v)$ is a string containing all the vertices that represent dependencies of v and $\mathcal{E}(v)$ is the function computing the value of v given the values of all the dependencies of v in $E(v)$. An *assignment* is then a function $A \colon V \to D$, mapping each vertex to an element of the NOR $\mathfrak{D} = (D, \sqsubseteq, \bot)$. We let $\mathfrak{A}$ denote the set of all assignments and lift the ordering from $\mathfrak{D}$ to assignments: for any two assignments $A_1, A_2 \in \mathfrak{A}$, $A_1 \sqsubseteq A_2$ iff $\forall v \in V. A_1(v) \sqsubseteq A_2(v)$. It follows that $(\mathfrak{A}, \sqsubseteq)$ is a NOR, with minimum element $A_\bot$ defined for any $v \in V$ as $A_\bot(v) = \bot$. We define the *minimum fixed-point assignment* $A_{\min}$ for G as the minimum fixed point of the function $F : \mathfrak{A} \to \mathfrak{A}$, defined for any $v \in V$ as $F(A)(v) = \mathcal{E}(v)(A(v_1), A(v_2), \ldots, A(v_k))$ where $E(v) = v_1 v_2 \cdots v_k$. As each $\mathcal{E}(v)$ is monotone, it follows that F is a monotone function. In [25] it is proven, by applying standard reasoning for fixed points of monotonic functions [41], that $A_{\min}$ exists and is computable by repeated application of F on $A_\bot$. We end this section by presenting the result of [25]. For any $A \in \mathfrak{A}^{\mathbf{k}}$ let $F^i(A)$ be the i'th repeated application of F on A, defined for $i = 0$ as $F^i(A) = A$ and $F^i(A) = F(F^{i-1}(A))$ for $i > 0$.

Theorem 1 ([25]). *There exists* $j \in \mathbb{N}$ *such that* $F^k(A_\bot) = A_{\min}$ *for all* $k \geq j$*.*

4.2 The Reduction

We fix a game $\mathcal{G} = (\Pi, M, \{M_i\}_{i \in \Pi}, \to, \ell)$ for the remainder of this section and present the encoding of the problem $\mathcal{G}, m \models \phi$ for some state $m \in M$ and PWATL formula $\phi \in \mathcal{L}_{\mathsf{ATL}}$ by reduction to computing the minimal fixed point of a suitable abstract dependency graph $G = (V, E, \mathfrak{D}, \mathcal{E})$. In general, vertices of the graph are pairs (m, ϕ) where m is a state of $\mathcal{G}$ and $\phi \in \mathcal{L}_{\mathsf{ATL}}$ is a state-formula. These are referred to as *concrete vertices*. As our approach is symbolic, we introduce another type of vertex. For this, we let $\mathcal{L}^?_{\mathsf{ATL}} = \{\langle\!\langle C \rangle\!\rangle_{\triangleright ?}[\phi_1 U_{\leq \mathbf{k}} \phi_2] \mid \mathbf{k} \in \mathbb{N}^n, \phi_1, \phi_2 \in \mathcal{L}_{\mathsf{ATL}}\} \cup \{\langle\!\langle C \rangle\!\rangle_{\triangleright ?}[X_{\leq \mathbf{k}} \phi] \mid \mathbf{k} \in \mathbb{N}^n, \phi \in \mathcal{L}_{\mathsf{ATL}}\}$ be the set of all symbolic state-formulae. The *symbolic* vertices are then on the form $(m, \phi_?)$, where $\phi_? \in \mathcal{L}^?_{\mathsf{ATL}}$. We proceed by defining the domain $\mathfrak{D}$.



(a) ADG encoding of $\mathcal{G}, m_1 \models \phi$ for $\mathcal{G}$ from Figure 1a and $\phi = \langle\langle \bigcirc, \diamondsuit \rangle\rangle_{> \frac{1}{2}} [\mathsf{a}\, U_{\leq 8}\, \mathsf{b}]$

	v_1	v_2	v_3	v_4	$v_5 \cdots v_7$	$v_8 \cdots v_{10}$	$A_{\min}(v_{11\cdots 12})$	$A_{\min}(v_{13\cdots 14})$
A'	0	0	0	0	0	0	$\frac{1}{10}$	1
$F(A')$	0	$\frac{9}{100}$	1	1	1	$\widetilde{0}$	$\frac{1}{10}$	1
$F^2(A')$	0	$\frac{11}{20}$	1	1	1	$\widetilde{0}$	$\frac{1}{10}$	1
$F^3(A')$	1	$\frac{11}{20}$	1	1	1	$\widetilde{0}$	$\frac{1}{10}$	1

(b) Fixed point computation of ADG in Figure 2a

Fig. 2: Abstract dependency graph encoding example

The domain $\mathfrak{D}$ During the fixed point computation, the value of any node is, in general, a number that represents a lower bound on the probability of satisfaction. However, as we employ the *certain-zero* optimization of [20], we use also a special value $\widetilde{0}$, indicating that the value is 0 and can never change. Hence, 0 is a lower bound whereas $\widetilde{0}$ is an upper bound on the probability of satisfaction. We define the ordering depicted in Figure 3, where the dotted line represents all numbers between 0 and 1, and where $0 \sqsubseteq \widetilde{0}$ and $p_1 \sqsubseteq p_2$ if $p_1 \leq p_2$ and $p_1, p_2 \in [0, 1]$. Hence, the certain zero value $\widetilde{0}$ and the strictly positive probabilities in $(0, 1]$ are incomparable. Thus, the domain is given by $\mathfrak{D} = ([0, 1] \cup \{\widetilde{0}\}, \sqsubseteq, 0)$. For any concrete vertices (m, ϕ), the value assigned is either 0, 1 or $\widetilde{0}$. If the value becomes 1, m satisfies ϕ, thus whenever the root is assigned 1, the algorithm can safely terminate. However, if the value is 0, the current belief is that m does not satisfy ϕ and the algorithm cannot terminate as the value is a lower bound that may change. Once the value becomes $\widetilde{0}$, it is *certain* that m does not satisfy ϕ and the algorithm can terminate. For symbolic vertices $(m, \langle\langle C \rangle\rangle_{\triangleright ?} \psi)$, assigning a probability p to the vertex indicates the existance of a strategy for the coalition C, such that measuring paths from m satisfying ψ, yields a probability at least p, no matter the strategy for the remaining players in $\overline{C}$. Hence, $\mathcal{G}, m \models \langle\langle C \rangle\rangle_{\triangleright p} \psi$.

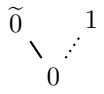


Fig. 3: Ordering $\sqsubseteq$

Anticipating the definition of the vertex labelling function, we define the operations $\min, \max, +$ and $\cdot$ on elements from the domain $\mathfrak{D}$. If the operands are regular probabilities in $[0, 1]$, the operations are defined in the natural way. Otherwise, for the certain zero value $\widetilde{0}$ and for any probability $p \in [0, 1]$ we let $\min\{\widetilde{0}, p\} = \widetilde{0}$, $\max\{\widetilde{0}, p\} = p$, $\widetilde{0} + p = p$ and $\widetilde{0} \cdot p = \widetilde{0}$. Hence, $\widetilde{0}$ behaves like

0 when used in operations with regular probabilities. If both operands are $\widetilde{0}$ we let $\min\{\widetilde{0},\widetilde{0}\} = \widetilde{0}$, $\max\{\widetilde{0},\widetilde{0}\} = \widetilde{0}$, $\widetilde{0} + \widetilde{0} = \widetilde{0}$ and $\widetilde{0} \cdot \widetilde{0} = \widetilde{0}$.

Graph construction We define the set of vertices V and for each $v \in V$, the edge function $E(v)$ and labelling function $\mathcal{E}(v)$. The root of the graph is $(m, \phi) \in V$ and the rest of the graph is constructed by induction on ϕ.

For any vertex on the form $v = (m_*, \phi_*)$, where $\phi_* \in \mathcal{L}_{\mathsf{ATL}}$, the following rules define the edge function $E(v)$ and labelling function $\mathcal{E}(v)$.

[$\phi_* = \mathsf{a}$]: The formula has no dependencies and can be verified directly by inspecting the labelling of the state. Hence, $E(v) = \varepsilon$ and if $\mathsf{a} \in \ell(m_*)$ then $\mathcal{E}(v) = 1$, otherwise $\mathcal{E}(v) = \widetilde{0}$.

[$\phi_* = \neg\mathsf{a}$]: We let $E(v) = \varepsilon$, $\mathcal{E}(v) = 1$ if $\mathsf{a} \notin \ell(m_*)$ and $\mathcal{E}(v) = \widetilde{0}$ otherwise.

[$\phi_* = \phi_1 \vee \phi_2$]: We let the vertices $(m_*, \phi_1), (m_*, \phi_2) \in V$ be the dependencies of v, hence $E(v) = (m_*, \phi_1)(m_*, \phi_2)$. As each successor receives a Boolean value, disjunction is naturally defined as the maximum of the values of the two successor vertices and we let $\mathcal{E}(v)(p_1, p_2) = \max\{p_1, p_2\}$.

[$\phi_* = \phi_1 \wedge \phi_2$]: Similar to disjunction we let $(m_*, \phi_1), (m_*, \phi_2) \in V$ be the dependencies of v, i.e, $E(v) = (m_*, \phi_1)(m_*, \phi_2)$ and $\mathcal{E}(v)(p_1, p_2) = \min\{p_1, p_2\}$.

[$\phi_* = \langle\!\langle C \rangle\!\rangle_{\triangleright\lambda}(\phi_1 U_{\leq \mathbf{k}} \phi_2)$]: The only dependency of v is the symbolic vertex $v' = (m_*, \langle\!\langle C \rangle\!\rangle_{\triangleright ?}[\phi_1 U_{\leq \mathbf{k}} \phi_2]) \in V$, i.e. $E(v) = v'$. As the value of v' is the probability p of satisfying the inner path formula, the value of v is 1 if and only if $p \triangleright \lambda$:

$$\mathcal{E}(v)(p) = \begin{cases} 1 & \text{if } p \triangleright \lambda \\ \widetilde{0} & \text{if } p = \widetilde{0} \wedge (\lambda > 0 \vee \triangleright = >) \\ 0 & \text{otherwise} \end{cases}$$

[$\phi_* = \langle\!\langle C \rangle\!\rangle_{\triangleright\lambda}(X_{\leq \mathbf{k}} \phi)$]: We let the symbolic vertex $v' = (m_*, \langle\!\langle C \rangle\!\rangle_{\triangleright ?}(X_{\leq \mathbf{k}} \phi)) \in V$ be the dependency of v, i.e. $E(v) = v'$. The labelling of v is given by:

$$\mathcal{E}(v)(p) = \begin{cases} 1 & \text{if } p \triangleright \lambda \\ \widetilde{0} & \text{if } p = \widetilde{0} \wedge (\lambda > 0 \vee \triangleright = >) \\ 0 & \text{otherwise} \end{cases}$$

For any vertex $v = (m_*, \phi_?)$ with $\phi_? \in \mathcal{L}^?_{\mathsf{ATL}}$, the edge function $E(v)$ and labelling function $\mathcal{E}(v)$ are given by the following rules:

[$\phi_? = \langle\!\langle C \rangle\!\rangle_{\triangleright ?}(\phi_1 U_{\leq \mathbf{k}} \phi_2)$]: To satisfy the inner path formula $\phi_1 U_{\leq \mathbf{k}} \phi_2$ for any path starting in m_*, either ϕ_2 must be satisfied by m_* or ϕ_1 must be satisfied by m_*. Hence, we let $v_1 = (m_*, \phi_1)$, $v_2 = (m_*, \phi_2)$ with $v_1, v_2 \in V$ be the two immediate dependencies of v. In case ϕ_2 is not satisfied by m_* but ϕ_1 is, the satisfaction of the inner path formula is due to the successors of m_*. Hence, any successor of m_* is also a dependency, if the cost of transitioning to the successor is within the formula bound $\mathbf{k}$. We let $\mathsf{Act}_{\mathbf{k}}(m_*) = \{\alpha_1, \ldots, \alpha_n\}$ be the $\mathbf{k}$-enabled actions in m_* and for any $\alpha_k \in \mathsf{Act}_{\mathbf{k}}(m_*)$ let $\mathtt{succ}(m_*)_{\alpha_k} = \{m_1^{\alpha_k}, \ldots, m_{j_{\alpha_k}}^{\alpha_k}\}$ be the set of all α_k-successors of m_* where, for all $1 \leq i \leq$

j_{α_k}, $\mathbf{w}_i^{\alpha_k} \leq \mathbf{k}$ is the cost and $p_i^{\alpha_k}$ is the probability of transitioning to $m_i^{\alpha_k}$, respectively.
For each $m_i^{\alpha_k}$ we let $v_i^{\alpha_k} = (m_i^{\alpha_k}, \langle\langle C\rangle\rangle_{\triangleright ?}(\phi_1 U_{\leq \mathbf{k}-\mathbf{w}_i^{\alpha_k}} \phi_2)) \in V$ be a dependency of m_*. Hence, the edge function of v is given as

$$E(v) = v_1 v_2 v_1^{\alpha_1} \cdots v_{j_{\alpha_1}}^{\alpha_1} \cdots v_1^{\alpha_n} \cdots v_{j_{\alpha_n}}^{\alpha_n} \ .$$

For defining the labelling $\mathcal{E}(v)(q_1, q_2, q_1^{\alpha_1}, \ldots, q_{j_{\alpha_1}}^{\alpha_1}, \ldots, q_1^{\alpha_n}, \ldots, q_{j_{\alpha_n}}^{\alpha_n})$, we let $q_\Sigma^{\alpha_k} = \sum_{i=1}^{j_{\alpha_k}} p_i^{\alpha_k} \cdot q_i^{\alpha_k}$ be the weighted sum of successor values for any action $\alpha_k \in \mathsf{Act}_{\mathbf{k}}(m_*)$. The exact labelling function of m_* depends on whether m_* is owned by a player in the coalition or not.
If $m_* \in M_i$ for some player $i \in C$ we let

$$\mathcal{E}(v)(q_1, q_2, q_1^{\alpha_1}, \ldots, q_{j_{\alpha_1}}^{\alpha_1}, \ldots, q_1^{\alpha_n}, \ldots, q_{j_{\alpha_n}}^{\alpha_n}) = \max\{q_2, \min\{q_1, q_\Sigma^{\alpha_1}\}, \ldots, \min\{q_1, q_\Sigma^{\alpha_n}\}\} \ .$$

Otherwise, if $m_* \notin M_i$ for all players $i \in C$ we let

$$\mathcal{E}(v)(q_1, q_2, q_1^{\alpha_1}, \ldots, q_{j_{\alpha_1}}^{\alpha_1}, \ldots, q_1^{\alpha_n}, \ldots, q_{j_{\alpha_n}}^{\alpha_n}) = \max\{q_2, \min\{q_1, q_\Sigma^{\alpha_1}, \ldots, q_\Sigma^{\alpha_n}\}\} \ .$$

[$\phi_? = \langle\langle C\rangle\rangle_{\triangleright ?}(X_{\leq \mathbf{k}}\phi)$]: Let $\mathsf{Act}_{\mathbf{k}}(m_*) = \{\alpha_1, \ldots, \alpha_n\}$ be the set of $\mathbf{k}$-enabled actions in m_* and for any $\alpha_k \in \mathsf{Act}_{\mathbf{k}}(m_*)$ let $\mathtt{succ}(m_*)_{\alpha_k} = \{m_1^{\alpha_k}, \ldots, m_{j_{\alpha_k}}^{\alpha_k}\}$ be the set of all α_k-successors of m_* where, for all $1 \leq i \leq j_{\alpha_k}$, $\mathbf{w}_i^{\alpha_k} \leq \mathbf{k}$ is the cost and $p_i^{\alpha_k}$ is the probability of transitioning to $m_i^{\alpha_k}$, respectively.
For each $m_i^{\alpha_k}$ we let $v_i^{\alpha_k} = (m_i^{\alpha_k}, \phi) \in V$ be a dependency of m_*. Hence, the edge function of v is given as $E(v) = v_1^{\alpha_1} \cdots v_{j_{\alpha_1}}^{\alpha_1} \cdots v_1^{\alpha_n} \cdots v_{j_{\alpha_n}}^{\alpha_n}$. For defining the labelling $\mathcal{E}(v)(q_1^{\alpha_1}, \ldots, q_{j_{\alpha_1}}^{\alpha_1}, \ldots, q_1^{\alpha_n}, \ldots, q_{j_{\alpha_n}}^{\alpha_n})$, we let $q_\Sigma^\gamma = \sum_{i=1}^{j_{\alpha_k}} p_i^{\alpha_k} \cdot q_i^{\alpha_k}$ be the weighted sum of successor values for any action $\alpha_k \in \mathsf{Act}_{\mathbf{k}}(m_*)$. The exact labelling function of m_* depends on whether m_* is owned by a player in the coalition or not.
If $m_* \in M_i$ for some player $i \in C$ we let

$$\mathcal{E}(v)(q_1^{\alpha_1}, \ldots, q_{j_{\alpha_1}}^{\alpha_1}, \ldots, q_1^{\alpha_n}, \ldots, q_{j_{\alpha_n}}^{\alpha_n}) = \max\{q_\Sigma^{\alpha_1}, \ldots, q_\Sigma^{\alpha_n}\} \ .$$

Otherwise, if $m_* \notin M_i$ for all players $i \in C$ we let

$$\mathcal{E}(v)(q_1^{\alpha_1}, \ldots, q_{j_{\alpha_1}}^{\alpha_1}, \ldots, q_1^{\alpha_n}, \ldots, q_{j_{\alpha_n}}^{\alpha_n}) = \min\{q_\Sigma^{\alpha_1}, \ldots, q_\Sigma^{\alpha_n}\} \ .$$

Monotonicity of the constructed labelling function $\mathcal{E}$ follows from the fact that the functions max, min, sum and product are monotonic functions. By applying the above definitions repeatedly from the root (m, ϕ), we obtain an abstract dependency graph encoding of the problem $\mathcal{G}, m \models \phi$.

Example 4. Consider again the stochastic game depicted in Figure 1a. For any $k \in \mathbb{N}$ we let $\phi^k = \langle\langle \bigcirc, \diamondsuit\rangle\rangle_{>\frac{1}{2}}[\mathsf{a}\, U_{\leq k} \mathsf{b}]$ and $\phi_?^k = \langle\langle \bigcirc, \diamondsuit\rangle\rangle_{>?}[\mathsf{a}\, U_{\leq k} \mathsf{b}]$. We now

encode the model-checking problem $\mathcal{G}, m_1 \models \phi^8$ into an abstract dependency graph $G = (V, E, \mathfrak{D}, \mathcal{E})$. A part of the resulting graph is visualised in Figure 2a. Edges connecting the vertices correspond to the specific monotone functions given by our encoding. The greyed out shapes are not vertices but part of the monotonic function for a symbolic node, responsible for computing a weighted sum of successor values, q^{γ}_{Σ}, as prescribed by the encoding. We let $E(v_i) = \varepsilon$ for $5 \leq i \leq 10$, $\mathcal{E}(v_i) = \widetilde{0}$ for $8 \leq i \leq 10$ and $\mathcal{E}(v_i) = 1$ for $5 \leq i \leq 7$. This is visualised by vertices having either no outgoing edge or an edge pointing to the empty set. In general, separate unlabelled edges encode a maximum, while a minimum is computed over each unlabelled edge. For vertex v_2, the edge function is given by $E(v_2) = v_3 v_4 v_5 v_8 v_{11} v_{12}$ and the function computed at v_2 is thus

$$\mathcal{E}(v_2)(q_3, q_4, q_5, q_8, q_{11}, q_{12}) = \max\left\{q_8, \min\{q_5, q^{\alpha}_{\Sigma}\}, \min\{q_5, q^{\beta}_{\Sigma}\}\right\}$$

where $q^{\alpha}_{\Sigma} = \frac{1}{2} \cdot q_{11} + \frac{1}{2} \cdot q_3$ and $q^{\beta}_{\Sigma} = \frac{1}{10} \cdot q_4 + \frac{9}{10} \cdot q_{12}$. The dashed edge encodes

$$\mathcal{E}(v_1)(q_2) = \begin{cases} 1 & \text{if } q_2 > \frac{1}{2} \\ \widetilde{0} & \text{if } q_2 = \widetilde{0} \\ 0 & \text{otherwise} \end{cases} .$$

Theorem 2 (Correctness). *Let $\mathcal{G} = (\Pi, M, \{M_i\}_{i \in \Pi}, \to, \ell)$ be a game, $m \in M$ a state and $\phi \in \mathcal{L}_{\mathsf{ATL}}$ a property. For the abstract dependency graph rooted by (m, ϕ) it holds that $\mathcal{G}, m \models \phi$ iff $A_{\min}((m, \phi)) = 1$.*

As our domain $\mathfrak{D}$ does not satisfy the ascending chain condition, we cannot reuse the termination argument from [25]. We instead prove the termination by relying on our assumption that all loops are of strictly positive magnitude.

Theorem 3 (Termination). *There is $k \in \mathbb{N}$ s.t. $F^j(A_\perp) = A_{\min}$ for all $j \geq k$.*

Example 5. Consider the abstract dependency graph in Figure 2a. For vertices $v_{11}, \ldots, v_{14}$, the minimal fixed point assignment is given by $A_{\min}(v_{11}) = A_{\min}(v_{12}) = \frac{1}{10}$ and $A_{\min}(v_{13}) = A_{\min}(v_{14}) = 1$. Assuming that these assignments have been pre-computed, we now repeatedly apply the fixed point operator to compute the minimal fixed point assignment to the remaining vertices. Hence, we start from an assignment A' such that $A'(v_i) = A_{\min}(v_i)$ for $11 \leq i \leq 14$ and $A'(v_i) = A_\perp(v_i)$ otherwise. The result can be seen in Figure 2b After 3 iterations, the fixed point has been computed with a value of 1 assigned to v_1, hence by Theorem 2 we can conclude $\mathcal{G}, m_1 \models \langle\!\langle \bigcirc, \diamondsuit \rangle\!\rangle_{> \frac{1}{2}} [\mathsf{a}\, U_{\leq 8} \mathsf{b}]$.

5 Implementation and Experimental Evaluation

We evaluate our implementation on three different PRISM-games case studies. In *robot coordination* [34] problem two robots must reach a goal by traversing a square grid without crashing into each other; a 3-dimensional weight encodes

experiment	prism	above	$\frac{\text{prism}}{\text{above}}$	exact	$\frac{\text{prism}}{\text{exact}}$	below10	$\frac{\text{prism}}{\text{below10}}$	below20	$\frac{\text{prism}}{\text{below20}}$
R-1-20-5	5.96	3.10	1.92	2.27	2.63	2.21	2.69	2.18	2.73
R-1-20-6	9.54	5.73	1.66	4.39	2.17	4.44	2.15	4.38	2.18
R-1-30-5	14.74	10.50	1.40	10.32	1.43	7.69	1.92	7.87	1.87
R-1-30-6	45.99	25.93	1.77	23.23	1.98	20.71	2.22	20.59	2.23
R-2-20-5	6.38	4.00	1.59	2.84	2.25	2.86	2.23	2.88	2.22
R-2-20-6	9.08	7.67	1.18	5.78	1.57	5.94	1.53	5.87	1.55
R-2-30-5	12.76	11.55	1.10	11.55	1.10	8.75	1.46	9.11	1.40
R-2-30-6	38.11	32.02	1.19	25.56	1.49	25.61	1.49	25.44	1.50
Average	17.82	12.56	1.48	10.74	1.83	9.78	1.96	9.79	1.96
S-1-10	1.03	0.17	6.11	0.11	9.06	0.12	8.54	0.10	10.34
S-1-20	3.32	2.14	1.55	2.07	1.60	0.95	3.48	0.91	3.64
S-2-10	1.00	0.19	5.21	0.10	9.66	0.11	9.15	0.11	9.27
S-2-20	3.74	2.47	1.51	2.37	1.57	1.03	3.62	1.08	3.45
S-3-10	0.98	0.18	5.55	0.11	8.70	0.11	9.19	0.10	9.57
S-3-20	3.95	2.59	1.52	2.30	1.72	1.29	3.05	1.07	3.69
S-4-10	1.22	0.20	6.11	0.10	11.73	0.12	10.16	0.11	11.23
S-4-20	4.84	2.49	1.94	2.47	1.96	2.31	2.10	1.11	4.36
Average	2.51	1.30	3.69	1.20	5.75	0.76	6.16	0.57	6.94
T-29-1697	50.30	55.54	0.91	56.27	0.89	55.75	0.90	53.73	0.94
T-18-1115	73.83	60.40	1.22	64.39	1.15	59.37	1.24	61.59	1.20
T-28-1803	34.43	40.21	0.86	38.53	0.89	36.94	0.93	34.84	0.99
T-29-1871	38.18	45.06	0.85	45.55	0.84	41.84	0.91	39.69	0.96
T-27-1907	38.32	20.17	1.90	17.44	2.20	17.33	2.21	17.89	2.14
T-20-1209	30.21	23.60	1.28	23.81	1.27	22.36	1.35	20.49	1.47
T-23-1565	37.27	28.34	1.32	30.49	1.22	26.96	1.38	26.96	1.38
T-16-828	20.92	27.90	0.75	26.92	0.78	26.33	0.79	25.16	0.83
Average	40.43	37.65	1.14	37.93	1.16	35.86	1.21	35.04	1.24

Fig. 4: R-A-B-C is a 2-robot model with A collaborating robots, cost-bound of B on a grid of size C with queries of the type $\langle\!\langle r1, \ldots, rA \rangle\!\rangle_{\triangleright\lambda}(\neg\text{crash } U_{\leq(\mathbf{B},\mathbf{B},\mathbf{B})} \text{ goal1})$. S-X-Y is a sensor model with 4 sensors with X collaborating sensors with a cost-bound of Y and the query $\langle\!\langle s1, \ldots, sX \rangle\!\rangle_{\triangleright\lambda}(\text{true } U_{\leq(\mathbf{Y},\mathbf{Y})} \text{ decision\_made})$. T-Q-R is task graph problem and checks whether all tasks can be completed within at most Q time using R energy by the query $\langle\!\langle \text{sched} \rangle\!\rangle_{\triangleright\lambda}(\text{true } U_{\leq(\mathbf{Q},\mathbf{R})} \text{ tasks\_complete})$.

the energy consumption of both robots and the time elapsed. In *collective decision making for sensor networks* [13] 4 sensors must agree on 3 preferable sites; a 2-dimensional weight encodes total energy consumption and time elapsed. In *task-graph-scheduling* [9,33], a set of tasks must be scheduled on two processors; a 3-dimensional weight encodes energy consumption for each processor and time elapsed. We also compare with a Python implementation for PCTL model-checking from [30] on the PRISM case study *synchronous leader election* [29].

A package to reproduce our results can be found at `http://people.cs.aau.dk/~am/LOPSTR2020/`. Our open-source implementation is written in C++ without platform specific code. To obviate the need to create our own parser for PRISM models, we modify the export functionality in PRISM-games to construct an explicit transition system that becomes an input to implementation. Furthermore, as PRISM-games do not directly support verification of multidimensional cost-bounded properties, we cannot rely on built-in reward structures

experiment	tool	above	$\frac{\text{python}}{\text{adg}}$	exact	$\frac{\text{python}}{\text{adg}}$	below10	$\frac{\text{python}}{\text{adg}}$	below20	$\frac{\text{python}}{\text{adg}}$
L-4-4-10	python	0.45	11.25	0.48	12.00	0.42	10.50	0.36	12.00
	adg	0.04		0.04		0.04		0.03	
L-5-4-12	python	3.67	14.12	2.97	11.88	3.14	12.56	2.71	16.94
	adg	0.26		0.25		0.25		0.16	
L-4-6-10	python	3.8	15.83	3.64	15.83	3.16	21.07	3.24	21.60
	adg	0.24		0.23		0.15		0.15	
L-6-4-14	python	36.99	25.69	38.32	27.57	35.99	25.89	28.29	31.79
	adg	1.44		1.39		1.39		0.89	
L-5-6-12	python	88.52	42.56	91.2	45.37	86.31	63.93	85.57	62.92
	adg	2.08		2.01		1.35		1.36	
Average	python	26.69	21.89	27.32	22.53	25.80	26.79	24.03	29.05
	adg	0.81		0.78		0.64		0.52	

Fig. 5: L-N-K-W is a leader election model with N processes, K choices and queries of the form $\mathcal{P}_{\triangleright\lambda}(\text{true } U_{\leq(\mathbf{W},\mathbf{2W},\mathbf{3W})} \text{ elected})$. Additionally, python denotes the implementation from [30] and adg denotes our implementation.

and instead introduce variables to capture the accumulated cost. For each model-checking question, we bound the variables by a precision derived from the property, effectively creating a bounded unfolding of the original model, sufficient for verifying the query in question. As the model is bounded by the query precision, it is sufficient to verify in PRISM-games the corresponding unbounded query to solve the original model-checking problem.

5.1 Results

Experiments are run on a Ubuntu 14.04 cluster with AMD Opteron 6376 processors. Each experiment has a maximum time-out of two hours and 14GB of virtual memory. Figure 4 displays the experimental data for the PRISM-games comparison. The verified formulae are of the form $\langle\langle C \rangle\rangle_{\triangleright\lambda}(\psi)$ and specified in the caption of the table—the weight dimension being 3 for the robot experiment and 2 for the remaining two. The column labelled with 'prism' shows the time (in seconds) it took PRISM-games to verify a query (as PRISM-games computes the exact solution, the times do not vary for the different variants of the formula). The columns for 'above' ($\lambda = p + 0.000001$), 'exact' ($\lambda = p$), 'below10' ($\lambda = p - \frac{p}{10}$) and 'below20' ($\lambda = p - \frac{p}{5}$) describe the different instantiations of λ used in the queries, where p is the exact probability computed by PRISM-games. Hence, it is always the case that a formula is satisfied for 'exact', 'below10', 'below20' and never for 'above'. The remaining columns, e.g. $\frac{\text{prism}}{\text{above}}$, show the speedup-ratio. As both tools rely on the explicit engine of PRISM-games for model construction, we report only the time spent on verification, as the model construction time is identical for both tools.

The experiments show that for formulae that query the exact or slightly above probability, our on-the-fly approach achieves verification times comparable or better than those of PRISM-games. Our approach takes slightly more time to derive that a formula does not hold, which is expected for an on-the-fly method. Our running times in general improve as we allow for more slack in the λ bound.

The robot experiment achieves on average about twice as fast verification for the 'below10' and 'below20' queries. In the sensor experiment, the certain-zero approach in combination with on-the-fly verification achieves for the 'below20' on average seven times faster verification, sometimes showing an order of magnitude improvement. Regarding the memory consumption, our method uses on average 3.4 times less memory on the robot experiment, 11.0 times less memory on the sensor experiment and 1.5 times less memory on task graphs.

The efficiency of our approach comes from i) early termination including the certain-zero optimization and ii) the local (on-the-fly) construction and exploration of the ADG. In contrast to PRISM-GAMES, we do not calculate the entire fixed point but only what is necessary to answer the model-checking question. Experiments show that we are on average 30%, 50%, and 15% (resp.) times faster for the robot, sensor, and task graph cases (resp.) when terminating early as opposed to computing the entire fixed point.

Figure 5 displays the experimental data for the comparison with the Python PCTL model checker from [30], for the synchronous leader election case-study where the weight dimension is 3. Each row in Figure 5 describes a leader election instance, run using both the Python implementation (python) and our C++ implementation (adg). The columns labelled $\frac{\text{python}}{\text{adg}}$ show the speedup relative to the previous column (i.e. the column to left). The C++ implementation is an order of magnitude faster than the Python implementation and tends toward two orders of magnitude as the size of the model increases.

6 Conclusion

We presented an on-the-fly technique for answering whether a turn-based stochastic multiplayer game with weighted transitions satisfies a given alternating-time temporal logic formula with upper-bounds on the accumulated weight in the temporal operators and lower-bounds on the probabilities that a certain path formula is satisfied. Our approach reduces the problem to the computation of minimum fixed point on a recently introduced notion of abstract dependency graphs, using a novel reduction relying on a special abstract domain that includes the certain-zero optimization. We formally prove the correctness of our reduction and provide an efficient C++ implementation. On a series of experiments, we compare the performance of our approach with PRISM-games and show in several instances the advantage of using on-the-fly algorithm compared to the traditional value-iteration method. Our current implementation does not explicitly output winning strategies, however, this information can be recovered from the fixed point computed on the constructed ADG. Other interesting applications of the framework include verifying logics involving both minimal and maximal fixed points, such as the modal μ-calculus [24], efficient analysis of various process algebra such as CCS with quantities (generalizing [21]) and symbolic analysis of timed systems (see e.g [11]).

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM **49**(5), 672–713 (2002). https://doi.org/10.1145/585265.585270, https://doi.org/10.1145/585265.585270
2. Andova, S., Hermanns, H., Katoen, J.: Discrete-time rewards model-checked. In: Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS 2003, Marseille, France, September 6-7, 2003. Revised Papers. pp. 88–104 (2003). https://doi.org/10.1007/978-3-540-40903-8_8, http://dx.doi.org/10.1007/978-3-540-40903-8_8
3. Ashok, P., Chatterjee, K., Kretínský, J., Weininger, M., Winkler, T.: Approximating values of generalized-reachability stochastic games. In: LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020. pp. 102–115 (2020). https://doi.org/10.1145/3373718.3394761, https://doi.org/10.1145/3373718.3394761
4. Baier, C., Größer, M., Leucker, M., Bollig, B., Ciesinski, F.: Controller synthesis for probabilistic systems. In: Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004), 22-27 August 2004, Toulouse, France. pp. 493–506 (2004). https://doi.org/10.1007/1-4020-8141-3_38, https://doi.org/10.1007/1-4020-8141-3_38
5. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
6. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: Interval iteration for markov decision processes. In: Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. pp. 160–180 (2017). https://doi.org/10.1007/978-3-319-63387-9_8, https://doi.org/10.1007/978-3-319-63387-9_8
7. Baldan, P., König, B., Mika-Michalski, C., Padoan, T.: Fixpoint games on continuous lattices. Proc. ACM Program. Lang. **3**(POPL), 26:1–26:29 (2019). https://doi.org/10.1145/3290339, https://doi.org/10.1145/3290339
8. Baldan, P., König, B., Padoan, T., Mika-Michalski, C.: Fixpoint games on continuous lattices. CoRR **abs/1810.11404** (2018), http://arxiv.org/abs/1810.11404
9. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N.: Quantitative analysis of real-time systems using priced timed automata. Commun. ACM **54**(9), 78–87 (2011). https://doi.org/10.1145/1995376.1995396, https://doi.org/10.1145/1995376.1995396
10. Brázdil, T., Brozek, V., Forejt, V., Kucera, A.: Stochastic games with branching-time winning objectives. In: 21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings. pp. 349–358 (2006). https://doi.org/10.1109/LICS.2006.48, https://doi.org/10.1109/LICS.2006.48
11. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings. pp. 66–80 (2005). https://doi.org/10.1007/11539452_9, http://dx.doi.org/10.1007/11539452_9
12. Chatterjee, K., Randour, M., Raskin, J.: Strategy synthesis for multi-dimensional quantitative objectives. Acta Informatica **51**(3-4), 129–163 (2014). https://doi.org/10.1007/s00236-013-0182-6, https://doi.org/10.1007/s00236-013-0182-6

13. Chen, T., Forejt, V., Kwiatkowska, M.Z., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Formal Methods Syst. Des. **43**(1), 61–92 (2013). https://doi.org/10.1007/s10703-013-0183-7, https://doi.org/10.1007/s10703-013-0183-7
14. Chen, T., Forejt, V., Kwiatkowska, M.Z., Simaitis, A., Wiltsche, C.: On stochastic games with multiple objectives. In: Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings. pp. 266–277 (2013). https://doi.org/10.1007/978-3-642-40313-2_25, https://doi.org/10.1007/978-3-642-40313-2_25
15. Chen, T., Kwiatkowska, M.Z., Simaitis, A., Wiltsche, C.: Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In: Quantitative Evaluation of Systems - 10th International Conference, QEST 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings. pp. 322–337 (2013). https://doi.org/10.1007/978-3-642-40196-1_28, https://doi.org/10.1007/978-3-642-40196-1_28
16. Chen, T., Lu, J.: Probabilistic alternating-time temporal logic and model checking algorithm. In: Fourth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2007, 24-27 August 2007, Haikou, Hainan, China, Proceedings, Volume 2. pp. 35–39 (2007). https://doi.org/10.1109/FSKD.2007.458, https://doi.org/10.1109/FSKD.2007.458
17. Cloth, L., Katoen, J., Khattri, M., Pulungan, R.: Model checking Markov reward models with impulse rewards. In: 2005 International Conference on Dependable Systems and Networks (DSN 2005), 28 June - 1 July 2005, Yokohama, Japan, Proceedings. pp. 722–731 (2005). https://doi.org/10.1109/DSN.2005.64, https://doi.org/10.1109/DSN.2005.64
18. Condon, A.: On algorithms for simple stochastic games. In: Advances In Computational Complexity Theory, Proceedings of a DIMACS Workshop, New Jersey, USA, December 3-7, 1990. pp. 51–71 (1990). https://doi.org/10.1090/dimacs/013/04, https://doi.org/10.1090/dimacs/013/04
19. Condon, A.: The complexity of stochastic games. Inf. Comput. **96**(2), 203–224 (1992). https://doi.org/10.1016/0890-5401(92)90048-K, https://doi.org/10.1016/0890-5401(92)90048-K
20. Dalsgaard, A.E., Enevoldsen, S., Fogh, P., Jensen, L.S., Jensen, P.G., Jepsen, T.S., Kaufmann, I., Larsen, K.G., Nielsen, S.M., Olesen, M.C., Pastva, S., Srba, J.: A distributed fixed-point algorithm for extended dependency graphs. Fundam. Inform. **161**(4), 351–381 (2018). https://doi.org/10.3233/FI-2018-1707, https://doi.org/10.3233/FI-2018-1707
21. Dalsgaard, A.E., Enevoldsen, S., Larsen, K.G., Srba, J.: Distributed computation of fixed points on dependency graphs. In: Dependable Software Engineering: Theories, Tools, and Applications - Second International Symposium, SETTA 2016, Beijing, China, November 9-11, 2016, Proceedings. pp. 197–212 (2016). https://doi.org/10.1007/978-3-319-47677-3_13, http://dx.doi.org/10.1007/978-3-319-47677-3_13
22. Droste, M., Kuich, W., Vogler, H.: Handbook of Weighted Automata. Springer (2009)
23. Enevoldsen, S., Larsen, K.G., Mariegaard, A., Srba, J.: Dependency graphs with applications to verification. International Journal on Software Tools for Technology Transfer (STTT) pp. 1–22 (2020). https://doi.org/10.1007/s10009-020-00578-9, https://doi.org/10.1007/s10009-020-00578-9
24. Enevoldsen, S., Larsen, K.G., Srba, J.: Extended abstract dependency graphs, manuscript Under Submission

25. Enevoldsen, S., Larsen, K.G., Srba, J.: Abstract dependency graphs and their application to model checking. In: Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I. pp. 316–333 (2019). https://doi.org/10.1007/978-3-030-17462-0_18, https://doi.org/10.1007/978-3-030-17462-0_18
26. Fahrenberg, U., Juhl, L., Larsen, K.G., Srba, J.: Energy games in multiweighted automata. In: Proceedings of the 8th International Colloquium on Theoretical Aspects of Computing (ICTAC'11). LNCS, vol. 6916, pp. 95–115. Springer-Verlag (2011)
27. Hartmanns, A., Junges, S., Katoen, J., Quatmann, T.: Multi-cost bounded reachability in MDP. In: Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II. pp. 320–339 (2018). https://doi.org/10.1007/978-3-319-89963-3_19, https://doi.org/10.1007/978-3-319-89963-3_19
28. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. In: Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II. pp. 488–511 (2020). https://doi.org/10.1007/978-3-030-53291-8_26, https://doi.org/10.1007/978-3-030-53291-8_26
29. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Inf. Comput. **88**(1), 60–87 (1990). https://doi.org/10.1016/0890-5401(90)90004-2, https://doi.org/10.1016/0890-5401(90)90004-2
30. Jensen, M.C., Mariegaard, A., Larsen, K.G.: Symbolic model checking of weighted PCTL using dependency graphs. In: NASA Formal Methods - 11th International Symposium, NFM 2019, Houston, TX, USA, May 7-9, 2019, Proceedings. pp. 298–315 (2019). https://doi.org/10.1007/978-3-030-20652-9_20, https://doi.org/10.1007/978-3-030-20652-9_20
31. Kelmendi, E., Krämer, J., Kretínský, J., Weininger, M.: Value iteration for simple stochastic games: Stopping criterion and learning algorithm. In: Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. pp. 623–642 (2018). https://doi.org/10.1007/978-3-319-96145-3_36, https://doi.org/10.1007/978-3-319-96145-3_36
32. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: PRISM-games 3.0: Stochastic game verification with concurrency, equilibria and time. In: Proc. 32nd International Conference on Computer Aided Verification (CAV'20). LNCS, Springer (2020)
33. 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. pp. 379–396 (2019). https://doi.org/10.1007/978-3-030-31175-9_22, https://doi.org/10.1007/978-3-030-31175-9_22
34. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: Equilibria-based probabilistic model checking for concurrent stochastic games. In: Formal Methods - The Next 30 Years - Third World Congress, FM 2019, Porto, Portugal, October 7-11,

2019, Proceedings. pp. 298–315 (2019). https://doi.org/10.1007/978-3-030-30942-8_19, https://doi.org/10.1007/978-3-030-30942-8_19
35. Liu, X., Smolka, S.A.: Simple linear-time algorithms for minimal fixed points (extended abstract). In: Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings. pp. 53–66 (1998). https://doi.org/10.1007/BFb0055040, http://dx.doi.org/10.1007/BFb0055040
36. Nguyen, H.N., Rakib, A.: A probabilistic logic for resource-bounded multi-agent systems. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019. pp. 521–527 (2019). https://doi.org/10.24963/ijcai.2019/74, https://doi.org/10.24963/ijcai.2019/74
37. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994). https://doi.org/10.1002/9780470316887, https://doi.org/10.1002/9780470316887
38. Quatmann, T., Katoen, J.: Sound value iteration. In: Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. pp. 643–661 (2018). https://doi.org/10.1007/978-3-319-96145-3_37, https://doi.org/10.1007/978-3-319-96145-3_37
39. Shapley, L.S.: Stochastic games. Proceedings of the National Academy of Sciences **39**(10), 1095–1100 (1953). https://doi.org/10.1073/pnas.39.10.1095, https://www.pnas.org/content/39/10/1095
40. Svorenová, M., Kwiatkowska, M.: Quantitative verification and strategy synthesis for stochastic games. Eur. J. Control **30**, 15–30 (2016). https://doi.org/10.1016/j.ejcon.2016.04.009, https://doi.org/10.1016/j.ejcon.2016.04.009
41. Tarski, A., et al.: A lattice-theoretical fixpoint theorem and its applications. Pacific journal of Mathematics **5**(2), 285–309 (1955). https://doi.org/10.2140/pjm.1955.5.285

Appendix

A Proofs

Proof. Let $v \in V$ be a vertex. We prove the following, by structural induction:

1. If $v = (m, \phi)$ is a concrete state,

$$\mathcal{G}, m \models \phi \text{ iff } A_{\min}(v) = 1 \ .$$

2. If $v = (m, \phi_?)$ is a symbolic vertex, where $\phi_?$ is of the form $\langle\langle C \rangle\rangle_{\triangleright ?}(\psi_{\mathbf{k}})$, either

   – There exists a $p \in [0, 1]$ such that

$$A_{\min}(v) = \max_{\sigma_C \in \mathfrak{S}_C} \min_{\sigma_{\overline{C}} \in \mathfrak{S}_{\overline{C}}} \mathbb{P}_{\mathcal{G}}^{\sigma_C, \sigma_{\overline{C}}}(\{\pi \in \mathsf{Paths}(m) \mid \mathcal{G}, \pi \models \psi_{\mathbf{k}}\}) = p \ ,$$

   – or it is the case that

$$A_{\min}(v) = \widetilde{0} \text{ and } \max_{\sigma_C \in \mathfrak{S}_C} \min_{\sigma_{\overline{C}} \in \mathfrak{S}_{\overline{C}}} \mathbb{P}_{\mathcal{G}}^{\sigma_C, \sigma_{\overline{C}}}(\{\pi \in \mathsf{Paths}(m) \mid \mathcal{G}, \pi \models \psi_{\mathbf{k}}\}) = 0 \ .$$

Clearly the theorem follows from (1).
We proceed by a case-analysis.

[$v = (m, \mathsf{a})$]:
As v is a concrete vertex, we consider (1). By the encoding, it follows trivially that $A_{\min}(v) = 1$ if and only if $\mathsf{a} \in \ell(m)$ and by PWATL semantics we thus have $\mathcal{G}, m \models \mathsf{a}$.

[$v = (m, \neg\mathsf{a})$]:
Similar to the case $v = (m, \mathsf{a})$.

[$v = (m, \phi_1 \vee \phi_2)$]:
As v is a concrete vertex, we consider (1). By PWATL semantics, $\mathcal{G}, m \models \phi_1 \vee \phi_2$ if and only if $\mathcal{G}, m \models \phi_1$ or $\mathcal{G}, m \models \phi_2$. By the inductive hypothesis, $\mathcal{G}, m \models \phi_1$ if and only if $A_{\min}((m, \phi_1)) = 1$ and $\mathcal{G}, m \models \phi_1$ if and only if $A_{\min}((m, \phi_2)) = 1$. By the encoding, we thus have $A_{\min}(v) = \max\{A_{\min}((m, \phi_1)), A_{\min}((m, \phi_2))\} = 1$ if and only if $\mathcal{G}, m \models \phi_1 \vee \phi_2$, proving (1).

[$v = (m, \phi_1 \wedge \phi_2)$]:
Similar to the case $v = (m, \phi_1 \wedge \phi_2)$.

[$v = (m, \langle\langle C \rangle\rangle_{\triangleright\lambda}(\phi_1 U_{\leq \mathbf{k}} \phi_2))$]:
As v is a concrete vertex, we consider (1). By the encoding, vertex v has a single dependency, given by the vertex $v' = (m, \langle\langle C \rangle\rangle_{\triangleright ?}(\phi_1 U_{\leq \mathbf{k}} \phi_2)$.
For the left-to-right implication, we assume $\mathcal{G}, m \models \langle\langle C \rangle\rangle_{\triangleright\lambda}(\phi_1 U_{\leq \mathbf{k}} \phi_2)$ and prove that this implies $A_{\min}(v) = 1$. By PWATL semantics, we have that

$\mathcal{G}, m \models \langle\!\langle C \rangle\!\rangle_{\triangleright\lambda}(\phi_1 U_{\leq \mathbf{k}} \phi_2)$ implies that there exists a strategy for the coalition C such that the probability of satisfying the inner path formula yields a probability $p \in [0,1]$, such that $p \triangleright \lambda$. In particular, there exists a maximal one. By applying the inductive hypothesis for v', we then have $A_{\min}(v') = p$ and by the labelling function of v, we can conclude that $A_{\min}(v) = 1$ if and only if $A_{\min}(v') \triangleright \lambda$.
For the right-to-left implication, we assume $A_{\min}(v) = 1$ and prove that this implies $\mathcal{G}, m \models \langle\!\langle C \rangle\!\rangle_{\triangleright\lambda}(\phi_1 U_{\leq \mathbf{k}} \phi_2)$. By the labelling function of v, we immediately have that $A_{\min}(v) = 1$ if and only if $A_{\min}(v') \triangleright \lambda$. By applying the inductive hypothesis to v', we have that there exists a strategy σ_C for C that maximises the probability of satisfying the inner path formula, and that the probability is given by $A_{\min}(v')$. Hence, σ_C is a witness for the satisfaction of $\langle\!\langle C \rangle\!\rangle_{\triangleright\lambda}(\phi_1 U_{\leq \mathbf{k}} \phi_2)$ by m.

$[v = (m, \langle\!\langle C \rangle\!\rangle_{\triangleright\lambda}(X_{\leq \mathbf{k}} \phi))\ ]$:
Similar to the case $v = (m, \langle\!\langle C \rangle\!\rangle_{\triangleright\lambda}(\phi_1 U_{\leq \mathbf{k}} \phi_2))$.

$[v = (m, \langle\!\langle C \rangle\!\rangle_{\triangleright ?}(\phi_1 U_{\leq \mathbf{k}} \phi_2))\ ]$:
As v is a symbolic vertex, we consider (2). Let $\phi_?^{\mathbf{k}'} = \langle\!\langle C \rangle\!\rangle_{\triangleright ?}(\phi_1 U_{\leq \mathbf{k}'} \phi_2)$ for any $\mathbf{k}' \in \mathbb{N}^n$. We consider in the following the case where $m \in M_i$ for some player $M_i \in C$. The argument can easily be modified for the other case.
The encoding implies that $A_{\min}(v)$ is given by

$$\max\{A_{\min}((m,\phi_2), \min\{A_{\min}((m,\phi_1)), q_\Sigma^{\alpha_1}\}, \ldots, \min\{A_{\min}((m,\phi_1)), q_\Sigma^{\alpha_n}\}\} \quad .$$

where $q_\Sigma^\alpha = \sum_{i=1}^{j_\alpha} p_i^\alpha \cdot A_{\min}((m_i^\alpha, \phi_?^{\mathbf{k}-\mathbf{w}_\mathbf{i}^\alpha}))$ is the weighted sum of successor values for any $\mathbf{k}$-enabled action $\alpha \in \mathsf{Act}_\mathbf{k}(m)$.
We now consider three cases:
(a) $\mathcal{G}, m \models \phi_2$.

(b) $\mathcal{G}, m \not\models \phi_2$ and $\mathcal{G}, m \not\models \phi_1$.

(c) $\mathcal{G}, m \not\models \phi_2$ and $\mathcal{G}, m \models \phi_1$.

For (a), clearly $A_{\min}(v) = 1$ and by PWATL semantics it follows that the probability of satisfaction of the inner path formula by any path in $\mathsf{Paths}(m)$ is 1, independent of the strategies chosen for C and $\overline{C}$. Hence, $\max_{\sigma_C \in \mathfrak{S}_C} \min_{\sigma_{\overline{C}} \in \mathfrak{S}_{\overline{C}}} \mathbb{P}_\mathcal{G}^{\sigma_C, \sigma_{\overline{C}}}(\{\pi \in \mathsf{Paths}(m) \mid \mathcal{G}, \pi \models \psi_\mathbf{k}\}) = 1$.
For (b), similar reasoning implies that $A_{\min}(v) \in \{0, \widetilde{0}\}$ and the probability of satisfying the inner path formula is 0, no matter the coalition strategies. Hence, both cases of (2) are satisfied.
For (c), we have

$$A_{\min}(v) = \max\{q_\Sigma^{\alpha_1}, \ldots, q_\Sigma^{\alpha_n}\} = q_\Sigma^{\alpha_j}$$

for some $1 \leq j \leq n$. By application of the inductive hypothesis for (2), we have, for all $1 \leq i \leq j_{\alpha_j}$ that there exists a strategy ensuring that

the probability of picking a path from $m_i^{\alpha_j}$ that satisfies $\phi_1 U_{\leq \mathbf{k}-\mathbf{w_i}^{\alpha_j}} \phi_2$ is maximal and where the probability is given by $A_{\min}((m_i^{\gamma}, \phi_?^{\mathbf{k}-\mathbf{w_i}^{\gamma}}))$.
We consider now the two cases $A_{\min}(v) = p \in [0,1]$ and $A_{\min}(v) = \widetilde{0}$ in (2).
If $A_{\min}(v) = \widetilde{0}$ we have, by definition of operations on $\widetilde{0}$, that

$$A_{\min}((m_i^{\alpha_j}, \phi_?^{\mathbf{k}-\mathbf{w_i}^{\alpha_j}})) = \widetilde{0}$$

for any α_j-successor $m_i^{\alpha_j}$. Hence the maximal successor strategies have to assign probability 0 for the satisfaction of the formulae $\phi_1 U_{\leq \mathbf{k}-\mathbf{w_i}^{\alpha_j}} \phi_2$. As the action α_j is maximal, we can conclude that there exists no strategy that makes $\phi_1 U_{\leq \mathbf{k}} \phi_2$ satisfiable by paths from m and we can conclude that $\max_{\sigma_C \in \mathfrak{S}_C} \min_{\sigma_{\overline{C}} \in \mathfrak{S}_{\overline{C}}} \mathbb{P}_{\mathcal{G}}^{\sigma_C, \sigma_{\overline{C}}}(\{\pi \in \mathsf{Paths}(m) \mid \mathcal{G}, \pi \models \psi_{\mathbf{k}}\}) = 0$.
If $A_{\min}(v) = p \in [0,1]$, recall that

$$A_{\min}(v) = q_{\Sigma}^{\alpha_j} = \sum_{i=1}^{j_{\alpha_j}} p_i^{\alpha_j} \cdot A_{\min}((m_i^{\alpha_j}, \phi_?^{\mathbf{k}-\mathbf{w_i}^{\alpha_j}})) \ .$$

It then follows that there exists a maximizing strategy, ensuring that the probability of picking a path from m that satisfies $\phi_1 U_{\leq \mathbf{k}} \phi_2$ is $q_{\Sigma}^{\alpha_j}$, by extending the successor strategies with the action α_j. Hence,

$$A_{\min}(v) = \max_{\sigma_C \in \mathfrak{S}_C} \min_{\sigma_{\overline{C}} \in \mathfrak{S}_{\overline{C}}} \mathbb{P}_{\mathcal{G}}^{\sigma_C, \sigma_{\overline{C}}}(\{\pi \in \mathsf{Paths}(m) \mid \mathcal{G}, \pi \models \psi_{\mathbf{k}}\}) \ .$$

$[v = (m, \langle\!\langle C \rangle\!\rangle_{\rhd ?}(X_{\leq \mathbf{k}} \phi)) \ ]$:
Similar to the case $v = (m, \langle\!\langle C \rangle\!\rangle_{\rhd ?}(\phi_1 U_{\leq \mathbf{k}} \phi_2))$. □

Proof. As the accumulated cost of any loop of any game is of strictly positive magnitude, the abstract dependency graph encoding for any PWATL model-checking problem is a directed acyclic graph. Hence, by the monotonicity of $\mathcal{E}$, the value of any vertex can only change a finite number of times during the fixed point computation. □

B Adapting PRISM-games to our domain

Although PRISM-GAMES supports reward-structures that assign rewards to states and transitions, reward-bounded properties of the form $\phi_1 U_{\leq \mathbf{k}} \phi_2$ where the dimensionality of $\mathbf{k}$ is greater than 1, cannot directly be expressed in the query language of PRISM-games. To solve this problem, we modify the existing models for each of the three case-studies so that verifying a reward-bounded property is reduced to verifying a corresponding unbounded formula on the modified model. To illustrate our approach, we consider a PRISM-games encoding of the running example in Figure 1a, with respect to the query $\langle\!\langle C \rangle\!\rangle_{> \frac{1}{2}} [\mathsf{a}\, U_{\leq 8} \mathsf{b}]$ with $C = \{\bigcirc, \diamondsuit\}$. The resulting PRISM-games model can be seen in Listing 1.1. The model utilizes a query precision $\mathsf{q_w}$ to do a finite unfolding of the model sufficient

for verification of the query. Each module represents the behaviour of states belonging to a given player. For each cost dimension, we introduce a variable to keep track of the accumulated weight in the finite unfolding of the game. In this case the dimension is 1 and the variable is $\mathsf{acc_w}$. Using $\mathsf{weight_{OK}}$ on all guards ensures that the underlying transitions respect the precision $\mathsf{q_w}$ given by the query. Given the encoding, the corresponding unbounded property to verify is given by the PRISM-games query $\langle\langle \texttt{circle}, \texttt{diamond} \rangle\rangle P_{>\frac{1}{2}} [\text{“}\mathsf{a}\text{”}\ U\ (\text{“}\mathsf{b}\text{”}\ \&\ \text{“}\mathsf{weight_{OK}}\text{”})]$ where $\mathsf{q_w}$ is set to 8 in the model.

Listing 1.1: PRISM encoding of running example

```
1  smg
2  // The players
3  player circle circle   endplayer
4  player diamond diamond endplayer
5
6  const int q_w; // Query precision
7  const int max_cost = 5; // Max cost
8  const int w_max = q_w + max_cost; // Max accumulated weight
9  global acc_w : [0..w_max]; // Accumulated weight
10 formula weight_OK = (acc_w <= q_w); // Command guard
11 global state: [1..3] init 1; // m1,m2,m3
12
13 module circle
14         [] state=1 & weight_OK
15                 -> (0.5) : (state'=2) & (acc_w'=acc_w+3)  +
16                    (0.5) : (acc_w'=acc_w+5);
17         [] state=1 & weight_OK
18                 -> (0.1) : (state'=3) & (acc_w'=acc_w+3) +
19                    (0.9) : (acc_w'=acc_w+3);
20         [] state=3 & weight_OK
21                 -> (acc_w'=acc_w+1);
22 endmodule
23
24 module diamond
25         [] state=2 & weight_OK
26                 -> (state'=3) & (acc_w'=acc_w+1);
27         [] state=2 & weight_OK
28                 -> (acc_w'=acc_w+1);
29 endmodule
30
31
32 // State labels
33 label "weight_OK" = weight_OK;
34 label "b" = state=3;
35 label "a" = state=1 | state=2;
```