

On the Integration of Explainable Formal Methods Tools Into PMC-VIS

Christel Baier[1(✉)], Calvin Chau[1], Raimund Dachselt[2], Sascha Klüppelholz[1], Max Korn[1], Ricardo Langner[2], Johannes Lehmann[1], and Julián Méndez[2]

[1] Algebraic and Logical Foundations of Computer Science, Technische Universität Dresden, Dresden, Germany
{christel.baier,sascha.klueppelholz, johannes_alexander.lehmann}@tu-dresden.de

[2] Interactive Media Lab Dresden, Technische Universität Dresden, Dresden, Germany
julian.mendez2@tu-dresden.de

Abstract. This work reports on recent advances of the interactive visualization tool PMC-VIS that supports users in the development, analysis, and debugging of probabilistic systems modeled by Markov decision processes. As such, PMC-VIS contributes to the field of explainable formal methods and, in particular, to explainability in the context of probabilistic model checking. In previous versions of PMC-VIS, the probabilistic model checker PRISM was used exclusively to provide analysis results. We use a newly developed data import interface for PMC-VIS to integrate two analysis tools for MDPs: The first tool computes responsibility values, which quantify the impact of states on the satisfaction or violation of properties. The second tool computes witnessing subsystems, which are mathematical explications for the satisfaction of threshold conditions for probabilities or expectations. Both tools themselves enhance the aspect of explainability in the context of probabilistic model checking. The potential of this combined toolchain including visualization for explainability is showcased in a case study that demonstrates a typical debugging workflow, and we show how the general-purpose data import interface facilitates the integration of external formal methods tools for Markov decision processes and thus enables the inclusion of additional analytical data.

Keywords: Explainability · Formal Methods · Probabilistic Model Checking · Responsibility · Causality · Witnessing Subsystems · Certificates · PMC-VIS

1 Introduction

Formal methods provide mathematical rigor for system verification, but their practical usefulness crucially depends on the explainability of the computed

© The Author(s) 2027
T. Margaria and B. Steffen (Eds.): ISoLA 2026, LNCS 17144, pp. 136–157, 2027.
https://doi.org/10.1007/978-3-032-40121-2_8

results. A well-established branch of formal methods is *probabilistic model checking* (PMC), which considers the automated verification and analysis of probabilistic systems. To this end, the system and specification are formalized as a probabilistic model and logical formula, respectively, and subsequently given to a probabilistic model checker. Typically, the model checker then computes quantities concerning the worst-case (best-case) scenarios, e.g., maximal probability of system failure or minimal expected energy consumption. This gives detailed insights into the quantitative aspects of the system's operational behavior and, in principle, enables the exploration of design alternatives and debugging of undesired behavior. Yet, there remains a gap between the output of verification tools and what is needed for users and model developers to develop a deep understanding of the system and to take the required actions within debugging, maintenance, and model repair. This motivated the development of approaches for enhancing the explainability and trustworthiness of PMC, e.g., by providing additional diagnostic information in the form of sets of paths [2,35,36], witnessing subsystems [25,42,43,65,66], causal relations [11,47,48], Farkas and fixed-point certificates [1,17,25,38,44,64,68], responsibility or influence measures [3,7,18], etc. These concepts are of mathematical nature and provide explications in terms of high-dimensional data (e.g., vectors with values for each system state). Although domain experts know how to interpret them mathematically, it remains difficult to gain an intuitive understanding of such explications, even for experts. The overarching goal is thus to incorporate visualization techniques in verification tools that support users in understanding not only the system model, but also verification results and accompanying explications. This line of research towards *explainable formal methods* can be seen as a model-based facet of explainable artificial intelligence (XAI) that enhances the trustworthiness of systems and provides actionable insights (model repair, design, and adaptation of strategies).

Explainability can also be improved through visualization, by making large and complex structured datasets more accessible to humans. For instance, abstract model details can be transformed into suitable visual objects, such as graphs or maps. This bridges the gap between black box outputs and user understanding. In fact, all major model checkers, such as PRISM [52], Storm [37], nuXmv/NuSMV [16,20], and SPIN [39], provide export capabilities for models to graph-based or visually-renderable representations. Most commonly, DOT/Graphviz [23] or equivalent graph formats are used, which are best-suited for small models and non-interactive visualization via external general-purpose graph tools. However, workflow-oriented tools with interactive visual interfaces that also enable the exploration of large-scale models through different perspectives are more desirable. This enables their use throughout the development cycle of the model, such as design and inspection time, but potentially also at runtime. Interactive model exploration is supported by the model checkers mCRL2 [14], CADP [26] and UPPAAL [13]. PMC-VIS [51][1] is a visual explorer for large, multivariate graphs (here for MDPs) that makes PMC results (such as proba-

[1] imld.de/pmc-vis.

bilities of temporal logic properties, expectations of random variables, and also optimal MDP schedulers) visually accessible. At the backend PMC-VIS provides an interface to the PRISM model checker as data provider for PMC-results. For the embedding into a dynamic development cycle, PMC-VIS has recently been updated with editing and advanced debugging capabilities [50]. The long-term goal is to turn PMC-VIS into a model development environment (MDE) for PMC that integrates advanced visualization approaches for dynamic processes of developing, comparing, and debugging of formal operational models.

Contribution. The work presented in this article combines these two directions of explainability by integrating two analytical tools into a visual and interactive exploration setting. The tool integration builds on our latest version of PMC-VIS, to which we added (1) the ability to execute external tools and (2) a data import interface that we use to incorporate structured data, such as results from other model checkers or suitable analytical tools for MDP analysis, into PMC-VIS. The first analytical tool integrated to PMC-VIS computes *responsibility values* [9,10], which measure the impact of states on the satisfaction or violation of properties. In [9,10], the theory and tooling were limited to transition systems. We present here a generalization of this theoretical work from transition systems to MDPs, which includes solving stochastic two-player games, its implementation in a new responsibility tool, and the binding of the new tool to PMC-VIS. The second analytical tool computes *witnessing subsystems* [6] as mathematical explications for the satisfaction of threshold conditions for probabilities or expectations. Similarly, we present the integration of this tool into PMC-VIS and the new interactive visualization capabilities for responsibility values and witnessing subsystems within PMC-VIS. The potential of this combined toolchain for explainability is demonstrated in a case study illustrating typical workflows that occur in the debugging process. The version of PMC-VIS used in this paper, as well as all relevant experimental data, are available via Zenodo[2].

Related Work. The model checkers mCRL2 [14], CADP [26] and UPPAAL [13] provide extended capabilities for interactive model exploration. The mCRL2 toolset is based on a process-algebraic modeling formalism and provides an interface for connecting to various interactive 2D and 3D internal visualization tools, such as ltsgraph, ltsview, and DiaGraphica, as well as export formats for using external tools. However, its visualization capabilities are largely restricted to transition systems derived from individual process specifications. CADP is a toolbox for the specification and verification of asynchronous concurrent systems using process-algebraic languages. Its semantic model is also transition systems, and it supports model checking of temporal logic properties, but its visualization is limited to 2D state graphs without automated layout. UPPAAL is based on networks of timed automata and provides graphical simulation and visualization support for real-time systems. In particular, diagnostic traces produced by the

[2] doi.org/10.5281/zenodo.19810393.

model checker can be loaded into the simulator and visualized, including as message sequence charts, but the visualization support is primarily tied to traces and simulated executions rather than broader system-level abstractions. None of these three tools provides visual support for the stochastic case. PRISM [52] and Storm [37] support basic graph output via Graphviz [23].

Besides model checkers, several tools employ visualization to support analysis of MDPs in specific domains, such as HMMEditor [21] for DNA, RNA, and protein sequencing, or TraceVis [29,30] for grid-world driving with neural networks. For general purpose MDPs, systems like MDPVis [57] employ heatmaps and aggregated charts to analyze evaluation policies, parameters, and state development. MDPVis does not, however, employ node-link diagrams to explicitly analyze specific states or the MDP topology. Other tools that support analyzing the results, but not the model explicitly, are Palette [46], Theseus [27], and DPVis [53]. Furthermore, visualization for transition graphs has been researched by van Ham et al. [33,34] who proposed a 3D view with cone shapes, and by Pretorius et al., who adapted the conic view to 2D in the aforementioned tool DiaGraphica [61]. They also proposed representations that employ hypergrids, principal component analysis, and parallel histograms [60].

2 Preliminaries

We first introduce *Markov decision processes* (MDPs), which are the main model of PMC-VIS. Afterwards, we consider *turn-based stochastic games* (SGs), which are a two-player extension of MDPs and become relevant in Sect. 4.

Markov Decision Processes (MDPs). An MDP [62] $\mathcal{M} = (S, s_{init}, Act, \mathbf{P})$ is a tuple, where S is a finite set of *states*, $s_{init} \in S$ the *initial state*, Act a finite set of *actions*, and $\mathbf{P}\colon S \times Act \times S \to [0,1]$ a *transition function* such that $\sum_{s' \in S} \mathbf{P}(s,a,s') \in \{0,1\}$ for all $(s,a) \in S \times Act$. An action a is *enabled* in a state s if $\sum_{s' \in S} \mathbf{P}(s,a,s') = 1$ and we write $Act(s)$ to denote the enabled actions in s. A *path* π in $\mathcal{M}$ is a sequence $s_0 a_0 s_1 a_1 \ldots$ such that $s_0 = s_{init}$ and $\mathbf{P}(s_i, a_i, s_{i+1}) > 0$ for all $i \geq 0$. The sets of *finite* and *infinite* paths in $\mathcal{M}$ are denoted by $FPaths(\mathcal{M})$ and $IPaths(\mathcal{M})$, respectively.

A (positional) *scheduler* is a function $\mathfrak{S}\colon S \to Act$ such that for all $s \in S$, we have $\mathfrak{S}(s) \in Act(s)$. Given a scheduler $\mathfrak{S}$, we consider the *standard probability measure* $\Pr^{\mathfrak{S}}_{\mathcal{M}}$ on sets of infinite paths (see [8, Def. 10.9]). For a measurable objective $\Phi \subseteq IPaths(\mathcal{M})$, we define $\Pr^{\max}_{\mathcal{M}}(\Phi) = \sup_{\mathfrak{S}} \Pr^{\mathfrak{S}}_{\mathcal{M}}(\Phi)$ and $\Pr^{\min}_{\mathcal{M}}(\Phi) = \inf_{\mathfrak{S}} \Pr^{\mathfrak{S}}_{\mathcal{M}}(\Phi)$. For $G \subseteq S$, we write $\Diamond G$ and $\Box G$ to denote the objective of *eventually reaching* G and *always remaining in* G, respectively.

Turn-based Stochastic Games (SGs). Turn-based SGs are a two-player extension of MDPs, where states are either controlled by Player Max or Player Min. Formally, a (turn-based) SG [24, Ch. 7] $\mathcal{G}$ is a tuple $(S_{max}, S_{min}, s_{init}, Act, \mathbf{P})$, where S_{max} and S_{min} are the finite sets of *states* controlled by Player Max and Player Min, respectively, (and we write $S := S_{min} \dot\cup S_{max}$) and s_{init}, Act and $\mathbf{P}$ are the same as for MDPs. The sets of finite and infinite paths are defined analogously to MDPs. In the context

of SGs, we refer to schedulers as *strategies* and distinguish between strategies of Player Max and Player Min. A strategy for Player Max is a function $Strat_{max} : S_{max} \to Act$. Strategies for Player Min are defined analogously. As before, given a strategy pair $Strat_{max}$, $Strat_{min}$, we consider the standard probability measure $\Pr_{\mathcal{G}}^{Strat_{max},Strat_{min}}$. For a measurable objective $\Phi \subseteq IPaths(\mathcal{G})$, we define $\Pr_{\mathcal{G}}(\Phi) = \sup_{Strat_{max}} \inf_{Strat_{min}} \Pr_{\mathcal{G}}^{Strat_{max},Strat_{min}}(\Phi)$.

3 Overview on PMC-VIS

This section provides an overview of the current implementation of PMC-VIS [50, 51] and the extensions made for embedding existing explainable formal methods tools. Figure 1 shows the general extendable architecture. The current frontend implementation consists of two linked web components: (1) a *Code Editor* for the PRISM modeling language [52], and (2) a multivariate graph view employing degree-of-interest (DOI) graphs [32] to dynamically explore MDPs and parallel coordinates plots (PCP) [41] to inspect state properties and PMC-results. The multivariate graph view can be instantiated multiple times across partitions of the MDP, thereby employing multiple DOI graphs and PCPs simultaneously [51]. However, for simplicity, in the following, we will refer to (all) the DOI graphs as *Graph Explorer*, and to the PCPs as *Property Explorer*.

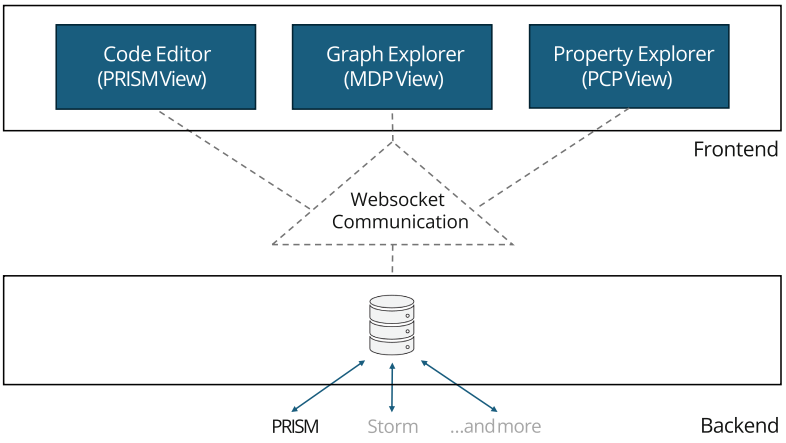


Fig. 1. PMC-VIS general architecture.

The main views are, on the one hand, independent, such that the editor can be shown on a web browser on one computer (or shown on another display) and the multivariate graph view on another computer (or shown on a different display). On the other hand, the views are linked, such that changes or selections made in one view result in changes or selections in the other views. To achieve this, PMC-VIS builds on a client–server architecture that is used to connect

the views via WebSocket-based communication and to the backend server. Due to the client–server architecture, the frontend user interface is decoupled from expensive computations occurring in the backend. The current implementation provides a connection to the PRISM model checker in the backend, which serves as the sole data provider. At the end of this section we will see how the new data import interface enables the integration of other (explainable) formal methods tools. We will now describe the main features of PMC-VIS per view. The typical starting point of the model development process is the code editor.

Code Editor (PRISM View). The Code Editor supports the PRISM modeling and specification language [52]. We developed a *Visual Studio Code* extension to provide functionalities of integrated development environments (IDEs) for more commonly used languages, such as creating and organizing projects (and models within), syntax highlighting, code completion, etc. From the code editor, the visualizations can be opened (by clicking a play button) to explore the resulting MDP by expanding the graph on demand from the initial states. This is illustrated in Fig. 2. This is possible even before the entire state space has been built.

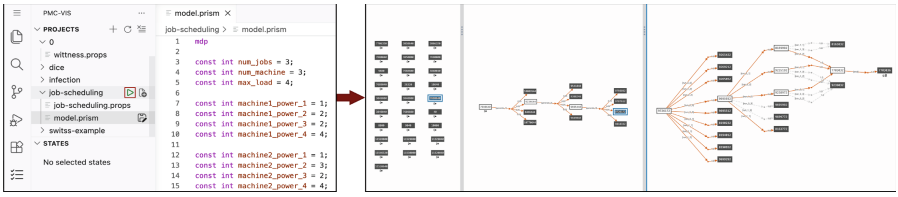


Fig. 2. Code Editor (left) and simulation on the Graph Explorer (right).

We will revisit the editor later in this section to see in more detail, e.g., how PMC-results computed in the backend and shown in the other two views lead to highlighting in the PRISM code, closing one loop of the development cycle.

Graph Explorer (MDP View). The right-hand side of Fig. 2 and Fig. 3(a) show our Graph Explorer. It visualizes parts of an MDP using DOI graphs [32]: starting in the initial states and expanding successor states on demand via user interaction (e.g., double click). Nodes correspond to MDP states and carry individual IDs. Edges correspond to transitions and show the name of nondeterministic choices as well as transition probabilities. Users can select and drag states, as well as pan and zoom on the views. Based on the available nondeterministic choices, users can explore and interactively simulate the model by expanding corresponding paths. Doing so adds more states to the visualization and thus eventually produces clutter. This is why we allow creating new instances of the view (i.e., in new *panes*, as shown in Fig. 2). Each pane displays a part of the MDP. Users can switch between panes and/or remove panes to continue exploration by starting from another existing pane. State-based information, e.g., about atomic

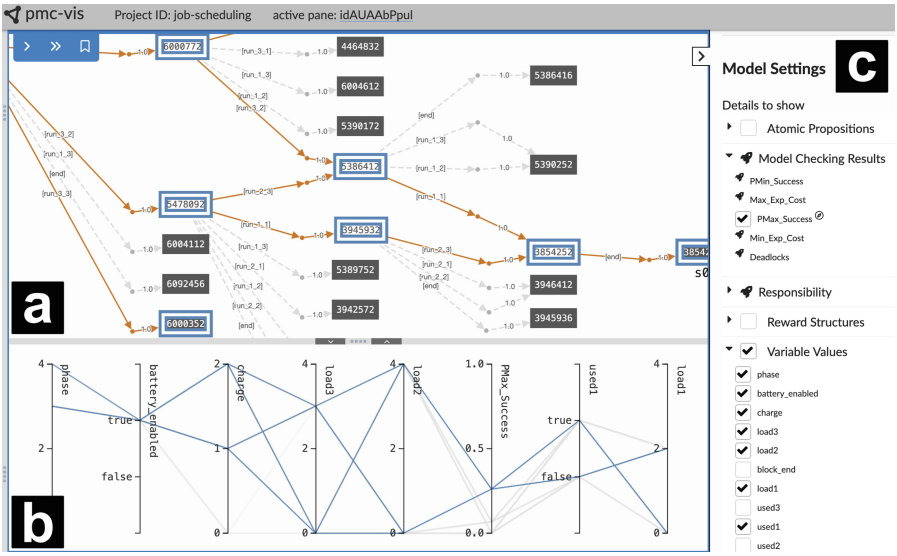


Fig. 3. The PMC-VIS frontend containing the Graph Explorer (a), the Property Explorer (b), and the context and options sidebar (c).

propositions and evaluations of PRISM variables can be queried for each node and shown as tooltips. PMC-results will only be computed (and shown) when the corresponding formula is selected in the menu bar (cf. Figure 3(c)) invoking PRISM asynchronously and filling the database with model checking results. After this, PMC-results can be shown as additional state information and optimizing schedulers can be shown via highlighting of actions within the Graph Explorer to guide users while navigating the MDP. Moreover, all state (or state-action pair) information can be shown in the Property Explorer.

Property Explorer (PCP View). Figure 3(b) shows a single parallel coordinates plot (PCP) [41] as used in the Property Explorer. The "properties" (e.g., atomic propositions, PRISM variables, PMC-results, etc.) of all states or state-action pairs are encoded as parallel axes. For each state(-action pair), the corresponding values on each axis are then connected, drawing a polyline across all properties. PCPs allow identifying and filtering states (and actions) according to their properties: Dragging along an axis ("brushing") selects all lines with values within the selected range. Selections are synchronised between the Graph and Property Explorers and the Code editor: In Fig. 3, the nodes with blue borders in (a) correspond to the blue polylines in (b).

Code Editor Highlighting. The Code Editor receives the current selection on the Graph and Property Explorers and highlights relevant PRISM commands. This is shown in Fig. 4. Here, green highlighting marks commands/actions as enabled in the current state, red means that they are blocked, and yellow means that the command would be enabled locally within the module, but is currently

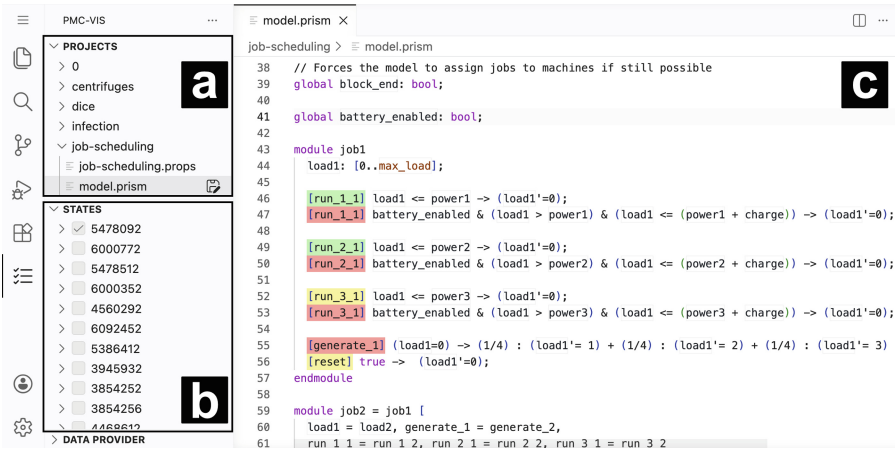


Fig. 4. The Code Editor, containing an overview of the current projects (a) and selected states (b) in the sidebar, as well as the text-editor (c) with syntax highlighting and traffic-light color coding for the selected state.

blocked by a command within another module due to blocking of a synchronous action. In such cases, we provide a jump-to function to the blocking command. This highlighting aids debugging, as simple patterns can be detected when comparing (parts of) modules with similar logic.

The New Data Import Interface. For this paper, we equipped PMC-VIS with a *data import interface* that allows binding external tools as data providers. For now, we support only tools that provide information for states (or state-action pairs). Calling an external tool from PMC-VIS requires a tool-specific implementation of this interface that specifies, e.g., the name of the executable, relevant command line switches, and how to transform the textual output of the tool into a *comma-separated values* (CSV) file. The results are then stored in the database of PMC-VIS where, e.g., model checking results are also stored. After registering a tool-specific class in a configuration file, PMC-VIS treats the external tool as a data provider. The tool can then be called from the PMC-VIS sidebar (Fig. 3(c)), and the results are displayed as additional axes in the Property Explorer once the computation finishes. In addition, data providers can define highlighting (based on specific properties) of states and state-action pairs in the Graph Explorer, similar to the scheduler highlighting mentioned in Sect. 3. Additional context per state can be shown as tooltip text and custom syntax highlighting is supported for the Code Editor. Detailed information about the interface can be found in the documentation of PMC-VIS[3].

For tools that compute state-based information and export it to CSV, the results can be imported without additional implementation via the project manager of PMC-VIS. Together with state mapping, one can then propagate those

[3] doi.org/10.5281/zenodo.19810393.

results to the Graph and Property Explorers. We tested this by importing model checking results from Storm [37] and from a tool that determines the causal relations between MDP states [11], the results of which were then visualized directly on PMC-VIS. However, for tools without CSV export, our data import interface ensures a tight integration at the cost of minimal tool-specific implementation.

This paper provides bindings for (1) a new tool for responsibility and (2) a tool for witnessing subsystems, which are introduced in the following sections.

4 Responsibility

Responsibility values quantify how much influence a given component has on the satisfaction or violation of requirements. This is orthogonal to the fulfillment probabilities computed by a probabilistic model checker, as a state can have high fulfillment probability but low responsibility. Identifying responsible states aids debugging by highlighting system components whose non-deterministic behavior influences the outcome. Responsibility can also be used to improve system reliability by determining the system components whose failure has the highest impact on overall reliability.

There is a wide range of techniques to assess responsibility [18,28,40,58]. Many [15,49,54] are based on Halpern and Pearl's notion of actual causality [31] by assessing responsibility under *contingencies*. One class of methods for quantifying responsibility are power indices, which measure the average marginal contribution of each component. The Shapley value [63] is the only power index which satisfies the axioms of efficiency, symmetry, linearity, and null player [67]. It is frequently used to assess responsibility [7,55,56]. Other power indices include the Banzhaf index [22]. Alternatively, responsibility of a component can be inversely correlated to the size of the smallest cause containing the component [12,19]. A distinction is made between *forward* and *backward* responsibility [59]. Forward responsibility assesses a system prior to execution, while backward responsibility takes a specific (failing) system execution and assigns responsibility for this specific outcome.

This section presents a notion of responsibility first introduced in [56] and extended to the backward case in [3]. Two-player games are used to determine winning coalitions. If a state is part of many winning coalitions, it has high responsibility. In the following, we extend the definitions from [3,56] to the probabilistic case. We primarily consider reachability and safety objectives (but [56] and [9] analyze more expressive objective classes).

Definition 1 (Value of a Coalition). *Let* $\mathcal{M} = (S, s_{init}, Act, \mathbf{P})$ *be an MDP,* $\Phi \subseteq IPaths(\mathcal{M})$ *a measurable objective and* $C \subseteq S$*. Then the* value of coalition C *is* $val(C) = \Pr_{\mathcal{G}}(\Phi)$*, where* $\mathcal{G} = (C, S \setminus C, s_{init}, Act, \mathbf{P})$ *is a turn-based stochastic game.*

The Shapley value transforms this value function (i.e. a value per set of states) into responsibility values (i.e. a value per state). To this end, the Shapley value computes a weighted average over the marginal contribution of a state to each coalition.

Definition 2 (Responsibility Values). *Let $\mathcal{M}$ be an MDP with states S and $\Phi \subseteq IPaths(M)$ a measurable objective. The* responsibility value*, often shortened to* responsibility*, of a state $s \in S$ is given by*

$$Resp(s) = \sum_{C \subseteq S} \frac{(|S| - |C| - 1)! \cdot |C|!}{|S|!} \cdot (val(C \cup \{s\}) - val(C)).$$

For $s \in S$, $C \subseteq S$, (s, C) is called a switching pair *if $val(C \cup \{s\}) - val(C) > 0$.*

For every coalition C, the Shapley value measures how much the probability of fulfilling the objective is increased by adding s. If the probability increases for many coalitions, then s has high responsibility.

These definitions consider the forward view. [3] extends this to the backward view by adding an *engraving construction* that fixes the behavior of some states to follow a previously-observed execution. Here, we restrict ourselves to the forward case, as there is no straight-forward extension of backward responsibility to probabilistic models. Responsibility is further extended in [10], which introduces the notion of *component-based responsibility*, where the state space of the model is partitioned into groupsand each group gets a single responsibility value. [9] presents several efficient algorithms for the non-probabilistic case. These works deal with non-probabilistic systems only. The definitions in this section generalise responsibility to the probabilistic case, and we similarly have extended the implementation to support probabilistic models.

In the following, we first showcase how state-based responsibility is visualized in PMC-VIS before demonstrating how component-based responsibility is visualized in the code editor.

4.1 State-Based Responsibility in PMC-VIS

Figure 5 shows PMC-VIS displaying responsibility values. The model contains two variables `x` and `y`, atomic propositions `special` and `goal` and the property `ReachGoal` $= \Pr^{\max}_{\mathcal{M}}(\Diamond \texttt{Goal})$. The right-most axis in the PCP shows responsibility values. The range from 0.1 to 0.3 is selected in the example, which highlights states whose responsibility falls in this range. This quickly shows that these states have `y` $=$ `false` and `goal` $=$ `false` in common.

4.2 Component-Based Responsibility in PMC-VIS

The previous section dealt with state-based responsibility, where each state has a separate responsibility value. However, this is both computationally intractable for large models and often too fine-grained to provide a practical utility. After all, models are specified in a higher-level language (the Prism modeling language in the case of PMC-VIS), so it is desirable to ascribe responsibility to the syntactical elements of the high-level description.

To this end, [10] provides an approach based on state grouping. The state-space of the MDP is partitioned into groups such that each group corresponds

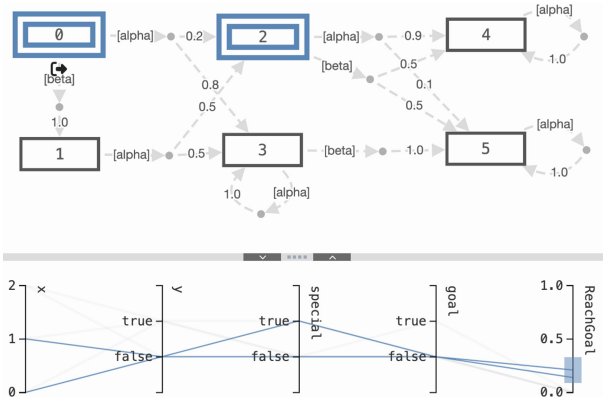


Fig. 5. PMC-VIS's graph view (top) and PCP (bottom) visualizing a 6-state MDP. The PCP shows variable values (`x` and `y`), atomic propositions (`special` and `goal`) and the responsibility for fulfilling objective `ReachGoal`. States with high responsibility are selected by the PCP brush. The states and corresponding lines are highlighted in the graph view and PCP, respectively.

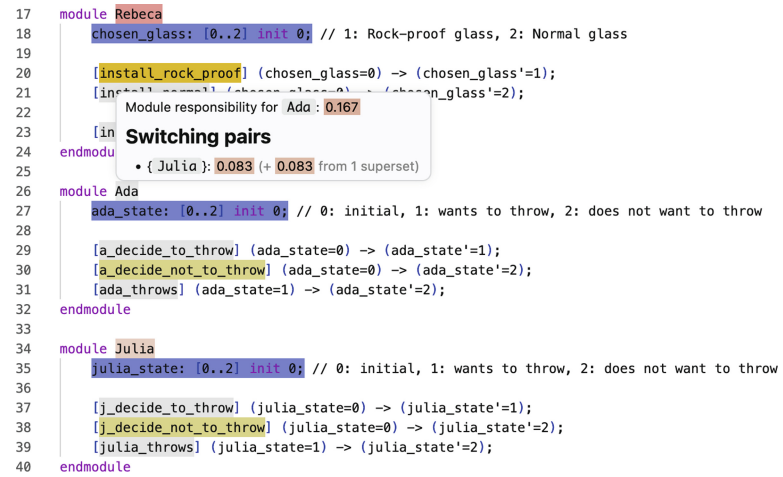
to a logical unit of the PRISM model. This can be done per module, per action, per variable value (for a given subset of variables) and per label. Some constructions require a transformation of the PRISM model or the MDP, but this is an implementation detail hidden from the user.

All four methods of grouping are incorporated into the PMC-VIS editor. Label and variable grouping can be configured by selecting which labels and variable values to group by.

Example. Figure 6 shows a model with module, variable and action responsibility being displayed using red, blue and yellow syntax highlighting. Different colors allow several responsibility techniques to be displayed at the same time. Each highlight also has a tooltip that gives additional information. In the example, this indicates that $(\texttt{Ada}, \{\texttt{Julia}\})$ is a switching pair that contributes 0.083 to her total responsibility score.

Superset Switching Pairs. The presence of "dummy states" that have no responsibility contaminates the set of switching pairs. If there is a state s_d with no responsibility, then for every switching pair (s, C), the pair $(s, C \cup \{s_d\})$ is also switching. To declutter the list, such "superset switching pairs" are summarized in the switching pair list. For example, in Fig. 6, there is one superset for $(\texttt{Ada}, \{\texttt{Julia}\})$, which also contributes 0.083 to Ada's total responsibility.

Per-label and Variable-Value Grouping. For module and action responsibility, there is a direct correspondence between syntax elements and state groups. For label and variable grouping, there is not, so these require special treatment:




```
module Rebeca
    chosen_glass: [0..2] init 0; // 1: Rock-proof glass, 2: Normal glass

    [install_rock_proof] (chosen_glass=0) -> (chosen_glass'=1);
    [install_normal] (chosen_glass=0) -> (chosen_glass'=2);

    [in
endmodu

module Ada
    ada_state: [0..2] init 0; // 0: initial, 1: wants to throw, 2: does not want to throw

    [a_decide_to_throw] (ada_state=0) -> (ada_state'=1);
    [a_decide_not_to_throw] (ada_state=0) -> (ada_state'=2);
    [ada_throws] (ada_state=1) -> (ada_state'=2);
endmodule

module Julia
    julia_state: [0..2] init 0; // 0: initial, 1: wants to throw, 2: does not want to throw

    [j_decide_to_throw] (julia_state=0) -> (julia_state'=1);
    [j_decide_not_to_throw] (julia_state=0) -> (julia_state'=2);
    [julia_throws] (julia_state=1) -> (julia_state'=2);
endmodule
```

Fig. 6. Visualization of grouped responsibility in the source code. Here, red, blue and yellow correspond to module, variable and action responsibility. A tooltip provides additional information, e.g. (Color figure online) that `Ada` has one switching pair with `Julia` that contributes 0.083 to her overall responsibility. Ada has one additional switching pair that is a superset of {`Julia`} and thus not listed separately.

- For **label groups**, states are grouped together based on which labels hold in them. For example, for two labels, ℓ_1 and ℓ_2, there are four groups $\varnothing$, $\{\ell_1\}$, $\{\ell_2\}$ and $\{\ell_1, \ell_2\}$. When hovering over, e.g., ℓ_1 in the source code, the total responsibility $\mathit{Resp}(\{\ell_1\}) + \mathit{Resp}(\{\ell_1, \ell_2\})$ is shown. This is followed by a list of all groups containing ℓ_1, i.e., $\{\ell_1\}$ and $\{\ell_1, \ell_2\}$ and their associated responsibility. Finally, all switching pairs are listed, similar to above.
- **Variable-valuation grouping** works as follows. For variable x, we denote its valuations by $\mathit{Vals}(x)$. For a set of variables $x_1, \ldots, x_n$, the set of valuation groups is denoted by $\mathit{Vals}(x_1, \ldots, x_n) = \mathit{Vals}(x_1) \times \cdots \times \mathit{Vals}(x_n)$. The responsibility tool provides a responsibility value for each group. For variable x and $d \in \mathit{Vals}(x)$, we denote their total responsibility by $\mathit{Resp}(x = d) = \sum_{D \in \mathit{Vals}(x_2, \ldots, x_n)} \mathit{Resp}((x = d) \cup D)$. When hovering over variable x, for each $d \in \mathit{Vals}(x)$, both the total responsibility $\mathit{Resp}(x = d)$ and the responsibility of the individual groups are listed. In addition, the total *influence* of variable x is quantified by computing $\sum_{d \in \mathit{Vals}(x)} |\mathit{Resp}(x = d) - 1/|\mathit{Vals}(x)||$, i.e., a variable has a high influence if its valuations differ from the average significantly. The *influence* is used to determine the intensity of highlighting color, thus drawing the user's attention to variables whose values deserve further inspection.

5 Witnessing Subsystems

Probabilistic model checkers typically do not provide insights into why properties of the form $\Pr^{opt}_{\mathcal{M}}(\Phi) \geq \lambda$, where $opt \in \{\min, \max\}$ and $\lambda \in [0,1]$, are satisfied or violated. Moreover, from a user's perspective it is unclear whether the computed result is actually correct. Inspired by the paradigm of *certifying algorithms*, the work in [25,44] and its extensions [4–6] consider *Farkas certificates* to improve the explainability and trustworthiness of model checking tools. Farkas certificates are solutions to linear constraint systems and serve as correctness proofs that can be used to independently and easily validate the verification result. Notably, Farkas certificates also induce *witnessing subsystems* [42,43,65,66] which are parts of an MDP which on their own already satisfy the property. The following definition is adapted from [25,44].

Definition 3 (Witnessing Subsystem). *Let* $\mathcal{M} = (S, s_{init}, Act, \mathbf{P})$ *be an MDP and* $S' \subseteq S$. *The* subsystem *induced by* S' *is given by* $\mathcal{M}' = (S' \cup \{\bot\}, s_{init}, Act, \mathbf{P}')$ *where for all* $s \in S' \cup \{\bot\}$, $\alpha \in Act(s)$ *and* $s' \in S' \cup \{\bot\}$

$$\mathbf{P}'(s,\alpha,s') = \begin{cases} \mathbf{P}(s,\alpha,s') & \text{if } s \in S', s' \in S', \\ \sum_{t \in S \setminus S'} \mathbf{P}(s,\alpha,t) & \text{if } s \in S', s' = \bot, \\ 0 & \text{otherwise.} \end{cases}$$

A subsystem $\mathcal{M}'$ *is* witnessing *for an objective* Φ *and* $\lambda \in [0,1]$ *if* $\Pr^{opt}_{\mathcal{M}'}(\Phi) \geq \lambda$.

Intuitively, a subsystem $\mathcal{M}'$ of $\mathcal{M}$ is an MDP formed by a subset of states S', where transitions to $S \setminus S'$ are redirected to a fresh state $\bot$ which cannot be left.

It was shown in [25,44] that $\Pr^{opt}_{\mathcal{M}'}(\Phi) \geq \lambda$ implies that $\Pr^{opt}_{\mathcal{M}}(\Phi) \geq \lambda$ holds. Hence, a subsystem $\mathcal{M}'$ can serve as a *witness* for the satisfaction of a property in $\mathcal{M}$. For instance, consider the MDP $\mathcal{M}$ shown in Fig. 7 and the satisfied property $\Pr^{\max}_{\mathcal{M}}(\Diamond s_2) \geq 0.5$. It turns out that the subsystem $\mathcal{M}'$, which is obtained from $\mathcal{M}$ by omitting the transparent states, on its own already satisfies the property, i.e., $\Pr^{\max}_{\mathcal{M}'}(\Diamond s_2) \geq 0.5$. As such it serves as a witness for the constraint $\Pr^{\max}_{\mathcal{M}}(\Diamond s_2) \geq 0.5$.

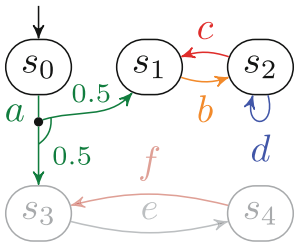


Fig. 7. Toy example from [4].

Motivated by the observation that smaller MDPs are often easier to understand, the work in [4–6,25] developed techniques based on *mixed-integer programming* for computing *minimal* witnesses. Depending on the use case, different minimality criteria can be used, e.g., minimal number of states or state-action pairs. In fact, the subsystem in Fig. 7 is a state-minimal witnessing subsystem for the considered property.

Intuitively, minimal witnessing subsystems isolate the parts in an MDP that are most relevant for the satisfaction (or violation) of a property. In principle,

this enables users to gain a better understanding of the system's behavior. The techniques for computing certificates and minimal witnessing subsystems are implemented in SWITSS [45] and Switss-Multi [4]. The latter is an efficient re-implementation of SWITSS based on Storm [37] and also supports ω-regular and multi-objective properties.

Building on the new data provider interface, we have now integrated Switss-Multi into PMC-VIS, enabling users to directly compute witnessing subsystems within PMC-VIS. This opens up the possibility for the interactive exploration of witnesses and debugging of undesirable behavior. We briefly show the new capabilities on the toy example MDP $\mathcal{M}$ from Fig. 7. The PRISM model of the toy example can be loaded into PMC-VIS, from which we can then start the computation of minimal witnessing subsystems via Switss-Multi. Thanks to the new data provider interface, the computed witness is seamlessly passed to PMC-VIS and can be directly highlighted, as shown in Fig. 8. In particular, the witness can then be interactively inspected and explored. We also emphasize that we can focus subsequent analysis and model refinement tasks on the witness, using any analytical tool that is integrated into PMC-VIS. This provides users with the flexibility and capabilities for debugging undesirable behavior.

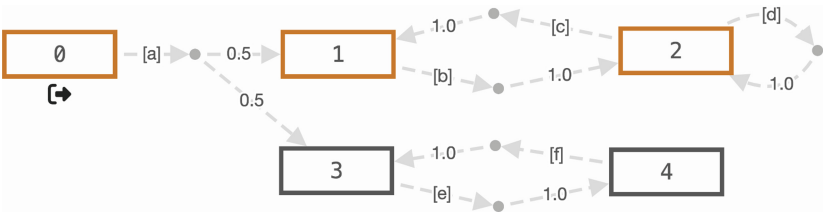


Fig. 8. Visualization of Fig. 7 in PMC-VIS. Orange highlighting shows states that are part of the witnessing subsystem.

6 Medical Lab Case Study

This section showcases how the combination of responsibility values, witnessing subsystems, and visualization in PMC-VIS is used to diagnose a fault that none of the techniques on their own can find.

Model. We model a medical lab that analyses medical samples for the presence of an infection. Both infected and clean samples grow, but infected samples grow faster. Growth is a probabilistic process and the lab uses two methods to assess samples.

- Method 1 lets the sample grow for four hours before assessing its growth. If it exceeds a threshold value, the sample is considered infected.

- Method 2 splits the sample into several smaller sections and lets each section grow independently. After one hour, the number of sections that exceed a threshold is counted. If more than half of the sections exceed the threshold, the sample is considered infected.

Specification. We analyze the maximal probability that the result of analysis does not match the sample, i.e. either a clean sample is classified as infected or an infected sample is classified as clean. After loading the model into PMC-VIS, we can explore the model or immediately trigger model checking.

Running the model checker shows that for the initial state, the failure probability is $\Pr^{\max}(\Diamond \mathit{failure}) = 0.265$. While a non-zero probability of failure is to be expected due to the probabilistic nature of the system, a failure rate of over a quarter is unacceptable.

Interactive Debugging with PMC-VIS. As a first step, we compute responsibility values. The values are added to the PCP. By brushing over the responsibility axis, states with high responsibility are highlighted (both as lines in the PCP and in the graph view). The PCP at this stage is depicted in Fig. 9. However, we find neither an observable pattern in the graph view nor a commonality in the PCP. This is likely because non-determinism occurs in several places of the model and some of it is benign (for example, the number of samples of Method 2 is chosen non-deterministically).

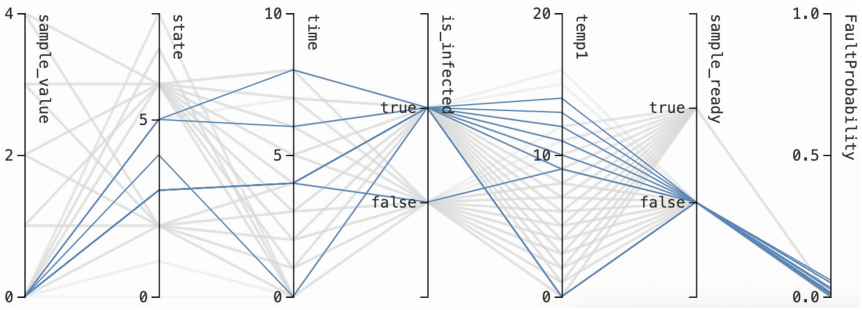


Fig. 9. Parallel coordinates plot of the medical lab model, with states with positive responsibility highlighted. No pattern is immediately apparent.

To cut down the analysis space, we now compute a witnessing subsystem. This calls for a qualitative query, and we select $\Pr^{\max}(\Diamond \mathit{failure}) \geq 0.2$. By choosing 0.2, we require the subsystem to include the states responsible for most of the 0.265 failure rate, while leaving some slack to reduce system size. The witnessing subsystem mainly focuses on Method 1, but contains both executions with an infected and with a clean sample. Nonetheless, the state space remains too large to identify an error.

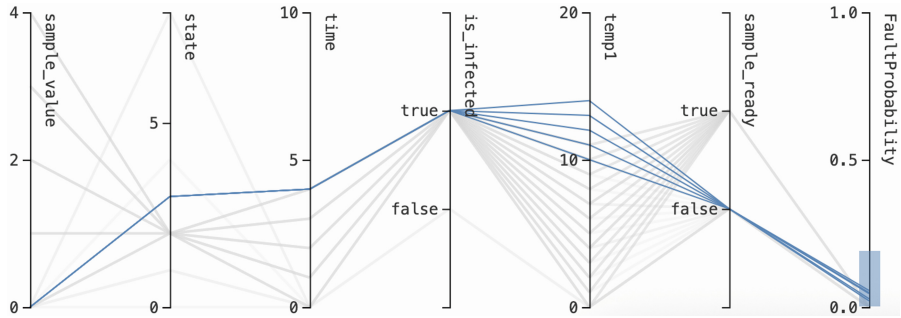


Fig. 10. Parallel coordinates plot of the witnessing subsystem of the medical lab model. States with positive responsibility values are highlighted. Unlike in Fig. 9, a clear pattern emerges, as the responsible states only differ in the value of `temp1` and share the other values.

```
26    [ProvideSample] (state=0) -> (state'=1);
27
28    [] (state=1) -> 0.5: (state'=2) + 0.5: (state'=4);
29
30    [Sample] (state=2 & time<4) -> true;
31    [TakeSample] (state=2 & time<4) -> (time'=time+1)&(temp1'=min(temp1max, temp1+sample_value));
32    [] (state=2 & time=4 & !sample_ready) -> (state'=3);
33
34    [] (state=3 & temp1 < 15) -> (state'=8)&(analysis_result'=false)&(temp1'=0)&(time'=0);
35    [] (state=3 & temp1 >= 9) -> (state'=8)&(analysis_result'=true)&(temp1'=0)&(time'=0);
36
37    // Non-deterministically choose sampling duration
38    // To compensate for different sampling durations, we add an initial count to temp1
39    [] (state=4) -> (state'=5)&(time'=6);
40    [] (state=4) -> (state'=5)&(time'=7);
41    [] (state=4) -> (state'=5)&(time'=8);
```

Fig. 11. Excerpt from the code specifying the medical lab, with the highlighting indicating which transitions are active (green) and blocked (red) in the erroneous states. The first active transition has an incorrect guard: It should be `state=3 & temp1` (Color figure online) < 9

We now open the witnessing subsystem in a new tab. We expand the state space and compute responsibility values again, this time restricted to the subsystem. The result is shown in Fig. 10. States with positive responsibility are highlighted in blue. This selection is automatically applied to the graph view as well (but this is not depicted in Fig. 10). The highlighted states share `state` $= 3$, which is the second stage of Method 1. Switching over to the code editor highlights the two commands enabled in this state (cf. Figure 11). We quickly spot the mistake: The thresholds are incorrect: For the first command, it should be `temp1` < 9, not `temp1` < 15. With the incorrect threshold, for $9 \leq$ `temp1` < 15, the machine non-deterministically chooses the result (and because we maximize the error, it always chooses incorrectly). Fixing the error results in a reduced error probability of 0.033, which is acceptable given the probabilistic nature of the processes involved.

7 Conclusion

PMC-VIS has grown from a single-source visualization tool for Prism into a model development environment, integrating rich code editing features, scalable visualization approaches, and extensions that allow it to import data from various model checkers and analysis tools, as discussed in this paper. The case study illustrates how combining responsibility values with witnessing subsystems and visualizing both in PMC-VIS helps locate errors. More extensive studies are required to assess how broadly these techniques are applicable to model repair.

While the integration of these tools into PMC-VIS demonstrates its extensibility, much work remains in extending our support for tools that target not states or state-action pairs, but, e.g., specification modules, clusters, or other model abstractions. In future work, we will additionally explore facilities for model (and model version) comparison and compositional (i.e., module-based) exploration, with which we aim to further develop and enhance model explainability and development workflows.

Acknowledgments. Authors are ordered alphabetically. This work was funded by the German Research Foundation (DFG) under Germany's Excellence Strategy: EXC 2050/2, 390696704 – Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of TU Dresden; by DFG grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science); by the German Federal Ministry of Research, Technology and Space (BMFTR) within the project SEMECO Q1 (03ZU1210AG); within the DAAD project 57616814 (SECAI, School of Embedded and Composite AI) as part of the program Konrad Zuse Schools of Excellence in Artificial Intelligence; and by the Federal Ministry of Research, Technology and Space (BMFTR, SCADS22B) and Saxon State Ministry for Science, Culture and Tourism (SMWK) by funding the competence center for Big Data and AI "ScaDS.AI Dresden/Leipzig".

References

1. Abate, A., Giacobbe, M., Roy, D.: Quantitative supermartingale certificates. In: Piskac, R., Rakamaric, Z. (eds.) Computer Aided Verification, pp. 3–28. LNCS, Springer (2025). https://doi.org/10.1007/978-3-031-98679-6_1
2. Aljazzar, H., Leue, S.: Generation of counterexamples for model checking of Markov decision processes. In: QEST, pp. 197–206. IEEE Computer Society (2009). https://doi.org/10.1109/QEST.2009.10
3. Baier, C., van den Bossche, R., Klüppelholz, S., Lehmann, J., Piribauer, J.: Backward responsibility in transition systems using general power indices. In: Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, February 20-27, 2024, Vancouver, Canada. pp. 20320–20327. AAAI Press (2024). https://doi.org/10.1609/AAAI.V38I18.30013
4. Baier, C., Chau, C., Drobitko, V., Jantsch, S., Klüppelholz, S.: Certificates and witnesses for multi-objective omega-regular queries in Markov decision processes. In: Bianculli, D., Gómez-Martínez, E. (eds.) Software Engineering and Formal Methods - 23rd International Conference, SEFM 2025, Toledo, Spain, November 10-14, 2025, Proceedings, pp. 129–147. Lecture Notes in Computer Science, Springer (2025). https://doi.org/10.1007/978-3-032-10444-1_8

5. Baier, C., Chau, C., Klüppelholz, S.: Certificates and witnesses for multi-objective queries in Markov decision processes. In: Hillston, J., Soudjani, S., Waga, M. (eds.) Quantitative Evaluation of Systems and Formal Modeling and Analysis of Timed Systems - First International Joint Conference, QEST+FORMATS 2024, Calgary, AB, Canada, September 9-13, 2024, Proceedings. LNCS, vol. 14996, pp. 1–18. Springer (2024). https://doi.org/10.1007/978-3-031-68416-6_1
6. Baier, C., Chau, C., Klüppelholz, S.: Certificates and witnesses for multi-objective queries in Markov decision processes. Perf. Eval. (PEVA) **168**, 102482 (2025). https://doi.org/10.1016/j.peva.2025.102482, journal Version of QEST'24 paper
7. Baier, C., Funke, F., Majumdar, R.: A game-theoretic account of responsibility allocation. In: Zhou, Z.H. (ed.) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, pp. 1773–1779. International Joint Conferences on Artificial Intelligence Organization (2021). https://doi.org/10.24963/ijcai.2021/244
8. Baier, C., Katoen, J.: Principles of model checking, MIT Press (2008)
9. Baier, C., Klatt, R., Klüppelholz, S., Lehmann, J.: Backward responsibility in transition systems beyond safety. In: Remke, A., Steffen, B. (eds.) Formal Methods for Industrial Critical Systems - 30th International Conference, FMICS 2025, Aarhus, Denmark, August 27-28, 2025, Proceedings. LNCS, vol. 16040, pp. 105–123. Springer (2025). https://doi.org/10.1007/978-3-032-00942-5_6
10. Baier, C., Klüppelholz, S., Lehmann, J.: Responsibility in actor-based systems. In: Lee, E.A., Mousavi, M.R., Talcott, C.L. (eds.) Rebeca for Actor Analysis in Action - Essays Dedicated to Marjan Sirjani on the Occasion of Her 60th Birthday. LNCS, vol. 15560, pp. 44–69. Springer (2025).https://doi.org/10.1007/978-3-031-85134-6_3
11. Baier, C., Piribauer, J., Ziemek, R.: Foundations of probability-raising causality in Markov decision processes. Log. Meth. Comput. Sci. LMCS **20**(1) (2024). https://doi.org/10.46298/lmcs-20(1:4)2024, journal Version of FoSSaCS'22 paper
12. Beer, I., Ben-David, S., Chockler, H., Orni, A., Trefler, R.J.: Explaining counterexamples using causality. Form. Meth. Syst. Des. **40**(1), 20–40 (2012). https://doi.org/10.1007/S10703-011-0132-2
13. Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: UPPAAL – a tool suite for automatic verification of real-time systems. In: Hybrid Systems III, pp. 232–243. LNCS, Springer (1996). https://doi.org/10.1007/BFb0020949
14. Bunte, O., et al.: The mCRL2 toolset for analysing concurrent systems. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 21–39. LNCS, Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-17465-1_2
15. Caltais, G., Guetlein, S.L., Leue, S.: Causality for general LTL-definable properties. In: Finkbeiner, B., Kleinberg, S. (eds.) Proceedings 3rd Workshop on formal reasoning about Causation, Responsibility, and Explanations in Science and Technology, CREST@ETAPS 2018, Thessaloniki, Greece, 21st April 2018. EPTCS, vol. 286, pp. 1–15 (2018). https://doi.org/10.4204/EPTCS.286.1
16. Cavada, R., al.: The nuXmv symbolic model checker. In: Computer Aided Verification (CAV), pp. 334–342 (2014). https://doi.org/10.1007/978-3-319-08867-9_22
17. Chatterjee, K., Quatmann, T., Schäffeler, M., Weininger, M., Winkler, T., Zilken, D.: Fixed point certificates for reachability and expected rewards in MDPs. In: Gurfinkel, A., Heule, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 15697, pp. 130–151. Springer (2025). https://doi.org/10.1007/978-3-031-90653-4_7

18. Chockler, H., Halpern, J.Y.: Responsibility and blame: a structural-model approach. J. Artif. Intell. Res. **22**, 93–115 (2004). https://doi.org/10.1613/JAIR.1391
19. Chockler, H., Halpern, J.Y., Kupferman, O.: What causes a system to satisfy a specification? ACM Trans. Comput. Log. **9**(3), 20:1–20:26 (2008). https://doi.org/10.1145/1352582.1352588
20. Cimatti, A., et al.: NuSMV 2: an opensource tool for symbolic model checking. In: Computer Aided Verification (CAV), pp. 359–364 (2002). https://doi.org/10.1007/3-540-45657-0_29
21. Dai, J., Cheng, J.: HMMEditor: a visual editing tool for profile hidden Markov model. BMC Genom. **9**(1), S8 (2008). https://doi.org/10.1186/1471-2164-9-S1-S8
22. Dubey, P., Shapley, L.S.: Mathematical properties of the Banzhaf power index. Math. Oper. Res. **4**(2), 99–131 (1979). https://doi.org/10.1287/MOOR.4.2.99
23. Ellson, J., Gansner, E.R., Koutsofios, E., North, S.C., Woodhull, G.: Graphviz and Dynagraph — Static and Dynamic Graph Drawing Tools, pp. 127–148. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-642-18638-7_6
24. Fijalkow, N., et al.: Games on graphs: from logic and automata to algorithms (2025). https://arxiv.org/abs/2305.10546
25. Funke, F., Jantsch, S., Baier, C.: Farkas certificates and minimal witnesses for probabilistic reachability constraints. In: Biere, A., Parker, D. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, pp. 324–345. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-45190-5_18
26. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. STTT **15**(2), 89–107 (2013). https://doi.org/10.1007/s10009-012-0244-z
27. Goldsby, H., Cheng, B.H.C., Konrad, S., Kamdoum, S.: A visualization framework for the modeling and formal analysis of high assurance systems. In: International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 707–721. LNCS, Springer (2006). https://doi.org/10.1007/11880240_49
28. Groce, A., Chaki, S., Kroening, D., Strichman, O.: Error explanation with distance metrics. Int. J. Softw. Tools Technol. Transf. **8**(3), 229–247 (2006). https://doi.org/10.1007/S10009-005-0202-0
29. Gros, T.P., Groß, D., Gumhold, S., Hoffmann, J., Klauck, M., Steinmetz, M.: TraceVis: towards visualization for deep statistical model checking. In: International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2020), pp. 27–46. LNCS, Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-83723-5_3
30. Gros, T.P., Groß, D., Kamp, J., Gumhold, S., Hoffmann, J.: Visual analysis of action policy behavior: a case study in grid-world driving. In: Biecek, P., Nowaczyk, S., Kutyniok, G., Longo, L. (eds.) xAI-2025 Late-breaking Work, Demos and Doctoral Consortium Joint Proceedings, pp. 145–152. No. 4017 in CEUR Workshop Proceedings, Aachen (2025). https://ceur-ws.org/Vol-4017/paper_19.pdf
31. Halpern, J.Y., Pearl, J.: Causes and explanations: a structural-model approach. part i: causes. British J. Philos. Sci. (2005). https://doi.org/10.1093/bjps/axi147
32. van Ham, F., Perer, A.: "search, show context, expand on demand": supporting large graph exploration with degree-of-interest. IEEE TVCG **15**(6), 953–960 (2009). https://doi.org/10.1109/TVCG.2009.108
33. van Ham, F., van de Wetering, H., van Wijk, J.: Visualization of state transition graphs. In: IEEE Symposium on Information Visualization (InfoVis 2001), pp. 59–66 (2001). https://doi.org/10.1109/INFVIS.2001.963281

34. van Ham, F., van de Wetering, H., van Wijk, J.: Interactive visualization of state transition systems. IEEE Trans. Visual. Comput. Graph. **8**(4), 319–329 (2002). 10/d7jxz2
35. Han, T.: Diagnosis, Synthesis and Analysis of Probabilistic Models. Ph.D. thesis, RWTH Aachen University (2009). http://darwin.bth.rwth-aachen.de/opus3/volltexte/2010/3139/
36. Han, T., Katoen, J.P., Berteun, D.: Counterexample generation in probabilistic model checking. IEEE Trans. Softw. Eng. **35**(2), 241–257 (2009). https://doi.org/10.1109/TSE.2009.5
37. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The probabilistic model checker Storm. Int. J. Softw. Tools Technol. Transfer **24**(4), 589–610 (2022). https://doi.org/10.1007/S10009-021-00633-Z
38. Henzinger, T.A., Mallik, K., Sadeghi, P., Žikelić, Đ.: Supermartingale certificates for quantitative omega-regular verification and control. In: Piskac, R., Rakamaric, Z. (eds.) Computer Aided Verification. LNCS, vol. 15932, pp. 29–55. Springer (2025). https://doi.org/10.1007/978-3-031-98679-6_2
39. Holzmann, G.J.: The model checker SPIN. IEEE Trans. Softw. Eng. **23**(5), 279–295 (1997). https://doi.org/10.1109/32.588521
40. Horak, T., et al.: Visual analysis of hyperproperties for understanding model checking results. IEEE Trans. Visual. Comput. Graph. **28**(1), 357–367 (2022). https://doi.org/10.1109/TVCG.2021.3114866
41. Inselberg, A.: Parallel Coordinates, Springer, New York (2009). 10/d8krgt
42. Jansen, N.: Counterexamples in Probabilistic Verification. Ph.D. Thesis, RWTH Aachen University, Germany (2015). http://publications.rwth-aachen.de/record/479827
43. Jansen, N., Ábrahám, E., Katelaan, J., Wimmer, R., Katoen, J., Becker, B.: Hierarchical counterexamples for discrete-time Markov chains. In: Bultan, T., Hsiung, P. (eds.) Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings, pp. 443–452. LNCS, Springer (2011). https://doi.org/10.1007/978-3-642-24372-1_33
44. Jantsch, S.: Certificates and Witnesses for Probabilistic Model Checking. Ph.D. Thesis, Dresden University of Technology, Germany (2022). https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-804732
45. Jantsch, S., Harder, H., Funke, F., Baier, C.: Switss: computing small witnessing subsystems. In: 2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020, pp. 236–244. IEEE (2020). https://doi.org/10.34727/2020/ISBN.978-3-85448-042-6_31
46. Kamhi, G., Fix, L., Binyamini, Z.: Symbolic model checking visualization. In: Formal Methods in Computer-Aided Design (FMCAD), pp. 290–302. LNCS, Springer (1998). https://doi.org/10.1007/3-540-49519-3_19
47. Kleinberg, S., Mishra, B.: The temporal logic of token causes. In: Lin, F., Sattler, U., Truszczynski, M. (eds.) Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR'10). AAAI Press (2010). https://aaai.org/papers/67-1301-the-temporal-logic-of-token-causes/
48. Kobialka, P., Gerlach, L., Leofante, F., Ábrahám, E., Tapia Tarifa, S.L., Johnsen, E.B.: Counterfactual strategies for Markov decision processes. In: Kwok, J. (ed.) Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25, pp. 412–420. International Joint Conferences on Artificial Intelligence Organization (2025). https://doi.org/10.24963/ijcai.2025/47

49. Kölbl, M., Leue, S., Schmid, R.: Dynamic causes for the violation of timed reachability properties. In: Bertrand, N., Jansen, N. (eds.) Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020, Proceedings. LNCS, vol. 12288, pp. 127–143. Springer (2020). https://doi.org/10.1007/978-3-030-57628-8_8
50. Korn, M.: Formal-Methods Support for Runtime Adaptation in Self-Adaptive Systems. Ph.D. thesis, Technische Universität Dresden, Germany (2025). https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-953278
51. Korn, M., Méndez, J., Klüppelholz, S., Langner, R., Baier, C., Dachselt, R.: PMC-vis: an interactive visualization tool for probabilistic model checking. In: Software Engineering and Formal Methods. LNCS, vol. 14323. Springer (2023). https://doi.org/10.1007/978-3-031-47115-5_20
52. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: CAV '11, pp. 585–591. LNCS, Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_47
53. Kwon, B.C., et al.: DPVis: visual analytics with hidden Markov models for disease progression pathways. IEEE Trans. Vis. Comput. Graph. **27**(9), 3685–3700 (2021). https://doi.org/10.1109/TVCG.2020.2985689
54. Leitner-Fischer, F., Leue, S.: Causality checking for complex system models. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings. LNCS, vol. 7737, pp. 248–267. Springer (2013). https://doi.org/10.1007/978-3-642-35873-9_16
55. Li, M., Sun, H., Huang, Y., Chen, H.: Shapley value: from cooperative game to explainable artificial intelligence. Auton. Intell. Syst. **4**(1), 2 (2024). https://doi.org/10.1007/S43684-023-00060-8
56. Mascle, C., Baier, C., Funke, F., Jantsch, S., Kiefer, S.: Responsibility and verification: importance value in temporal logics. In: 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021, pp. 1–14. IEEE (2021). https://doi.org/10.1109/LICS52264.2021.9470597
57. McGregor, S., Buckingham, H., Dietterich, T.G., Houtman, R., Montgomery, C., Metoyer, R.: Interactive visualization for testing Markov decision processes: MDPVis. J. Vis. Lang. Comput. **39**, 93–106 (2017). https://doi.org/10.1016/j.jvlc.2016.10.007, special Issue on Programming and Modelling Tools
58. Parreaux, J., Piribauer, J., Baier, C.: Counterfactual causality for reachability and safety based on distance functions. In: Proceedings of the Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2023, Udine, Italy, 18-20th September 2023. EPTCS, vol. 390, pp. 132–149 (2023). https://doi.org/10.4204/EPTCS.390.9
59. Van de Poel, I.: The relation between forward-looking and backward-looking responsibility. In: Moral responsibility: Beyond free will and determinism, pp. 37–52. Springer (2011). https://doi.org/10.1007/978-94-007-1878-4_3
60. Pretorius, A.J., van Wijk, J.: Multidimensional visualization of transition systems. In: 9th International Conference on Information Visualisation (IV 2005), pp. 323–328 (2005). 10/cnzm2g
61. Pretorius, A.J., van Wijk, J.J.: Bridging the semantic gap: visualizing transition graphs with user-defined diagrams. IEEE Comput. Graph. Appl. **27**(5), 58–66 (2007). 10/dpj2rv
62. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994). https://doi.org/10.1002/9780470316887

63. Shapley, L.S.: A value for N-person games. In: Contribution to the Theory of Games, vol. 2 (1953)
64. Takisaka, T., Oyabu, Y., Urabe, N., Hasuo, I.: Ranking and repulsing supermartingales for reachability in probabilistic programs. In: Lahiri, S.K., Wang, C. (eds.) Automated Technology for Verification and Analysis. Lecture Notes in Computer Science, vol. 11138, pp. 476–493. Springer (2018). https://doi.org/10.1007/978-3-030-01090-4_28
65. Wimmer, R., Jansen, N., Ábrahám, E., Becker, B., Katoen, J.P.: Minimal critical subsystems for discrete-time Markov models. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 299–314. Springer Berlin Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_21
66. Wimmer, R., Jansen, N., Ábrahám, E., Katoen, J., Becker, B.: Minimal counter examples for linear-time probabilistic verification. Theoret. Comput. Sci. **549**, 61–100 (2014). https://doi.org/10.1016/j.tcs.2014.06.020
67. Winter, E.: The Shapley value. Handbook of game theory with economic applications **3**, 2025–2054 (2002)
68. Žikelić, Đ., Lechner, M., Henzinger, T.A., Chatterjee, K.: Learning control policies for stochastic systems with reach-avoid guarantees. In: Williams, B., Chen, Y., Neville, J. (eds.) Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023, pp. 11926–11935. AAAI Press (2023). https://doi.org/10.1609/AAAI.V37I10.26407

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

