Technische Universität Dresden | Faculty of Computer Science

Data Preperation for PMC-Visualization

Bachelor's thesis for obtaining the first university degree

Bachelor of Science (B.Sc.)

submitted by

Franz Martin SCHMIDT

(born on 7. April 1999 in Halle (Saale))

Date of Submission: August 27, 2023

Dipl. Inf Max Korn (Theoretical Computer Science)

Abstract

In today's world, our dependence on information and communications technology (ICT) is profound. Ensuring fault-free behavior is crucial. Model checking verifies systems, but faces challenges due to the state explosion problem, which affects both algorithms and human comprehension. This thesis introduces the concept of *view*, which preprocesses data for visualization to enhance human understanding. We formalize and define views for Markov decision processes (MDPs), present operations and types, and propose example views using MDP features as well as the graph structure of MDPs. An implementation is provided within the project PMC-Vis.

Statement of Authorship

I hereby declare that I have written this thesis independently and have listed all used sources and aids. I am submitting this thesis for the first time as part of an examination. I understand that attempted deceit will result in the failing grade „not sufficient“ (5.0).

Matriculation Number: 4793164

Franz Martin Schmidt
Dresden, August 27, 2023

Acknowledgements

I would like to express my sincere gratitude to Dipl. Ing. Max Korn for for his thoughtful discussions and his invaluable guidance throughout the implementation process. His technical expertise and willingness to share insights have truly helped shaping this work.

I extend my gratitude to Dr. Dipl. Ing. Sascha Klüppelholz for his contribution in refining my ideas and providing valuable assistance with the formal aspects of this thesis. His attention to detail and commitment to academic standards were highly valuable.

Contents

1 Introduction

The modern world relies heavily on information and communication technology (ICT) systems. They are found in devices used everyday like smartphones or laptops or in distributed systems such as the infrastructure sustaining the internet, but also in life-saving ones utilized in medicine. In addition to performance, features, and functionality, one of the most important aspects of such systems is their fault-free behavior during runtime. The impact of a system failure can range from a small disturbance in the user experience to the inoperability of the entire system, which could result in an annoyed user, millions of dollars in financial damage, or a lost life. To prevent these potentially severe negative effects, methods are needed to verify the correct behavior of a system.

In addition to commonly used approaches such as peer review and testing for software and emulation and simulation for hardware, formal methods can also be used to verify the correct behavior of a system. These can be based on models, where a state in the model refers to a possible state of the system. Models describe the possible system behavior in a mathematically precise and unambiguous manner and can be used for verification in different ways: the state space can be explored exhaustively, only certain scenarios can be considered, or they are tested in reality.

Model checking describes the approach of exhaustive exploration of the state space. It is a formal model-based method for system verification, that checks in an automated manner if a given property holds for every state of a given finite-state model. [1, chs. 1.1 and 1.2]. Probabilistic model checking allows not only nondeterministic transitions between states in the models checked, but also probabilistic ones. It enables to properly model and check systems in which both controllable, as well as stochastic phenomena are occurring. The main limitation for algorithms that run on the set of states of models to model check them is the state explosion issue. The state explosion issue states that the number of states grows exponentially with the number of variables in a program graph or with the number of components in concurrent systems [1, ch. 2.3]. Even simple system models can lead to complex system behavior and an immense number of states. This is not only a issue for algorithms, but also for humans who need to understand, analyze, and verify models in the context of model checking. An interactive visualization can assist with this issue and even be made use of to achieve further goals such as debugging or model repair.

There are techniques in visualization for large multivariate graphs (networks where nodes and relationships have attributes), such as MDPs. Aggregation and clustering methods are used in visualization for *large* graphs [2]. There also exist methods for visualizing *multivariate* graphs [3,4]. One approach for visualizing multivariate graphs are multiple coordinated view setups featuring parallel coordinate plots (PCPs) [5]. However, even these techniques in visualization have their limitations for very large amounts of data as they are easily reachable with models used in model checking. The pure data volume remains unaffected. One could address this issue by reducing the amount of data that is passed to the visualization. This is the goal of this thesis: to preprocess the data that is then used in the visualization.

Preprocessing is a term that describes an approach mostly used in data mining

and machine learning. Common problems are too much data, too little data, and imperfect data (noise, incompleteness) [6]. Preprocessing addresses these issue. As the state explosion problem causes an immense amount of states, approaches that reduce the amount of data are of interest. Methods reducing data can refer to the size of single samples (feature selection, space transformations) or the amount of samples (instance reduction). There are several methods to reduce the number of instances. The approach that will be explored in this thesis can be classified as instance reduction and can be considered as a variant of clustering, although in literature clustering can describe slightly different notions [7, 8].

Whereas these approaches are general in the sense that they concern arbitrary datatypes, the representations of MDPs are graphs, which have been heavily studied in the last decades. In consequence, there exist many approaches to simplifying graphs. There are simplifications based on the pure mathematical object, without any domain-specific context, such as connectivity [9], patterns [10], modularity [11] or cuts [2, 12]. There also exist methods specific to certain domains [13–15]. In this thesis we want to explore approaches specific to the domain of model checking. Approaches found in literature aim at preserving certain logic formulae or the ability to perform proper verification on the simplified graph [16–18]. An interesting model checking specific approach is based on equivalence or order relations on the set of states. States are in relation if they can simulate each other step by step or in several steps with respect to atomic propositions. This causes preservation of certain logical formulae used in model checking, but conversely other information is lost. In addition, the computation of these relations is rather costly. We will introduce and provide an implementation of a concept called view, which is similar to abstraction [1, p. 499]. It defines an equivalence relation on the set of states that share certain properties. Its intention is to show humans interacting with the visualization as much information as possible in a compact and concise form, rather than preserving logical formulae.

After giving some fundamentals about the systems where the concept view may be applied in chapter 2, we will formalize views, discuss types and operations on and with them in chapter 3. In chapter 4 we will give examples of views utilizing MDP characteristics and views utilizing the structure of the MDP graph, but for this bachelor thesis limit the scope of views considered to those, that do not take advantage of probabilities in MDPs. In chapter 5 we will elaborate where and how the views proposed in chapter 4 have been implemented. Lastly the views will be evaluated in chapter 6 by considering three use cases how views can be applied and used. An overview of the performance and scalability of the proposed views is also given.

2 Preliminaries

Views will be defined on MDPs. Instead of providing the definition of an MDP directly, we will consider less powerful classes of models to represent systems extended by MDPs.

First, we will consider *transition systems*. Transition systems are basically digraphs consisting of *states* and *transitions* in place of nodes and edges. A state describes some information about a system at a certain moment of its behavior, a transition the evolution from one state to another. We will use transition systems with *action names* and *atomic propositions* for states as in [1].

Definition 2.1 ([1], Definition 2.1.)**.** A *transition system* TS is a tuple $(S, Act, \longrightarrow, I, AP, L)$ where

- S is a set of states,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times Act \times S$ is transition relation,
- $I \subseteq S$ is a set of initial states,
- AP is a set of atomic propositions, and
- $L : S \to \mathcal{P}(AP)$

A transition system is called *finite* if S, AP, and L are finite. Actions are used for communication between processes. We denote them with Greek letters $(\alpha, \beta, \gamma, \dots)$. Atomic propositions are simple facts about states. For instance "x is greater than 20" or "red and yellow light are on" could be atomic propositions. We will denote atomic propositions with Arabic letters $(a, b, c, \dots)$. They are assigned to a state by a labeling function L.

The intuitive behavior of transition systems is as follows. The evolution of a transition system starts in some state $s \in I$. If the set of I of initial states is empty, the transition system has no behavior at all. From the initial state the transition system evolves according to the transition relation $\longrightarrow$. The evolution ends in a state that has no outgoing transitions. For each state there may be several possible transitions. The choice of which one to take is made nondeterministically. That is, the outcome of the selection can not be known a priori. In particular, it does not follow any probability distribution. Hence, no statement can be made about the likelihood of a transition being selected.

This is in contrast to *Markov Chains*, where the nondeterministic behavior is replaced by a probabilistic one. That is, for each state there exists a probability distribution describing the chance of a transition being selected. There are no actions and no nondeterminism in Markov Chains.

Definition 2.2 ([1], Definition 10.1.)**.** A *(discrete-time) Markov Chain* is a tuple $\mathcal{M} = (S, \mathbf{P}, \iota_{init}, AP, L)$ where

- S is a countable, nonempty set of states,

- $\mathbf{P} : S \times S \to [0,1]$ is the *transition probability function*, such that for all states s:

$$\sum_{s' \in S} \mathbf{P}(s, S') = 1.$$

- $\iota_{init} : S \to [0,1]$ is the *initial distribution*, such that $\sum_{s \in S} \iota_{init}(s) = 1$, and
- AP is a set of atomic propositions and,
- $L : S \to \mathcal{P}(AP)$ a labeling function.

A Markov Chain (MC) $\mathcal{M}$ is called *finite* if S and AP are finite. The probability function $\mathbf{P}$ specifies for each state s the probability $\mathbf{P}(s, s')$ of moving from s to s' in one step. The constraint put on $\mathbf{P}$ in the second item ensures that $\mathbf{P}$ is a probability distribution. The value $\iota_{init}(s)$ specifies the probability that the system evolution will start in s. All states s with $\iota_{init}(s) > 0$ are considered *initial states*. States s' with $\mathbf{P}(s, s') > 0$ are considered possible successors of state s. The operational behavior is as follows. The initial distribution ι_{init} yields a state s_0. Afterwards, in each state a transition is yielded at random according to the probability distribution $\mathbf{P}$ in that state.

The disadvantage of Markov Chains is that they do not enable process intercommunication and do not permit nondeterminism, but only probabilistic evolution. *Markov Decision Processes* (MDPs) allow both probabilistic and nondeterministic choices. An MDP is thus a model that somewhat merges the concept of transition systems with the concept of Markov Chains.

Definition 2.3 ([1], Definition 10.81.)**.** A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ where

- S is a countable set of states,
- Act is a set of actions,
- $\mathbf{P} : S \times Act \times S \to [0,1]$ is the transition probability function such that for all states $s \in S$ and actions $\alpha \in Act$:

$$\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{0, 1\},$$

- $\iota_{init} : S \to [0,1]$ is the initial distribution such that $\sum_{s \in S} \iota_{init}(s) = 1$,
- AP is a set of atomic propositions and
- $L : S \to \mathcal{P}(AP)$ a labeling function.

An action α is *enabled* in state s if and only if $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$. Let $Act(s)$ denote the set of enabled actions in s. For any state $s \in S$, it is required that $Act(s) \neq \emptyset$. Each state s' for which $\mathbf{P}(s, \alpha, s') > 0$ is called an α-*successor* of s.

An MDP is called *finite* if S, Act and AP are finite. The transition probabilities $\mathbf{P}(s, \alpha, t)$ can be arbitrary real numbers in $[0, 1]$ (that sum up to 1 or 0 for fixed s and α). For algorithmic purposes, they are assumed to be rational. The unique initial distribution ι_{init} could be generalized to a set of ι_{init} with a nondeterministic choice at the beginning. For simplicity, there is only one distribution. The operational behavior is as follows. A starting state s_0 is yielded by ι_{init} with $\iota_{init}(s_0) > 0$. From there, first a nondeterministic choice of an enabled action takes place followed by a probabilistic choice of a state. The action is fixed with the step of nondeterministic choice. Every Markov Chain is an MDP in which for every state s, $Act(s)$ is a singleton set. Conversely an MDP with the property of $\overrightarrow{Act}(s) = 1$ is a Markov Chain. Thus Markov Chains are a proper subset of Markov decision processes.

For convenience, we write $(s_1, \alpha, s_2) \in \mathbf{P}$ if $\mathbf{P}(s_1, \alpha, s_2) > 0$, where $s_1, s_2 \in S$ and $\alpha \in Act$. That is, we write $(s_1, \alpha, s_2) \in \mathbf{P}$ if there is a non-zero probability of evolving from state s_1 to state s_2 with action α. We define $\overrightarrow{Act}(s) := \{\alpha \mid (s, \alpha, \tilde{s}) \in \mathbf{P}\}$ for $s \in S$. For $s \in S$, we call an element of $\overrightarrow{Act}(s)$ *outgoing action* of s. We say that a state s "has an outgoing action α " or "α is outgoing from s " if $\alpha \in \overrightarrow{Act}(s)$. We use analogous definition and terminology for incoming actions $\overleftarrow{\alpha}$.

Example 2.1. In Figure 1 we see an graphical representation of an MDP. For this MDP it is $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$, $Act = \{\alpha, \beta, \gamma\}$, $AP = \{a, b\}$. In Table 1 we see the values for L, ι_{init} and $\mathbf{P}$.

State	$L(s_i)$	$\iota_{init}(s_i)$
s_1	$\emptyset$	0.7
s_2	$\{a\}$	0
s_3	$\{a, b\}$	0.2
s_4	$\{b\}$	0.1
s_5	$\{a\}$	0
s_6	$\{b\}$	0

$t \in S \times Act \times S$	$\mathbf{P}(t)$
(s_1, α, s_2)	0.5
(s_1, α, s_4)	1
(s_2, β, s_2)	0.6
(s_2, β, s_6)	0.4
(s_2, γ, s_1)	1
(s_3, β, s_3)	0.4
(s_3, β, s_2)	0.6
(s_3, γ, s_1)	0.8
(s_3, γ, s_4)	0.2
(s_4, α, s_4)	0.5
(s_4, γ, s_4)	1
(s_4, α, s_3)	0.5
(s_5, β, s_5)	1
(s_6, α, s_6)	1

Table 1: Labeling Function, initial distribution (left) and transition probability function (right) of the MDP in Figure 1. Omitted values of $\mathbf{P}(t)$ are meant to be zero.

We will declare MDPs by only providing its graphical representation. For this the sets S, Act, AP are assumed to be minimal. That is, they contain no more elements than displayed in the graphical representation. Mostly we will use simplified graphical representations of MDPs. In these we will omit information. If actions

are omitted, it is assumed that each transition $t \in \mathbf{P}$ has a distinct action. If probabilities are omitted for each state it is assumed the uniform distribution on each set of outgoing transitions with the same action. If the initial distribution is omitted, it is assumed to be the uniform distribution. If atomic propositions are omitted it is assumed that the set of atomic propositions is empty and every state is mapped to the empty set by the labeling function. The purpose of simplified representations is to focus on and only show relevant information. The remaining information is considered irrelevant in these cases.

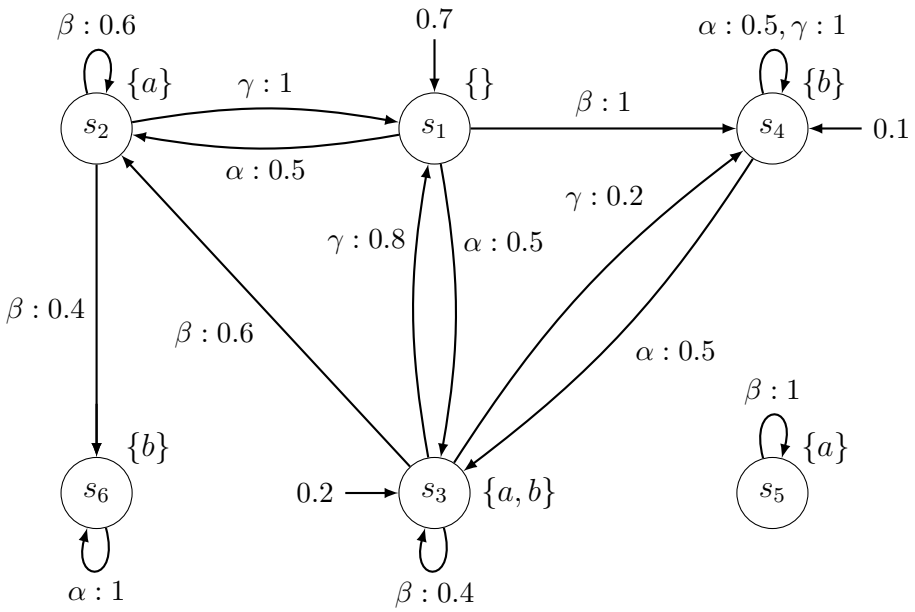


Figure 1: Simplified representation of $\mathcal{M}$ (left) and the view ${\mathcal{M}_I}^\top$ on it(left)

In the implementation and evaluation chapters 5 and 6 we will refer to PRISM (PRobabilistIc Symbolic Model checker), which is why we will give a brief introduction on it. PRISM is a model checker that can be used to define models and check them in an automated manner. In PRISM, a model is defined with modules that interact with each other. A module consists of variables and commands. The (current) values of all variables in the module define the (current) local state of the module. The global state of the model is defined by the local state of all modules. A command in a module is of the form

```
[action] guard -> p_1 : update_1 + ... + p_m : update_m;
```

where `p_1, ... , p_m > 0` and `p_1 + ... + p_m = 1`. The `guard` is a predicate on all variables of the model (including those of other modules). It may include operators such as negation (`!`), conjunction (`&`) disjunction (`|`), arithmetic operators (`+`, `-`, `*`, `/`) or relational operators (`<`, `<=`, `>=`, `>`, `!=`, `=`, `=>`, `<=>`) as well as predefined functions. An `update` represents a transition in the model, which normally reflects a change of state. As a state is defined by the values of all the variables, an update is specified by the assignment of new values to the variables of the module, possibly as a function of variables from other modules. The value of a variable remains

```
mdp

module onlymodule

x : [0..6] init 0;

[]  x=0 -> 0.7:(x'=1) + 0.2:(x'=3) + 0.1:(x'=4);

[a] x=1 -> 0.5:(x'=2) + 0.5:(x'=3);
[b] x=1 -> (x'=4);
[b] x=2 -> 0.6:(x'=2) + 0.4:(x'=6);
[c] x=2 -> (x'=1);
[b] x=3 -> 0.4:(x'=3) + 0.6:(x'=2);
[c] x=3 -> 0.8:(x'=1) + 0.2:(x'=4);
[a] x=4 -> 0.5:(x'=4) + 0.5:(x'=3);
[c] x=4 -> (x'=4);
[b] x=5 -> (x'=5);
[a] x=6 -> (x'=6);

endmodule
```

Listing 1: PRISM model file for the MDP $\mathcal{M}$ given by Figure 1. This example has only one variable representing the six states in $\mathcal{M}$ and the actions `a`, `b` and `c` referring to α, β, and γ in $\mathcal{M}$, which are only used for labeling here. The initial distribution has to be realized with an additional state (`x=0`).

unchanged, if it is not assigned a new value in an update. An update could look as follows:

```
(x_1'=1) & (x_2'=true) & (x_3'=0)
```

This update assigns `1` to `x_1`, `true` to `x_2` and `0` to `x_3`. In an update, a variable is written in primed form (with `'`) to indicate that this will be the new value of that variable. Each assignment must be enclosed in parentheses and separated with `&` from other assignments. The `action` can be included optionally to label the command or for synchronization purposes. Listing 1 shows the model file for the MDP $\mathcal{M}$ from figure 1.

Actions cause synchronization when included in multiple modules. For example, if `action1` is included in two commands that are in distinct modules, they will be selected at the same time. If the guard of one of the commands with `action1` is not true, none of the commands can be selected [19, 20].

3 View

In this chapter we will introduce the core concept of this thesis, which will be called *view*. The notion of a view is to obtain a simplification of a given MDP. It is an independent MDP derived from a given MDP and represents a (simplified) view on the given one - hence the name. The original MDP is preserved.

In the preliminaries, transition systems and Markov Chains were listed as simpler versions of MDPs. Roughly speaking, transition systems and MDPs are special MDPs, namely those that have no probability distribution in each state for each action or no actions. When defining views it seems feasible to do so for the most general system of the ones being considered. Therefore we will define views on MDPs. Only for specific views and their implementation it has to be kept in mind that if they concern an action or the probability distribution of an action in a state, it is not applicable to transition systems or MCs, respectively.

3.1 Grouping Function

The conceptual idea of a view is to group states by some criteria and structure the rest of the system accordingly. To formalize the grouping, we define a dedicated function.

Definition 3.1. Let $\tilde{S}$ and M be arbitrary sets with $\bullet \in M$. The function $\tilde{F}_\theta : \tilde{S} \to M$ is called *detached grouping function*, where the symbol θ is an unique identifier.

Definition 3.2. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and $\tilde{F}_\theta : \tilde{S} \to M$ a detached grouping function where $\tilde{S} \subseteq S$. The function $F_\theta : S \to M$ with

$$s \mapsto \begin{cases} \tilde{F}_\theta(s), & \text{if } s \in \tilde{S} \\ \bullet, & \text{otherwise} \end{cases}$$

is called *grouping function*. The symbol θ is an unique identifier.

The detached grouping function is used to formalize the behavior that groupings can also be defined on a subset of states while still obtaining a total function on the set of states S. We write $M(F_\theta)$ and $M(\tilde{F}_\theta)$ to refer to the codomain of F_θ and $\tilde{F}_\theta$, respectively. The identifier θ usually indicates the objective of the grouping function. Two states will be grouped to a new state if and only if the grouping function maps them to the same value and that value is not the unique symbol $\bullet$. The mapping to the symbol $\bullet$ will be used whenever a state is to be excluded from the grouping. The definition provides an easy way to declare groups of states. The exact mapping depends on the desired grouping. We generalize the concept of the symbol $\bullet$ to tuples of it.

Definition 3.3. Let $\xi := \{\bullet\} \cup \bigcup_{n \in \mathbb{N} \setminus \{0\}} \{t \in \{\bullet\}^{n+1}\}$.

With this definition it is $\xi = \{\bullet, (\bullet, \bullet), (\bullet, \bullet, \bullet), \dots\}$ an infinite set. That is, it contains every arbitrary sized tuple only containing the symbol $\bullet$. In order to define a new set of states for the view, we define an equivalence relation R based on a given grouping function F_θ.

Definition 3.4. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and F_θ a grouping function. We define the equivalence relation $R := \{(s_1, s_2) \in S \times S \mid F_\theta(s_1) = F_\theta(s_2) \notin \xi\} \cup \{(s, s) \mid s \in S\}$

The definition states that $(s_1, s_2() \in R$ if and only if $F_\theta(s_1) = F_\theta(s_2)$ and $F_\theta(s_2) \notin \xi$. R is an equivalence relation because it is reflexive, transitive, and symmetric. R is reflexive because $\{(s, s) \mid s \in S\} \subseteq R$. Thus for all states s it is $(s, s) \in R$. Therefore, for the properties transitivity and symmetry we only consider distinct $s_1, s_2 \in S$. Consider $(s_1, s_2) \in S \times S$. If $F_\theta(s_1) \in \xi$, then it is either $F_\theta(s_1) = F_\theta(s_2) \in \xi$ or $F_\theta(s_1) \neq F_\theta(s_2)$. In both cases it follows from the definition of R that $(s_1, s_2) \notin R$. If $F_\theta(s_2) \in \xi$, it follows directly from the definition of R that $(s_1, s_2) \notin R$. So for $F_\theta(s_1) \in \xi$ or $F_\theta(s_2) \in \xi$ it follows that $(s_1, s_2) \notin R$. Thus, when considering transitivity and symmetry, we assume $F_\theta(s_1), F_\theta(s_2) \notin \xi$. Obviously, if $F_\theta(s_1) = F_\theta(s_2)$, then $F_\theta(s_2) = F_\theta(s_1)$, which means that if $(s_1, s_2) \in R$, it follows that $(s_2, s_1) \in R$. Hence R is symmetric. The property is directly conveyed by equality. In the same way, transitivity conveys directly from the equality relation to R.

We observe that two states s_1, s_2 are grouped to a new state if and only if $(s_1, s_2) \in R$. This is the case if and only if $s_1, s_2 \in [s_1]_R = [s_2]_R$ where $[s_i]_R$ for $i \in \mathbb{N}$ denotes the corresponding equivalence class of R, i. e. $[\tilde{s}]_R := \{s \in S \mid (s, \tilde{s}) \in R\}$. S/R denotes the set of all equivalence classes of R, i.e. $S/R := \bigcup_{s \in S} \{[s]_R\}$.

3.2 Formal Definition

The definition of a view is dependent on a given MDP and a grouping function F_θ. We derive the equivalence relation R as in Definition 3.4 and use its equivalence classes $[s]_R$ ($s \in S$) as states for the view. The rest of the MDP is structured accordingly.

Definition 3.5. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be MDP, $\tilde{S} \subseteq S$ and F_θ a grouping function where $\tilde{F}_\theta : \tilde{S} \to M$ is its detached grouping function. A *view* is an MDP $\mathcal{M}_\theta = (S', Act', \mathbf{P}', \iota_{init}', AP', L')$ that is derived from $\mathcal{M}$ with the grouping function F_θ where

- $S' = \{[s]_R \mid s \in S\} = S/R$
- $Act' = Act$
- $\mathbf{P} : S/R \times Act \times S/R \to [0, 1]$ with

$$\mathbf{P}'([s_1]_R, \alpha, [s_2]_R) = \frac{\sum\limits_{\substack{s_a \in [s_1]_R, \\ s_b \in [s_2]_R}} \mathbf{P}(s_a, \alpha, s_b)}{\max\{1, |\{s \in [s_1]_R \mid \alpha \in \overrightarrow{Act}(s)\}|\}}$$

- $\iota_{init}' : S/R \to [0, 1]$ with

$$\iota_{init}'([s]_R) = \sum_{s' \in [s]_R} \iota_{init}(s')$$

- $L' : S' \to \mathcal{P}(AP), [s]_R \mapsto \bigcup_{s' \in [s]_R} L(s')$

and R is the equivalence relation according to Definition 3.4.

The identifier θ is inherited from F_θ to $\mathcal{M}_\theta$. The identifier declares that F_θ is the grouping function of $\mathcal{M}_\theta$ and thereby also uniquely determines $\mathcal{M}_\theta$. If $\tilde{S}$ is not provided when instantiating a view, it is assumed that $\tilde{S} = S$. If a view takes parameters and we want to talk about an instance of a view with specific parameters $p_1, \ldots, p_n$, we write it as $\mathcal{M}_\theta(p_1, \ldots, p_n)$. If we want to talk about the grouping function F_θ of $\mathcal{M}_\theta$ with the parameters $p_1, \ldots, p_n$, we will write $F_\theta(s \,|\, p_1, \ldots, p_n)$, where $s \in S$ and S is the state set of $\mathcal{M}$.

The definition of $\mathbf{P}'$ is as it is, because a state in a view is an equivalence class, and it may be that there are several states with the same outgoing action in it. Therefore, the sum of the probabilities for two given equivalence classes and action α may yield a value greater than one (no probability anymore). Therefore, the sum of the probabilities has to be normalized with the number of states that have the action α outgoing. The maximum in the denominator ensures that there is no division by zero if an action is not outgoing from any state in the equivalence class.

Note that the definition is in a most general form in the sense that if in a view a property belongs to a part of some entity, the whole entity receives the property, i.e.

- $(s_1, \alpha, s_2) \in \mathbf{P} \;\Rightarrow\; ([s_1]_R, \alpha, [s_2]_R) \in \mathbf{P}'$
- $s \in \iota_{init} \;\Rightarrow\; [s]_R \in \iota_{init}{}'$
- $\forall s \in S : L(s) \subseteq L'([s]_R)$

Example 3.1. To clarify the concept we will consider a view $\mathcal{M}_\theta$ (Figure 2) on the MDP $\mathcal{M}$ with $S = \{s_1, \ldots, s_6\}$ in Figure 1. The view $\mathcal{M}_\theta$ is defined by its grouping function F_θ where $\tilde{F}_\theta : \tilde{S} \to M$ with

States	$\tilde{F}_\theta(s_i)$
s_1	1
s_3	1
s_4	2
s_5	2
s_6	$\bullet$

where $\tilde{S} = S \setminus \{s_2\}$ and $M = \{1, 2, \bullet\}$. We obtain $S/R = \{\{s_1, s_3\}, \{s_4, s_5\}, \{s_2\}, \{s_6\}\}$, because $F_\theta(s_1) = F_\theta(s_3)$, $F_\theta(s_4) = F_\theta(s_5)$ and $F_\theta(s_2) = F_\theta(s_6) = \bullet \in \xi$. It is $F_\theta(s_2) = \bullet$ due to Definition 3.2. Because in $\mathcal{M}$ it is

$$\iota_{init}(s_1) = 0.7 \qquad \iota_{init}(s_4) = 0.1$$
$$\iota_{init}(s_3) = 0.2 \qquad \iota_{init}(s_5) = 0$$

in $\mathcal{M}_\theta$ it is $\iota_{init}{}'(\{s_1, s_3\}) = 0.7 + 0.3 = 0.9$ and $\iota_{init}{}'(s_4, s_5) = 0.1 + 0 = 0.1$.

Because in $\mathcal{M}$ it is

$$\begin{array}{ll} L(s_1) = \emptyset & L(s_4) = \{b\} \\ L(s_3) = \{a, b\} & L(s_5) = \{a\} \end{array}$$

in $\mathcal{M}_\theta$ it is $L'(\{s_1, s_3\}) = L(s_1) \cup L(s_3) = \{a, b\}$ and $L'(\{s_4, s_5\}) = L(s_4) \cup L(s_5) = \{a, b\}$. Since s_2 and s_6 have not been grouped, they are mapped to same values with ${\iota_{init}}'$ and L' in $\mathcal{M}_\theta$ as with ι_{init} and L in $\mathcal{M}$. When considering $\mathbf{P}$ we will only look at some interesting outgoing actions for some states. In $\mathcal{M}$ it is

$$\begin{array}{ll} \mathbf{P}(s_1, \alpha, s_3) = 0.5 & \mathbf{P}(s_3, \beta, s_2) = 0.6 \\ \mathbf{P}(s_1, \beta, s_4) = 1 & \mathbf{P}(s_3, \beta, s_3) = 0.4 \end{array}$$

Because $[s_1]_R = \{s_1, s_3\}, [s_2]_R = \{s_2\}, [s_4]_R = \{s_4\}$ and each time there is only one transition with action β from a state in $[s_1]_R$ and to a state in $[s_1]_R$, $[s_2]_R$ or $[s_4]_R$ respectively, in $\mathcal{M}_\theta$ it is

$$\begin{aligned} \mathbf{P}'([s_1]_R, \beta, [s_1]_R) &= \frac{\mathbf{P}(s_3, \beta, s_3)}{|\{s \in [s_1]_R \mid \overrightarrow{Act}(s) = \beta\}|} = \frac{0.4}{|\{s_1, s_3\}|} = 0.2 \\ \mathbf{P}'([s_1]_R, \beta, [s_2]_R) &= \frac{\mathbf{P}(s_3, \beta, s_2)}{|\{s \in [s_1]_R \mid \overrightarrow{Act}(s) = \beta\}|} = \frac{0.6}{|\{s_1, s_3\}|} = 0.3 \\ \mathbf{P}'([s_1]_R, \beta, [s_4]_R) &= \frac{\mathbf{P}(s_1, \beta, s_4)}{|\{s \in [s_1]_R \mid \overrightarrow{Act}(s) = \beta\}|} = \frac{1}{|\{s_1, s_3\}|} = 0.5 \end{aligned}$$

The transition $\mathbf{P}(s_1, \alpha, s_3) = 0.5$ is interesting because it is between states in the same equivalence class. This causes a loop on the state $\{s_1, s_3\}$ in the view $\mathcal{M}_\theta$ with the transition $\mathbf{P}([s_1]_R, \alpha, [s_3]_R) = 0.5$. The value remains unchanged as in s_1, s_3 only s_1 has α outgoing, but s_3 has not. Hence the denominator of $\mathbf{P}'$ in Definition 3.5 is one.

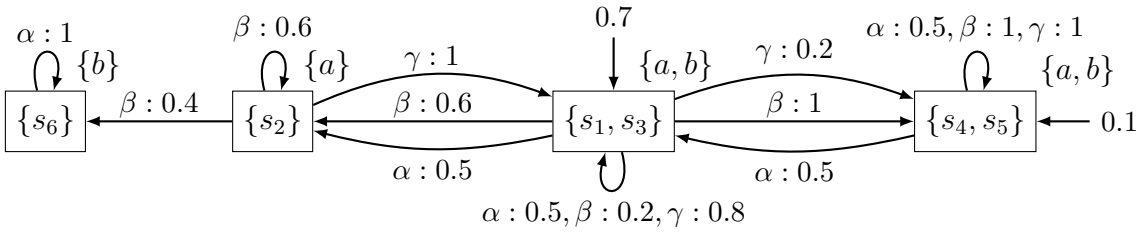


Figure 2: View $\mathcal{M}_\theta$ on MDP $\mathcal{M}$ from Figure 1

3.3 View Types and Disregarding Views

In this section we categorize views in two view types. We will present views of two types: binary and categorizing.

Definition 3.6. A view $\mathcal{M}_\theta$ is called *binary* if for every MDP $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ it holds $F_\theta[S] \in \{\{\top, \bot\}, \{\bullet, \bot\}, \{\top\, \bullet\}\}$.
A view $\mathcal{M}_\theta$ is called *categorizing* if for every MDP $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ it holds $\top, \bot \notin F_\theta[S]$.

The notion of a binary view is that it maps each state to whether or not it has a certain property. The symbol $\top$ shall be used if it has the property and the symbol $\bot$ shall be used if it does not have the property. The symbol $\bullet$ can either be used to not group states that have the property or to not group state states that do not have the property. The case of $F_\theta[S] = \{\top, \bot\}$ may seem of rather little benefit, since the resulting view only contains two states. In the actual implementation such a view might be very useful, as at the tier of visualization there will be the feature to expand grouped states and show the states they contain. Hence, at runtime it can be decided which states are to be shown, rather than in the phase of preprocessing.

The idea of a categorizing view is that it categorizes states into groups that have the same value with respect to a certain trait. One could argue that binary views also categorize states into two groups that have in common that they have or do not have a certain property. We have defined categorizing views as in Definition 3.6 so that a view is either binary or categorizing to distinguish the intention of a view.

For binary views, we have already presented three cases in which a view is called binary. We will now elaborate on these cases and generalize the concept and also enable it with categorizing views. Looking at the elements of the set $\{\{\top, \bot\}, \{\bullet, \bot\}, \{\top\, \bullet\}\}$ we observe that for distinct s, t in the set it holds $|s| - |s \cap t| = 1$, i.e. s and t differ in only one element. We declare the case when $\bullet \in F_\theta[S]$ as a special case, because instead of assigning $\top$ or $\bot$, the symbol $\bullet$ is assigned to avoid grouping on states with this property. In general, it may be that states that would be mapped to a certain value should not be grouped. To achieve this, we define disregarding views.

Definition 3.7. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and $\tilde{F}_\theta : \tilde{S} \to M$ a detached grouping function where $\tilde{S} \subseteq S$. The view ${\mathcal{M}_\theta}^{\Delta_1,\dots,\Delta_n}$ is defined by its grouping function ${F_\theta}^{\Delta_1,\dots,\Delta_n} : S \to M$ where it is $\tilde{F}_\theta^{\Delta_1,\dots,\Delta_n} : \tilde{S} \to M$ with

$$s \mapsto \begin{cases} \tilde{F}_\theta(s), & \text{if } \tilde{F}_\theta(s) \notin \{\Delta_1, \dots, \Delta_n\} \\ \bullet, & \text{otherwise} \end{cases}$$

Its grouping function ${F_\theta}^{\Delta_1,\dots,\Delta_n}$ is called F_θ *disregarding* $\Delta_1, \dots, \Delta_n$ and the respective view ${\mathcal{M}_\theta}^{\Delta_1,\dots,\Delta_n}$ is called $\mathcal{M}_\theta$ *disregarding* $\Delta_1, \dots, \Delta_n$.

With a disregarding view grouping is avoided if its detached grouping function would map to any of the values $\Delta_1, \dots, \Delta_n$. That is, when reading ${\mathcal{M}_\theta}^{\Delta_1,\dots,\Delta_n}$ we know that no grouping occurs on states with one of the properties $\Delta_1, \dots, \Delta_n$. In a sense, states with these properties are shown - perhaps among others if $\tilde{S} \subset S$. For this thesis we will only consider disregarding views for $n = 1$. For $n = 1$ $\mathcal{M}_\theta$ can be obtained from ${\mathcal{M}_\theta}^{\top}$ with

$$\tilde{F}_\theta(s) = \begin{cases} \top, & \text{if } \tilde{F}_\theta^{\top}(s) = \bullet \\ \tilde{F}_\theta^{\top}(s), & \text{otherwise} \end{cases}$$

and ${\mathcal{M}_\theta}^{\bot}$ from $\mathcal{M}_\theta$ with

$$\tilde{F}_\theta^{\bot}(s) = \begin{cases} \tilde{F}_\theta(s), & \text{if } \tilde{F}_\theta(s) = \top \\ \bullet, & \text{otherwise} \end{cases}$$

Similarly, $\mathcal{M}_\theta$ can be obtained from ${\mathcal{M}_\theta}^\perp$ and ${\mathcal{M}_\theta}^\top$ from $\mathcal{M}_\theta$. Hence, for a binary view it suffices to give a definition for one of ${\mathcal{M}_\theta}^\perp$, ${\mathcal{M}_\theta}^\top$ and $\mathcal{M}_\theta$. Normally we will define binary views as views disregarding $\top$ (i.e. ${\mathcal{M}_\theta}^\top$) and perceive them as filters that show us states that have some property. For categorizing views, we will not use disregarding views.

3.4 Composition of Views

In essence, views are a simplification generated from an MDP. It seems rather obvious that the composition of views is a very practical feature to combine simplifications. Therefore, in this chapter we will introduce, formalize, and discuss two different notions of compositions. All variants will ensure that the effect caused by each participating view also takes effect in the composed view. Furthermore, the effect of each individual view shall be reversible from the composed view.

The fundamental concept for composition that we will introduce is *parallel composition*. Its notion is to group states that match in the function value of all given grouping functions. This idea is parallel in the sense that is has no sequential characteristic.

Definition 3.8. Let $\mathcal{M}_{\theta_1}, \mathcal{M}_{\theta_2}, \ldots, \mathcal{M}_{\theta_n}$ be views, and $F_{\theta_1}, F_{\theta_2}, \ldots, F_{\theta_n}$ be their corresponding grouping functions. The grouping function $F_{\theta_1 || \theta_2 || \ldots || \theta_n} : S \to M(F_{\theta_1}) \times \cdots \times M(F_{\theta_n})$ is defined with

$$s \mapsto (F_{\theta_1}(s), F_{\theta_2}(s), \ldots, F_{\theta_n}(s))$$

and called *parallel composed grouping function*. The corresponding parallel composed view is denoted as $\mathcal{M}_{\theta_1 || \theta_2 || \ldots || \theta_n}$.

Note that for $F_{\theta_1 || \theta_2 || \ldots || \theta_n}$ if $F_{\theta_1}(s) = F_{\theta_2}(s) = \cdots = F_{\theta_n}(s) = \bullet$ the state s will not be grouped with any other state due to Definition 3.4 of R. An important property of parallel composed grouping functions is, that they defy order. That is, with regard to the impact on the grouping the order of the included grouping functions does not matter.

Proposition 3.1. *Let F_u and F_v be non parallel composed grouping functions and S a set of states. For all $s_1, s_2 \in S$ it holds that*

$$F_{u||v}(s_1) = F_{u||v}(s_2) \iff F_{v||u}(s_1) = F_{v||u}(s_2)$$

Proof. Let F_u and F_v be grouping functions, $s_1, s_2 \in S$ and

$$\begin{aligned} F_{u||v}(s_1) &= (F_u(s_1), F_v(s_1)) =: (a, b) \\ F_{u||v}(s_2) &= (F_u(s_2), F_v(s_2)) =: (x, y) \end{aligned}$$

Then it is

$$\begin{aligned} F_{v||u}(s_1) &= (F_v(s_1), F_u(s_1)) = (b, a) \\ F_{v||u}(s_2) &= (F_v(s_2), F_u(s_2)) = (y, x) \end{aligned}$$

It follows that

$$F_{u||v}(s_1) = F_{u||v}(s_2) \iff F_{v||u}(s_1) = F_{v||u}(s_2)$$

because $(a, b) = (x, y) \iff (a = x) \land (b = y) \iff (b, a) = (y, x)$. □

We assume that whenever a view or grouping function is parallel composed, it is notated as declared in Definition 3.8. That is, if and only if there is the operator $||$ in the subsrcipt of a view or grouping function, it is parallel composed. We define the operator $||$ to construct parallel composed grouping functions and views.

Definition 3.9. The operator $||$ maps two grouping functions to a new grouping function. It is defined inductively.

1. $(F_{\theta_1} || F_{\theta_2})(s) := F_{\theta_1||\theta_2}(s)$
2. $(F_{\theta_1||\ldots||\theta_n} || F_\theta)(s) := F_{\theta_1||\ldots||\theta_n||\theta}(s) \quad$ where $n \in \mathbb{N}$
3. $(F_\theta || F_{\theta_1||\ldots||\theta_n})(s) := F_{\theta||\theta_1||\ldots||\theta_n}(s) \quad$ where $n \in \mathbb{N}$

Proposition 3.2. *The operator $||$ is associative.*

Proof. Let $F_{\theta_1}, F_{\theta_2}, F_{\theta_3}$ be grouping functions. For simplicity we omit the parameter (state s).

$$F_{\theta_1} || (F_{\theta_2} || F_{\theta_3}) = F_{\theta_1} || F_{\theta_2||\theta_3} = F_{\theta_1||\theta_2||\theta_3} = F_{\theta_1||\theta_2} || F_{\theta_3} = (F_{\theta_1} || F_{\theta_2}) || F_{\theta_3}$$

The proof is analogous if one or several of the grouping functions are parallel composed. □

We write $\mathcal{M}_{\theta_1} || \mathcal{M}_{\theta_2}$ for $\mathcal{M}_{\theta_1||\theta_2}$ defined by the grouping function $F_{\theta_1} || F_{\theta_2} = F_{\theta_1||\theta_2}$. Note that the operator $||$ is obviously not commutative, but as parallel composed grouping functions defy order, the absence of this property will not have any effect on the resulting grouping.

Definition 3.10. Let $F_{\theta_1||\ldots||\theta_n}$ be a parallel composed grouping function. The operator $||^{-1}$ is defined as

$$F_{\theta_1,\ldots,\theta_n}||^{-1} F_{\theta_i} := (F_{\theta_1}, \ldots, F_{\theta_{i-1}}, F_{\theta_i+1}, \ldots, F_n) = F_{\theta_1||\ldots||\theta_{i-1}||\theta_{i+1}||\ldots||\theta_n}$$

The operator $||^{-1}$ is the reversing operator to $||$. Given a parallel composed view and an in it contained grouping function, it removes this view. We write $\mathcal{M}_{\theta_1||\ldots||\theta_n}||^{-1}\mathcal{M}_{\theta_i}$ for the view defined by the grouping function $F_{\theta_1||\ldots||\theta_{i-1}||\theta_{i+1}||\ldots||\theta_n}$.

A variant of parallel composition is *selective composition*. It aims for application of the grouping function only on certain states, where the other composing functions have a certain value.

Definition 3.11. Let $\mathcal{M}_{\theta_1||\ldots||\theta_n}$ be a parallel composed view with its grouping function $F_{\theta_1||\ldots||\theta_n}$, where n might be 1 and let $\mathcal{M}_\theta$ be another view with its grouping function F_θ. We write M_i for the image of F_{θ_i}. Given a set $Z \subseteq \{(F_{\theta_i}, a) \mid a \in M_i, i \in \{1, \ldots, n\}\}$ the operator $||_Z$ is defined as $F_{\theta_1||\ldots||\theta_n} \,||_Z\, F_\theta := F_{\theta_1||\ldots||\theta_n||\theta} : S \to M$ with

$$s \mapsto \begin{cases} (F_{\theta_1}(s), \ldots, F_{\theta_n}(s), F_\theta(s)) & \text{if } \forall i \in \{1, \ldots, n\} : (F_{\theta_i}, F_{\theta_i}(s)) \in Z \\ (F_{\theta_1}(s), \ldots, F_{\theta_n}(s), \bullet), & \text{otherwise} \end{cases}$$

Then $\mathcal{M}_{\theta_1||\ldots||\theta_n||\theta}$ is a parallel composed view with $F_{\theta_1||\theta_2||\ldots||\theta_n||\theta}$ being its parallel composed grouping function.

The set Z determines on which states the view $\mathcal{M}_\theta$ shall take effect. For instance, $Z = \{(F_{\theta_1}, a_1), (F_{\theta_1}, b_1), (F_{\theta_2}, a_2)\}$ induces that $\mathcal{M}_\theta$ only takes effect on a state s if

$$((F_{\theta_1}(s) = a_1) \vee (F_{\theta_1}(s) = b_1)) \wedge (F_{\theta_2}(s) = a_2)$$

That is, if this boolean expression is false, the last entry in the tuple is $\bullet$ and only otherwise it is $F_\theta(s)$.

4 View Examples

In this chapter we will introduce and discuss some view examples created by the author. Their purpose is to understand the idea and concept of a view and to get to know some views that might be useful in real world applications.
When considering views, we want to consider only those that exploit properties of MDPs, or that perform computations that are also feasible on normal graphs, but are of explicit relevance to MDPs.

4.1 Views Utilizing MDP Characteristics

In this section, we will introduce some views that are purely based on the properties of an MDP. This includes views that leverage elements of the tuple from the definition of MDPs ("$\mathcal{M}$ components"), but also information associated with it, such as variables, reward or cost structure, and model checking results.

4.1.1 Atomic Propositions

One of the least involved approaches to create a view is to base it on the atomic propositions assigned to each state by the labeling function. Atomic Propositions are of relevance because in they are the base tools to identify states with certain notable properties. The notion is to group states that were assigned the same set of atomic propositions.

Definition 4.1. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and $\tilde{S} \subseteq S$. The view $\mathcal{M}_{AP}$ is defined by its grouping function F_{AP} where $\tilde{F}_{AP} : \tilde{S} \to M, s \mapsto L(s)$ with $M = AP \cup \{\bullet\}$.

The grouping function is exactly the labeling function i.e. for all $s \in S$ it is $F_{AP}(s) := L(s)$. So it is $F_{AP}(s_1) = F_{AP}(s_2)$ if and only if $L(s_1) = L(s_2)$. By Definition 3.4 for $\tilde{s} \in S$ it is $[\tilde{s}]_R = \{s \in S \mid L(s) = L(\tilde{s})\}$.

Thereby we obtain the view $\mathcal{M}_{AP}$ for a given MDP $\mathcal{M}$ where: $S' = \bigcup_{s \in S}\{[s]_R\} = \bigcup_{a \in AP}\{\{s \in S \mid L(s) = a\}\}$. All other components are constructed as in Definition 3.5.

Example 4.1. In Figure 3 we can observe the view $\mathcal{M}_{AP}$ on $\mathcal{M}$. In the simplified representation on the left the assigned set of atomic propositions of each state are noted next to them. There are four different sets of atomic propositions: $\{\}, \{a\}, \{b\}$ and $\{a, b\}$. In the view on the right the states with the same set of atomic propositions have been grouped.

Although this view may seem rather simple, since it essentially only performs $F_{AP} := L$, it is the most powerful. This is because every view presented below is reducible to this one. This is because a grouping function assigns a value to each state. This value could be interpreted as an atomic proposition. The reduction can be realized by replacing the labeling function with the grouping function of the respective view. That is, $L := F_\theta$ and $AP := F_\theta[S]$ for some grouping function F_θ. While this works, it alters the underlying MDP.

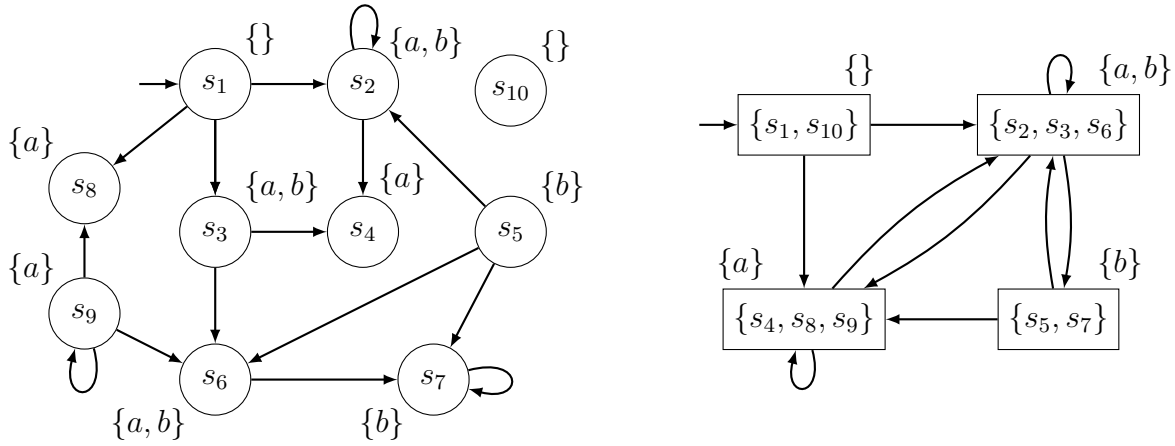


Figure 3: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{AP}$ on it (right)

4.1.2 Initial States

A slightly more complicated idea than directly using a given function is to use the support of the initial distribution $\{s \in S \mid \iota_{init}(s) > 0\}$. We can group states that have a probability greater than zero to start from. In practice, this may be useful to quickly find all initial states.

Definition 4.2. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $I := \{s \in S \mid \iota_{init}(s) > 0\}$. The view ${\mathcal{M}_I}^\top$ is defined by its grouping function ${F_I}^\top$ where $\tilde{F}_I^\top : \tilde{S} \to M$ with

$$\tilde{F}_I^\top(s) = \begin{cases} \bullet, & \text{if } s \in I \\ \bot, & \text{otherwise} \end{cases}$$

and $M := \{\bullet, \bot\}$.

By Definition 3.4 the equivalence classes of R on S are

$$[s]_R = \{s \in S \mid {F_I}^\top(s) = \bullet\} = \{s\} \qquad \text{for } s \in I$$
$$[s]_R = \{s \in S \mid {F_I}^\top(s) = \bot\} \qquad \text{otherwise}$$

Thereby we obtain the view ${\mathcal{M}_I}^\top$ for a given an MDP $\mathcal{M}$ where: $S' = \bigcup_{s \in S}\{[s]_R\} = \{s \in S \mid s \notin I\} \cup \bigcup_{s \in I}\{\{s\}\}$. All other components are constructed as in Definition 3.5.

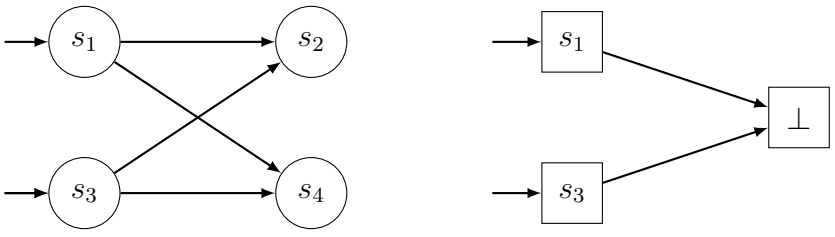


Figure 4: Simplified representations of $\mathcal{M}$ (left) and the view ${\mathcal{M}_I}^\top$ on it (right)

Example 4.2. In Figure 4 we can observe the effect of ${\mathcal{M}_I}^\top$. In the simplified representation of an MDP on the left the states s_1 and s_3 are marked as initial states ($s_1, s_3 \in I$). Hence, these two are not grouped whereas the remaining two are grouped.

4.1.3 Outgoing Actions

Another crucial component of an MDP is its set of actions *Act*. Actions are used for interprocess communication and synchronization. In this subsection we will provide and discuss some views that utilize outgoing actions from a state. The most apparent notion to group states is to group states that *have* a given outgoing action.

Definition 4.3. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view ${\mathcal{M}_{\exists\overrightarrow{\alpha}}}^\top$ is defined by its grouping function ${F_{\exists\overrightarrow{\alpha}}}^\top$ where $\tilde{F}^\top_{\exists\overrightarrow{\alpha}} : S \to M$ with

$$\tilde{F}^\top_{\exists\overrightarrow{\alpha}}(s) = \begin{cases} \bullet, & \text{if } \exists s' \in S : (s, \alpha, s') \in \mathbf{P} \\ \bot, & \text{otherwise} \end{cases}$$

and $M := \{\bullet, \bot\}$.

By Definition 3.4 the equivalence classes of R on S are

$$\begin{aligned} [s]_R &= \{s \in S \mid {F_{\exists\overrightarrow{\alpha}}}^\top(s) = \bullet\} = \{s\} && \text{if } \exists s' \in S : (s, \alpha, s') \in \mathbf{P} \\ [s]_R &= \{s \in S \mid {F_{\exists\overrightarrow{\alpha}}}^\top(s) = \bot\} && \text{otherwise} \end{aligned}$$

Thereby we obtain the view ${\mathcal{M}_{\exists\overrightarrow{\alpha}}}^\top$ for a given MDP $\mathcal{M}$ where $S' = \bigcup_{s\in S}\{[s]_R\} =: S_1 \cup S_2$ where

$$\begin{aligned} S_1 &:= \{\{s\} \mid s \in S, \exists s' \in S : (s, \alpha, s') \in \mathbf{P}\}\} \\ &= \{\{s\} \mid s \in S, {F_{n\leq\overrightarrow{\alpha}}}^\top(s) = \bullet\} = \bigcup_{s\in S\setminus S_2} \{\{s\}\} \text{ and} \\ S_2 &:= \{\{s \in S \mid \neg\exists s' \in S : (s, \alpha, s') \in \mathbf{P}\}\} \\ &= \{\{s \in S \mid {F_{n\leq\overrightarrow{\alpha}}}^\top(s) = \bot\}\}. \end{aligned}$$

Example 4.3. In Figure 5 we can observe the view ${\mathcal{M}_{\exists\overrightarrow{\alpha}}}^\top(\alpha)$ $\mathcal{M}$. In the simplified representation of an MDP on the left the action α is outgoing from states s_1, s_2, s_3 and s_4, whereas it is not outgoing from s_5 and s_6. Hence, $s_1, \ldots, s_4$ are not grouped but shown, and states s_5 and s_6 are grouped.

Since actions are a very important part of MDPs and TS, it seems useful to further enhance this view and look at variants of it. Instead of only showing states that *have* an action outgoing action we could quantify the amount of times that action should be outgoing. For example we could require that a given action has to be outgoing a minimum amount of times.

Definition 4.4. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view ${\mathcal{M}_{n\leq\overrightarrow{\alpha}}}^\top$ is defined by its grouping function ${F_{n\leq\overrightarrow{\alpha}}}^\top$ where $\tilde{F}^\top_{n\leq\overrightarrow{\alpha}} : \tilde{S} \to M$ with

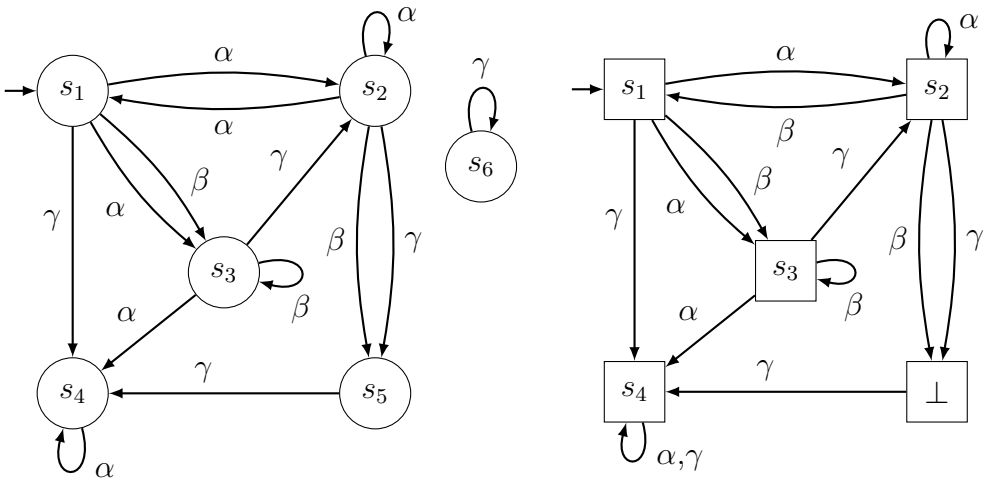


Figure 5: Simplified representations of $\mathcal{M}$ (left) and the view ${\mathcal{M}_{\exists\overrightarrow{\alpha}}}^{\top}(\alpha)$ on it (right)

$$\tilde{F}^{\top}_{n\leq\overrightarrow{\alpha}}(s) = \begin{cases} \bullet, & \text{if } \exists s_1,\dots,s_n \in S : Q_{n\leq\overrightarrow{\alpha}}(s,s_1,\dots,s_n) \\ \perp, & \text{otherwise} \end{cases}$$

where $M := \{\bullet, \perp\}$, $n \in \mathbb{N}$ is the minimum amount of times a transition with action α has to be outgoing in order to be grouped with the other states and

$$Q_{n\leq\overrightarrow{\alpha}}(s,s_1,\dots,s_n) := ((s,\alpha,s_1),\dots,(s,\alpha,s_n) \in \mathbf{P}) \wedge |\{s_1,\dots,s_n\}| = n$$

is a first order logic predicate.

The predicate $Q_{n\leq\overrightarrow{\alpha}}(s,s_1,\dots,s_n)$ requires that there are transitions with action α to n distinct states, but would also be satisfied, if there were more. By Definition 3.4 the equivalence classes of R on S are

$$\begin{aligned} &[s]_R = \{s \in S \mid {F_{n\leq\overrightarrow{\alpha}}}^{\top}(s) = \bullet\} = \{s\} && \text{if } \exists s_1,\dots,s_n \in S : Q_{n\leq\overrightarrow{\alpha}}(s,s_1,\dots,s_n) \\ &[s]_R = \{s \in S \mid {F_{n\leq\overrightarrow{\alpha}}}^{\top}(s) = \perp\} && \text{otherwise} \end{aligned}$$

Thereby we obtain the view ${\mathcal{M}_{n\leq\overrightarrow{\alpha}}}^{\top}$ for a given MDP $\mathcal{M}$ where $S' = \bigcup_{s\in S}\{[s]_R\} =: S_1 \cup S_2$ where

$$\begin{aligned} S_1 &:= \{\{s\} \mid s \in S,\, \exists s_1,\dots,s_n \in S : Q_{n\leq\overrightarrow{\alpha}}(s,s_1,\dots,s_n)\} \\ &= \{\{s\} \mid s \in S,\, {F_{n\leq\overrightarrow{\alpha}}}^{\top}(s) = \bullet\} = \bigcup_{s\in S\setminus S_2} \{\{s\}\} \text{ and} \\ S_2 &:= \{\{s \in S \mid \neg\exists s_1,\dots,s_n \in S : Q_{n\leq\overrightarrow{\alpha}}(s,s_1,\dots,s_n)\}\} \\ &= \{\{s \in S \mid {F_{n\leq\overrightarrow{\alpha}}}^{\top}(s) = \perp\}\}. \end{aligned}$$

Example 4.4. In Figure 6 we can observe the view ${\mathcal{M}_{n\leq\overrightarrow{\alpha}}}^{\top}(\alpha,2)$ on $\mathcal{M}$. In the simplified representation of an MDP on the left the action α is outgoing zero times from s_5, s_6, one time from s_3, s_4 and two times from s_1, s_2. Hence, s_1 and s_2 are not grouped but shown, and states $s_3,\dots,s_6$ are grouped.

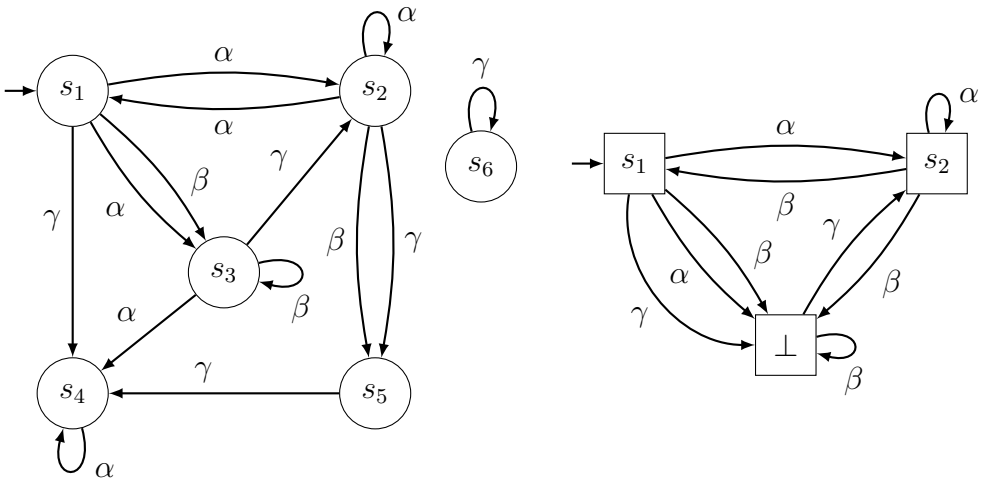


Figure 6: Simplified representations of $\mathcal{M}$ (left) and the view ${\mathcal{M}_{n\leq\overrightarrow{\alpha}}}^{\top}(\alpha, 2)$ on it (right).

In a similar fashion we define view, which groups states where a given action occurs at most a certain number of times.

Definition 4.5. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view ${\mathcal{M}_{\overrightarrow{\alpha}\leq n}}^{\top}$ is defined by its grouping function ${F_{\overrightarrow{\alpha}\leq n}}^{\top}$ where $\tilde{F}^{\top}_{\overrightarrow{\alpha}\leq n} : \tilde{S} \to M$ with

$$\tilde{F}^{\top}_{\overrightarrow{\alpha}\leq n}(s) = \begin{cases} \bullet, & \text{if } \forall s_1, \ldots, s_{n+1} \in S : Q_{\overrightarrow{\alpha}\leq n}(s, s_1, \ldots, s_{n+1}) \\ \perp, & \text{otherwise} \end{cases}$$

where $M := \{\bullet, \perp\}$, is the maximal number of times a transition with action α may be outgoing and

$$Q_{\overrightarrow{\alpha}\leq n}(s, s_1, \ldots, s_{n+1}) := ((s, \alpha, s_1), \ldots, (s, \alpha, s_{n+1}) \in \mathbf{P}) \implies \bigvee_{\substack{i,j\in\{1,\ldots,n+1\} \\ i<j}} s_i = s_j$$

is a first order logic predicate.

The predicate $Q_{\overrightarrow{\alpha}\leq n}(s, s_1, \ldots, s_{n+1})$ ensures that there are at most n actions α outgoing, by requiring that if there exist more than n transitions outgoing with action α, at least two of them are the same. It does so by enforcing that the states in which the transitions end are in fact the same. The reasoning about the equality of the grouping function values, the obtained equivalence classes, and the resulting set of states S' of view is analogous to ${\mathcal{M}_{n\leq\overrightarrow{\alpha}}}^{\top}$.

Example 4.5. In Figure 7 we can observe the view ${\mathcal{M}_{\overrightarrow{\alpha}\leq n}}^{\top}(\alpha, 1)$ on $\mathcal{M}$. In the simplified representation of an MDP on the left the action α is outgoing zero times from s_5, s_6, one time from s_3, s_4 and two times from s_1, s_2. Hence, $s_3, \ldots, s_6$ are not grouped but shown, and states s_1 and s_2 are grouped.

Since we have already defined grouping functions and hence views for a required minimum and maximal amount of times an action has to be outgoing it is now easily

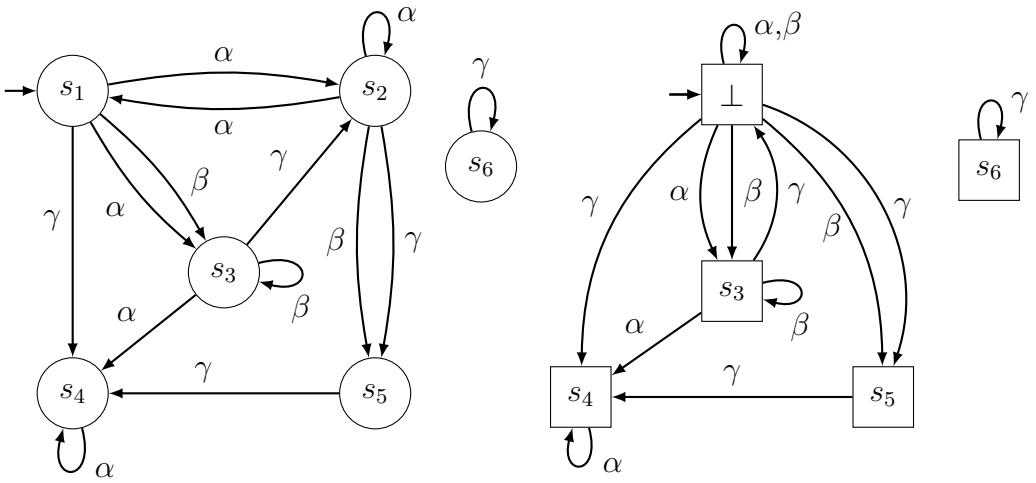


Figure 7: Simplified representations of $\mathcal{M}$ (left) and the view ${\mathcal{M}_{\overrightarrow{\alpha}\leq n}}^{\top}(\alpha, 1)$ on it (right)

possible to define a view that groups states where the amount of outgoing actions is at least n and at most m.

Definition 4.6. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view ${\mathcal{M}_{m\leq\overrightarrow{\alpha}\leq n}}^{\top}$ is defined by its grouping function where $\tilde{F}^{\top}_{m\leq\overrightarrow{\alpha}\leq n} : \tilde{S} \to M$ with

$$\tilde{F}^{\top}_{m\leq\overrightarrow{\alpha}\leq n}(s) = \begin{cases} \bullet, & \text{if } \exists s_1, \ldots, s_m \in S : Q_{m\leq\overrightarrow{\alpha}}(s, s_1, \ldots, s_m) \\ & \text{and } \forall s_1, \ldots, s_{n+1} \in S : Q_{\overrightarrow{\alpha}\leq n}(s, s_1, \ldots, s_{n+1}) \\ \bot, & \text{otherwise} \end{cases}$$

and $m, n \in \mathbb{N}$ are the minimum and maximum number of transitions with action α for a state to be grouped. The predicates $Q_{n\leq\overrightarrow{\alpha}}(s, s_1, \ldots, s_n)$ and $Q_{\overrightarrow{\alpha}\leq n}(s, s_1, \ldots, s_{n+1})$ are the predicates from Definition 4.4 and Definition 4.5 respectively.

For a given $s \in S$, the expressions $\exists s_1, \ldots, s_n \in S : Q_{n\leq\overrightarrow{\alpha}}(s, s_1, \ldots, s_n)$ and $\forall s_1, \ldots, s_{m+1} \in S : Q_{\overrightarrow{\alpha}\leq m}(s, s_1, \ldots, s_{m+1})$ from Definition 4.4 and Definition 4.5 require that s has a minimum and maximum amount of outgoing transitions with an action α. Hence, the conjunction will be true for states where the amount of outgoing transitions with action α is element of the set $\{m, n+1, \ldots, n-1, n\}$. We will write that the number of outgoing actions is *in the span*. For a given state s and action α, for convenience we set

$$\begin{aligned} C_{s,\overrightarrow{\alpha}} := {} & \exists s_1, \ldots, s_m \in S : Q_{m\leq\overrightarrow{\alpha}}(s, s_1, \ldots, s_m) \\ & \wedge \forall s_1, \ldots, s_{n+1} \in S : Q_{\overrightarrow{\alpha}\leq n}(s, s_1, \ldots, s_{n+1}) \end{aligned}$$

$C_{s,\overrightarrow{\alpha}}$ is true if and only if the number of outgoing actions is in the span. For $s_1, s_2 \in S$ it is ${F_{m\leq\overrightarrow{\alpha}\leq n}}^{\top}(s_1) = {F_{m\leq\overrightarrow{\alpha}\leq n}}^{\top}(s_2)$ if and only if $C_{s_1,\overrightarrow{\alpha}} \wedge C_{s_2,\overrightarrow{\alpha}}$ or $s_1 = s_2$.

By Definition 3.4 the equivalence classes of R on S are

$$[s]_R = \{s \in S \mid {F_{m\leq\vec{\alpha}\leq n}}^{\top}(s) = \bullet\} = \{s\} \qquad \text{if } C_{s,\vec{\alpha}} \text{ true}$$
$$[s]_R = \{s \in S \mid {F_{m\leq\vec{\alpha}\leq n}}^{\top}(s) = \bot\} \qquad \text{otherwise}$$

The new set of states S' of the view ${\mathcal{M}_{m\leq\vec{\alpha}\leq n}}^{\top}$ is the union of the equivalence classes of the equivalence relation R on the set of states S of the original MDP. Hence it is $S' = \bigcup_{s\in S}[s]_R =: S_1 \cup S_2$ where

$$S_1 := \{\{s\} \mid s \in S,\, C_{s,\vec{\alpha}} \text{ true}\}$$
$$= \{\{s\} \mid s \in S,\, {F_{m\leq\vec{\alpha}\leq n}}^{\top}(s) = \bullet\} = \bigcup_{s\in S\setminus S_2} \{\{s\}\}$$
$$S_2 := \{\{s \in S \mid C_{s,\vec{\alpha}} \text{ false}\}\} = \{\{s \in S \mid {F_{m\leq\vec{\alpha}\leq n}}^{\top}(s) = \bot\}\}.$$

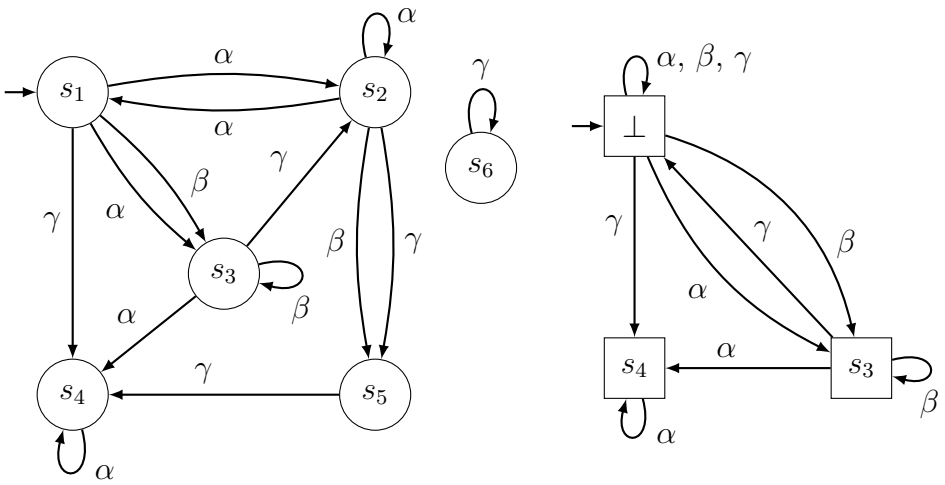


Figure 8: Simplified representations of $\mathcal{M}$ (left) and the view ${\mathcal{M}_{m\leq\vec{\alpha}\leq n}}^{\top}(1, \alpha, 1)$ on it (right)

Example 4.6. In Figure 8 we can observe the view ${\mathcal{M}_{m\leq\vec{\alpha}\leq n}}^{\top}(1, \alpha, 1)$ on $\mathcal{M}$. It shows states that have the action α at least one time outgoing and at most 1 one time. Hence, it does not group states, where the action α is outgoing exactly once. All all remaining states are grouped. In the simplified representation of an MDP on the left, the action α is outgoing zero times from s_5, s_6, one time from s_3, s_4 and two times from s_1, s_2. Hence, $s_4, \ldots, s_5$ are not grouped but shown, and the remaining states s_1, s_2, s_5, s_6 are grouped.

Instead of making requirements on states and grouping them based on whether they meet those requirements, it is also possible to group states that are very similar or even identical with respect to their outgoing actions. First, we will consider the case where the outgoing actions and their set are identical.

Definition 4.7. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and $\tilde{S} \subseteq S$. The view $\mathcal{M}_{\overrightarrow{Act(s)_=}}$ is defined by its grouping function $F_{\overrightarrow{Act(s)_=}}$ where $\tilde{F}^{\top}_{\overrightarrow{Act(s)_=}} : \tilde{S} \to M$ with

$$s \mapsto \{(\alpha, n) \mid \alpha \in Act, n = |\{(s, \alpha, s') \in \mathbf{P}\}|\}$$

and $M := Act \times \mathbb{N} \cup \{\bullet\}$.

The grouping function assigns a set of pairs to each state. For each action in Act, one pair is contained, declaring the number of times the action is outgoing from that state. Note that n is zero if an action is not outgoing from a state. By Definition 3.4 the equivalence classes of R on S are

$$[s]_R := \{s' \in S \mid F_{\overrightarrow{Act}(s)_=}(s') = F_{\overrightarrow{Act}(s)_=}(s) = \{(\alpha_1, n_1), \ldots, (\alpha_l, n_l)\}\} \text{ where } l = |Act|$$

By Definition 3.5 the set $S' := \bigcup_{s \in S} [s]_R$ is the set of states of $\mathcal{M}_{\overrightarrow{Act}(s)_=}$. All other components of $\mathcal{M}_{\overrightarrow{Act}(s)_=}$ are structured as usual according to the definition 3.5.

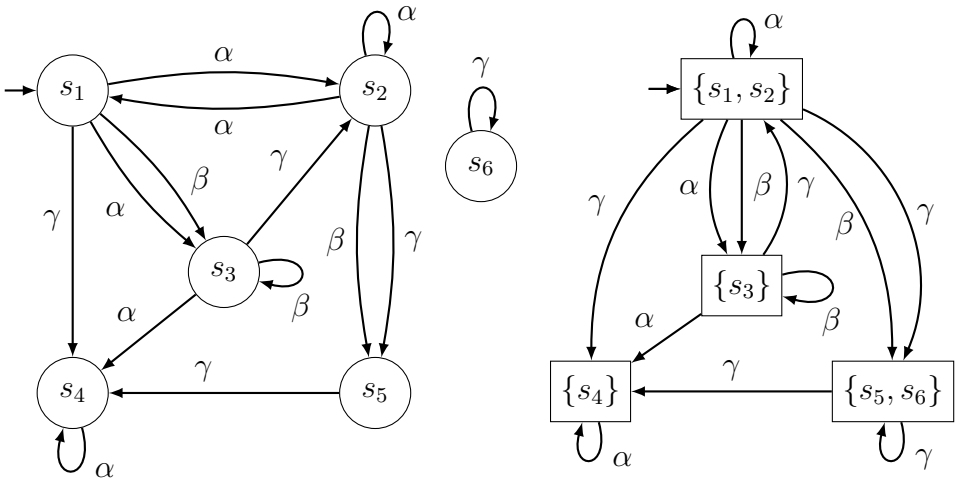


Figure 9: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\overrightarrow{Act}(s)_=}$ on it (right)

Example 4.7. In Figure 9 we can observe the view $\mathcal{M}_{\overrightarrow{Act}(s)_=}$ on $\mathcal{M}$ (both simplified representations). For $\mathcal{M}_{\overrightarrow{Act}(s)_=}$ it is $F_{\overrightarrow{Act}(s)_=}$:

$$s_1 \mapsto \{(\alpha, 2), (\beta, 1), (\gamma, 1)\} \qquad s_4 \mapsto \{(\alpha, 1), (\beta, 0), (\gamma, 0)\}$$
$$s_2 \mapsto \{(\alpha, 2), (\beta, 1), (\gamma, 1)\} \qquad s_5 \mapsto \{(\alpha, 0), (\beta, 0), (\gamma, 1)\}$$
$$s_3 \mapsto \{(\alpha, 1), (\beta, 1), (\gamma, 1)\} \qquad s_6 \mapsto \{(\alpha, 0), (\beta, 0), (\gamma, 1)\}$$

because these are number of corresponding outgoing actions from each state. We see that the sets are equal for s_1 and s_2 and for s_5 and s_6. So these two pairs are grouped into new states each. The remaining states are not grouped with any other state. The equivalence classes containing them are singleton sets.

The view $\mathcal{M}_{\overrightarrow{Act}(s)_=}$, as well as ${\mathcal{M}_{n \leq \overrightarrow{\alpha}}}^\top$, ${\mathcal{M}_{\overrightarrow{\alpha} \leq n}}^\top$, ${\mathcal{M}_{m \leq \overrightarrow{\alpha} \leq n}}^\top$ were originally motivated by transition systems in which there are no probabilities, but multiple actions starting from a single state may occur. Since MDPs can model transition systems, they were included. For MDPs with proper probability distributions, counting outgoing actions may be less helpful. Instead, one could disregard the amount of times an action is outgoing and only be interested in states that have the same options for the nondeterministic choice before the probabilistic one.

Definition 4.8. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and $\tilde{S} \subseteq S$. The view $\mathcal{M}_{\overrightarrow{Act}(s)\approx}$ is defined by its grouping function $F_{\overrightarrow{Act}(s)\approx}$ where $\tilde{F}^{\top}_{\overrightarrow{Act}(s)\approx} : \tilde{S} \to M$ with

$$s \mapsto \overrightarrow{Act}(s)$$

and $M := Act \cup \{\bullet\}$.

For $s_1, s_2 \in S$ it is $F_{\overrightarrow{Act}(s)\approx}(s_1) = F_{\overrightarrow{Act}(s)\approx}(s_2)$ if and only if they are mapped to the same set of actions. Hence the equivalence classes of R are

$$[s]_R = \{s' \in S \mid F_{\overrightarrow{Act}(s)\approx}(s') = F_{\overrightarrow{Act}(s)\approx}(s) =: \{\alpha_1, \ldots, \alpha_l\}\} \text{ where } l \leq |Act|.$$

By Definition 3.5 the set $S' := \bigcup_{s \in S} [s]_R$ is the set of states of $\mathcal{M}_{\overrightarrow{Act}(s)\approx}$.

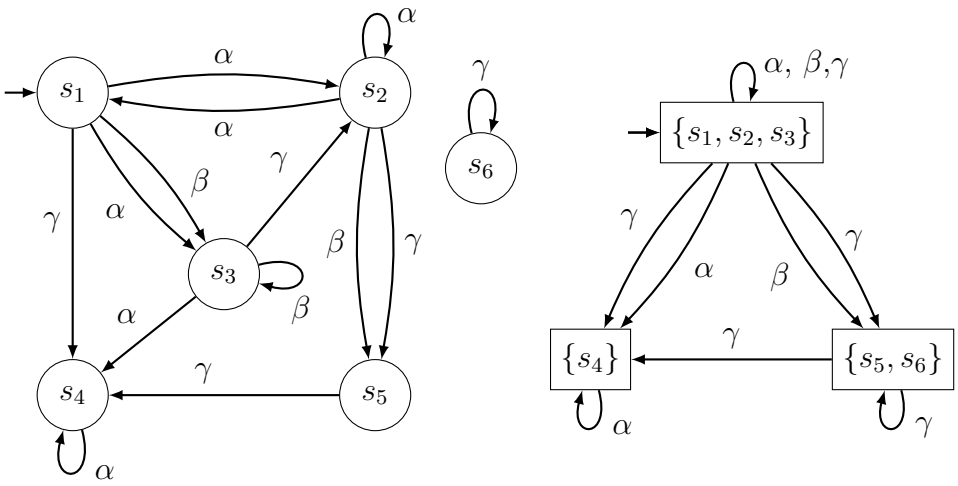


Figure 10: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\overrightarrow{Act}(s)\approx}$ on it (right)

Example 4.8. In Figure 10 we can observe the view $\mathcal{M}_{\overrightarrow{Act}(s)\approx}$ on $\mathcal{M}$ (both simplified representations). For $\mathcal{M}_{\overrightarrow{Act}(s)\approx}$ it is $F_{\overrightarrow{Act}(s)\approx}$:

$$\begin{aligned} s_1 &\mapsto \{\alpha, \beta, \gamma\} & s_4 &\mapsto \{\alpha\} \\ s_2 &\mapsto \{\alpha, \beta, \gamma\} & s_5 &\mapsto \{\gamma\} \\ s_3 &\mapsto \{\alpha, \beta, \gamma\} & s_6 &\mapsto \{\gamma\} \end{aligned}$$

because these are the corresponding outgoing actions from each state. We see that the sets are equal for s_1, s_2 and s_3 and for s_5 and s_6. Hence, the state set of the view $\mathcal{M}_{\overrightarrow{Act}(s)\approx}$ on $\mathcal{M}$ has the state set $\{\{s_1, s_2, s_3\}, \{s_4\}, \{s_5, s_5\}\}$.

Apart from the option of directly considering one or a set of outgoing actions with possible quantities it is also possible to only consider the quantity of outgoing actions without regarding any specific action.

Definition 4.9. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view $\mathcal{M}_{|\overrightarrow{Act}(s)|}$ is defined by its grouping function $F_{|\overrightarrow{Act}(s)|}$ where $\tilde{F}_{|\overrightarrow{Act}(s)|} : \tilde{S} \to M$ with

$$s \mapsto |\overrightarrow{Act}(s)|$$

and $M := \mathbb{N} \cup \{\bullet\}$.

By Definition 3.4 the equivalence classes of R on S are

$$[s]_R = \{s \in S \mid F_{|\overrightarrow{Act}(s)|}(s') = F_{|\overrightarrow{Act}(s)|}(s)\} = \{s \in S \mid \overrightarrow{Act}(s') = \overrightarrow{Act}(s)\}$$

By Definition 3.5 the set $S' := \bigcup_{s \in S} [s]_R$ is the set of states of $\mathcal{M}_{\overrightarrow{Act}(s)\approx}$.

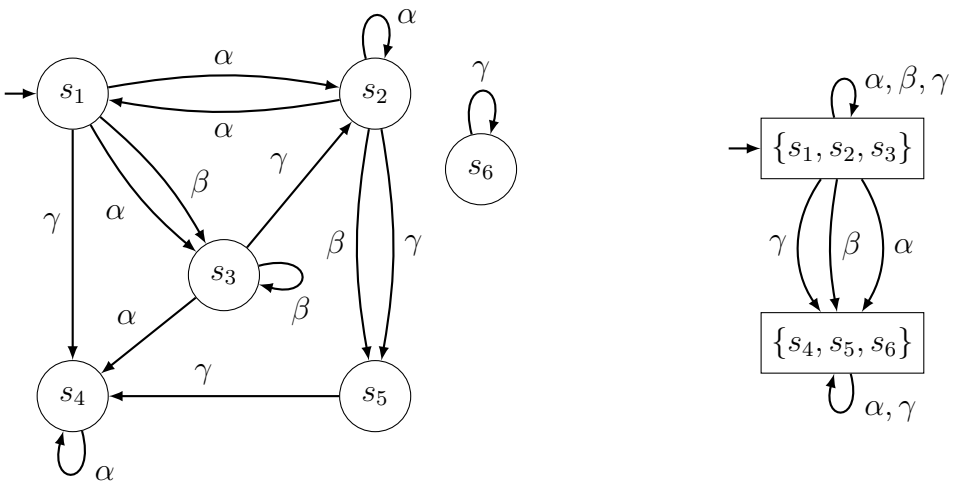


Figure 11: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{|\overrightarrow{Act}(s)|}$ on it (right)

Example 4.9. In Figure 11 we can observe the view $\mathcal{M}_{|\overrightarrow{Act}(s)|}$ on $\mathcal{M}$. In the simplified representation of an MDP on the states s_1, s_2, s_3 have the outgoing actions α, β, γ, the state s_4 has the outgoing action α and the state s_5, s_6 have the outgoing action γ. Hence, the number of outgoing actions for s_1, s_2, s_3 is three and for s_4, s_5, s_6 is one. Therefore, each of these states becomes a new single state in the view $\mathcal{M}_{|\overrightarrow{Act}(s)|}$.

The notion of the view $\mathcal{M}_{|\overrightarrow{Act}(s)|}$ is similar to utilize the outdegree, with the difference being here that ougoing transitions with the same action are considered as one single edge. This reflects on the options available for nondeterminism in this state.

The special case where there is only a single outgoing action is worth a distinct view, since it hides all non-deterministic choices, but nothing more.

Definition 4.10. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view $\mathcal{M}_{|\overrightarrow{Act}(s)|_1}{}^{\perp}$ is defined by its grouping function $F_{|\overrightarrow{Act}(s)|_1}{}^{\perp}$ where $\tilde{F}^{\perp}_{|\overrightarrow{Act}(s)|_1} : \tilde{S} \to M$ with

$$\tilde{F}^{\perp}_{|\overrightarrow{Act}(s)|_1}(s) = \begin{cases} \top, & \text{if } |\overrightarrow{Act}(s)| = 1 \\ \bullet, & \text{otherwise} \end{cases}$$

and $M := \mathbb{N} \cup \{\bullet\}$.

Note that this view is disregarding $\perp$ and hence groups states that *have* the property. This version was chosen because the intention of the view is to hide S in which there is no nondeterministic choice of an action.

By Definition 3.4 the equivalence classes of R on S are

$$[s]_R = \{s \in S \mid {F_{m\leq\overrightarrow{\alpha}\leq n}}^{\top}(s) = \bullet\} = \{s\} \qquad \text{if } |\overrightarrow{Act}(s)| = 1$$
$$[s]_R = \{s \in S \mid {F_{m\leq\overrightarrow{\alpha}\leq n}}^{\top}(s) = \bot\} \qquad \text{otherwise}$$

Thereby we obtain the view ${\mathcal{M}_{n\leq\overrightarrow{\alpha}}}^{\top}$ for a given MDP $\mathcal{M}$ where $S' = \bigcup_{s\in S}\{[s]_R\} =: S_1 \cup S_2$ where

$$S_1 := \{\{s \in S \mid |\overrightarrow{Act}(s)| = 1\}\} = \{\{s \in S \mid {F_{n\leq\overrightarrow{\alpha}}}^{\top}(s) = \top\}\} \text{ and}$$
$$S_2 := \{\{s\} \mid s \in S, \mid |\overrightarrow{Act}(s)| \neq 1\}\} = \{\{s\} \mid s \in S,\, {F_{n\leq\overrightarrow{\alpha}}}^{\top}(s) = \bullet\}\} = \bigcup_{s\in S\setminus S_1} \{\{s\}\}.$$

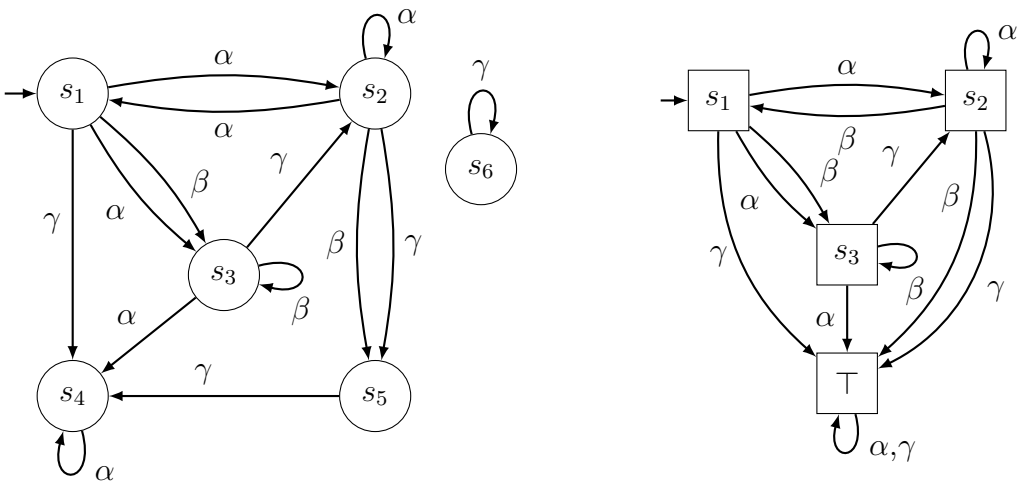


Figure 12: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{|\overrightarrow{Act}(s)|}$ on it (right)

Example 4.10. In Figure 12 we can observe the view ${\mathcal{M}_{|\overrightarrow{Act}(s)|_1}}^{\bot}$ on $\mathcal{M}$. In the simplified representation of an MDP, the states s_5 and s_6 are the only states with only one outgoing action. Hence, these two are grouped and the remaining four states are not grouped with any other state, but shown.

4.1.4 Incoming Actions

Analogous to outgoing actions, views utilzing incoming actions are feasible. Since there is no difference other than the definitions themselves, we only provide the definitions.

Definition 4.11. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view ${\mathcal{M}_{\exists\overleftarrow{\alpha}}}^{\top}$ is defined by its grouping function ${\tilde{F}_{\exists\overleftarrow{\alpha}}}^{\top}$ where $\tilde{F}_{\exists\overleftarrow{\alpha}} : \tilde{S} \to M$ with

$$\tilde{F}_{\exists\overleftarrow{\alpha}}(s) = \begin{cases} \bullet, & \text{if } \exists s' \in S : (s', \alpha, s) \in \mathbf{P} \\ \bot, & \text{otherwise} \end{cases}$$

and $M := \{\bullet, \bot\}$.

Definition 4.12. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view $\mathcal{M}_{n \leq \overleftarrow{\alpha}}{}^{\top}$ is defined by its grouping function $F_{n \leq \overleftarrow{\alpha}}{}^{\top}$ where $\tilde{F}_{n \leq \overleftarrow{\alpha}} : \tilde{S} \to M$ with

$$\tilde{F}_{n \leq \overleftarrow{\alpha}}(s) = \begin{cases} \bullet, & \text{if } \exists s_1, \ldots, s_n \in S : Q_{n \leq \overleftarrow{\alpha}}(s, s_1, \ldots, s_n) \\ \bot, & \text{otherwise} \end{cases}$$

where $M := \{\bullet, \bot\}$, is the minimum amount of times a transition with action α has to be incoming in order to be grouped with the other states and

$$Q_{n \leq \overleftarrow{\alpha}}(s, s_1, \ldots, s_n) := ((s_1, \alpha, s), \ldots, (s_n, \alpha, s) \in \mathbf{P}) \wedge |\{s_1, \ldots, s_n\}| = n$$

is a first order logic predicate.

Definition 4.13. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view $\mathcal{M}_{\overleftarrow{\alpha} \leq n}{}^{\top}$ is defined by its grouping function $F_{\overleftarrow{\alpha} \leq n}{}^{\top}$ where $\tilde{F}_{\overleftarrow{\alpha} \leq n} : \tilde{S} \to M$ with

$$\tilde{F}_{\overleftarrow{\alpha} \leq n}(s) = \begin{cases} \bullet, & \text{if } \forall s_1, \ldots, s_{n+1} \in S : Q_{\overleftarrow{\alpha} \leq n}(s, s_1, \ldots, s_{n+1}) \\ \bot, & \text{otherwise} \end{cases}$$

where $M := \{\bullet, \bot\}$ is the maximum number of times a transition with action α may be incoming and

$$Q_{\overleftarrow{\alpha} \leq n}(s, s_1, \ldots, s_{n+1}) := ((s_1, \alpha, s), \ldots, (s_{n+1}, \alpha, s) \in \mathbf{P}) \implies \bigvee_{\substack{i,j \in \{1,\ldots,n+1\} \\ i<j}} s_i = s_j$$

is a first order logic predicate.

Definition 4.14. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view $\mathcal{M}_{m \leq \overleftarrow{\alpha} \leq n}{}^{\top}$ is defined by its grouping function where $\tilde{F}_{m \leq \overleftarrow{\alpha} \leq n} : \tilde{S} \to M$ with

$$\tilde{F}_{m \leq \overleftarrow{\alpha} \leq n}(s) = \begin{cases} \bullet, & \text{if } \exists s_1, \ldots, s_m \in S : Q_{m \leq \overleftarrow{\alpha}}(s, s_1, \ldots, s_m) \\ & \text{and } \forall s_1, \ldots, s_{n+1} \in S : Q_{\overleftarrow{\alpha} \leq n}(s, s_1, \ldots, s_{n+1}) \\ \bot, & \text{otherwise} \end{cases}$$

where $M := \{\bullet, \bot\}$ and $m, n \in \mathbb{N}$ are the minimum and maximum number of transitions with action α in order for state to be grouped. The predicates $Q_{n \leq \overleftarrow{\alpha}}(s, s_1, \ldots, s_n)$ and $Q_{\overleftarrow{\alpha} \leq n}(s, s_1, \ldots, s_{n+1})$ are the predicates from Definition 4.11 and Definition 4.12 respectively.

Definition 4.15. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view $\mathcal{M}_{\overleftarrow{\alpha}(s)_=}$ is defined by its grouping function $F_{\overleftarrow{\alpha}(s)_=}$ where $\tilde{F}_{\overleftarrow{\alpha}(s)_=} : \tilde{S} \to M$ with

$$s \mapsto \{(\alpha, n) \mid \alpha \in Act, n \text{ is the number of times that } \alpha \text{ is incoming from } s\}$$

and $M := (Act \times \mathbb{N}_0) \cup \{\bullet\}$.

Definition 4.16. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view $\mathcal{M}_{\overleftarrow{\alpha}(s)\approx}$ is defined by its grouping function $F_{\overleftarrow{\alpha}(s)\approx}$ where $\tilde{F}_{\overleftarrow{\alpha}(s)\approx} : \tilde{S} \to M$ with

$$s \mapsto \{\alpha \in Act \mid \exists s' \in S : (s', \alpha, s) \in \mathbf{P}\}$$

and $M := Act \cup \{\bullet\}$.

Definition 4.17. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view $\mathcal{M}_{|\overleftarrow{\alpha}(s)|}$ is defined by its grouping function $F_{|\overleftarrow{\alpha}(s)|}$ where $\tilde{F}^{\top}_{|\overleftarrow{\alpha}(s)|} : \tilde{S} \to M$ with

$$s \mapsto |\{\alpha \in Act \mid \exists s' \in S : (s', \alpha, s) \in \mathbf{P}\}|$$

and $M := \mathbb{N} \cup \{\bullet\}$.

Definition 4.18. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $\alpha \in Act$. The view ${\mathcal{M}_{|\overleftarrow{\alpha}(s)|_1}}^{\perp}$ is defined by its grouping function ${F_{|\overleftarrow{\alpha}(s)|_1}}^{\perp}$ where $\tilde{F}^{\perp}_{|\overleftarrow{\alpha}(s)|_1} : \tilde{S} \to M$ with

$$\tilde{F}^{\perp}_{|\overleftarrow{\alpha}(s)|_1}(s) = \begin{cases} \top, & \text{if } |\overleftarrow{\alpha}(s)| = 1 \\ \bullet, & \text{otherwise} \end{cases}$$

and $M := \mathbb{N} \cup \{\bullet\}$.

4.1.5 Variables

The concept of variables is not part of the definitions of neither TS, MCs or MDPs even though, it is of great relevance in practice. We will introduce and discuss this concept before utilizing it for views.

Since states describe some information about a system at a certain moment of its behavior, the information they carry is usually not atomic, but rather consists of several pieces. For instance when considering a computer program at a given moment during execution all of its currently available variables will have some value, the stack will have a certain structure and the program counter will point to a certain instruction. Systems in general have several properties at a given moment of their behavior, which together constitute the current state of the system. That is, in practice, each state is actually derived from a possible variable assignment. Choosing the respective variable assignment as state representation would result in rather complex state objects. Therefore, the values of the variables are usually stored in a separate data structure and a simple identifier such as an integer represents the actual state. When formalizing this in practice used approach several options come into consideration. Available MDP components such as atomic propositions or the set of states could be used to hold this information. There could be a subset of atomic propositions whose elements declare that a certain variable has a certain value. To achieve unambiguous variable values, care had to be taken that for each variable in each state there was only one atomic proposition declaring its value. The set of states could be used in the sense that the states themselves are complex objects containing the information. We will formalize the variables with a set of variables and an evaluation function induced by the MDP. Although this approach is not based on MDP components, it is practical and convenient to work with.

Definition 4.19. Let $\mathcal{M}$ be an MDP. The set $Var_{\mathcal{M}}$ is called *variables* (of $\mathcal{M}$). It contains all variables induced by $\mathcal{M}$.

Definition 4.20. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and M be an arbitrary set. The by im induced function $VarEval_{\mathcal{M}} : S \times Var_{\mathcal{M}} \to M$ is called *variable evaluation function.*

For this thesis $Var_{\mathcal{M}}$ refers to the set of variables declared in the PRISM file and $VarEval_{\mathcal{M}}$ to their values. When we speak about the value of a variable in a state we refer to the image of $VarEval_{\mathcal{M}}$ for that state and variable. The set M is arbitrary so that arbitrary values can be assigned to a variable. Speaking in terms of computer science and programming this loosens as an example the restriction of only being able to assign numbers and no booleans. The most apparent idea for a view utilizing variables is to group states that meet some requieent regarding the values of the variables.

Definition 4.21. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$, $x \in Var_{\mathcal{M}}$ and $a \in VarEval_{\mathcal{M}}[S, Var_{\mathcal{M}}]$. The view $\mathcal{M}_{x=a}$ is defined by its grouping function ${F_{x=a}}^{\top}$ where $\tilde{F}^{\top}_{x=a} : \tilde{S} \to M$ with

$$\tilde{F}^{\top}_{x=a}(s) = \begin{cases} \bullet, & \text{if } VarEval_{\mathcal{M}}(s, x) = a \\ \bot, & \text{otherwise} \end{cases}$$

where $M := \{\bullet, \bot\}$.

The view $\mathcal{M}_{x=a}$ shows states where a given variable has a given value. By Definition 3.4 the equivalence classes of R on S are

$$\begin{aligned} [s]_R &= \{s \in S \mid {F_{x=a}}^{\top}(s) = \bullet\} = \{s\} && \text{if } VarEval_{\mathcal{M}}\ (s,\ x) = a \\ [s]_R &= \{s \in S \mid {F_{x=a}}^{\top}(s) = \bot\} && \text{otherwise} \end{aligned}$$

The set of states S' of $\mathcal{M}_{x=a}$ is the union of the equivalence classes of R. It is $S' = \bigcup_{s \in S} [s]_R =: S_1 \cup S_2$ where

$$\begin{aligned} S_1 &:= \{\{s\} \mid s \in S,\ VarEval_{\mathcal{M}}(s, x) = a\} \\ &= \{\{s\} \mid s \in S,\ {F_{x=a}}^{\top}(s) = \bullet\} = \bigcup_{s \in S \setminus S_2} \{\{s\}\} \\ S_2 &:= \{\{s \in S \mid VarEval_{\mathcal{M}}(s, x) \neq a\}\} = \{\{s \in S \mid {F_{x=a}}^{\top}(s) = \bot\}\}. \end{aligned}$$

Example 4.11. In Figure 13 we can observe the view of $\mathcal{M}_{x=a}(y = 1)$ on $\mathcal{M}$, in which states with $y = 1$ are not grouped, but all the remaining ones.

If states are to be grouped with a requirement that multiple variables be equal to or not equal to specified values, this can be accomplished by using parallel composition. For even more flexibility, a view may be employed, which also allows a combination of requirements on variables in a disjunctive manner. To extend this idea to its full potential, we will define a view that allows requirements to be

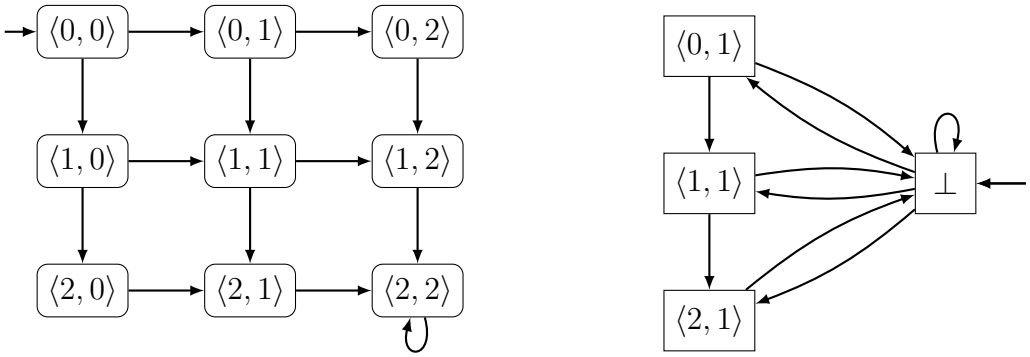


Figure 13: Simplified representations of $\mathcal{M}$ with the state set $S = \{\langle x, y\rangle \mid x, y \in \{0, 1, 2\}\}$ (left) and the view $\mathcal{M}_{x=a}(y = 1)$ on it (right)

expressed using a disjunctive normal formal (DNF). To formalize this view more efficiently, we will write $x_{s,i}$, short for $VarEval_{\mathcal{M}}(s, x_i)$, where $x_i \in Var_{\mathcal{M}}$ and $i \in \mathbb{N}$. We define the symbol $\doteq$ to be an element of the set $\{=, \neq\}$. That is, whenever it is used, each time it is written, it is a substitute for either the symbol $=$ or $\neq$. It allows to write one symbol whenever $=$ and $\neq$ could or should be possible. Furthermore, for this context we consider $(x_{s,i} = a)$ as a literal and $(x_{s,i} \neq a)$ as its negation. We write $(x_{s,i} \doteq a)$ for a literal that may or may not be negated.

Definition 4.22. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and

$$\begin{aligned} c(s) =& ((x_{s,i_1} \doteq a_{i_1}) \wedge \cdots \wedge (x_{s,i_{l_1}} \doteq a_{i_{l_1}})) \vee \\ & ((x_{s,i_{l_1+1}} \doteq a_{i_{l_1+1}}) \wedge \cdots \wedge (x_{s,i_{l_2}} \doteq a_{i_{l_2}})) \vee \\ & \cdots \\ & ((x_{s,i_{(l_{m-1})+1}} \doteq a_{i_{(l_{m-1})+1}}) \wedge \cdots \wedge (x_{s,i_{l_m}} \doteq a_{i_{l_m}})) \end{aligned}$$

proposition logical formula in disjunctive normal form where

- $x_{s,i_k}, a_{i_k} \in VarEval_{\mathcal{M}}[S, Var_{\mathcal{M}}]$ with $i_k \in \mathbb{N}$ and $k \in \{1, \ldots, l_m\}$
- $l_1 < l_2 < \cdots < l_m$ are natural numbers and m is the number of clauses in $c(s)$

The view $\mathcal{M}_{VarDNF}{}^{\top}$ is defined by its grouping function $F_{VarDNF}{}^{\top}$ where $\tilde{F}^{\top}_{VarDNF} : \tilde{S} \to M$ with

$$\tilde{F}^{\top}_{VarDNF}(s) = \begin{cases} \bullet, & \text{if } d(s) \text{ is true} \\ \bot, & \text{otherwise} \end{cases}$$

where $M := \{\top, \bullet\}$.

The DNF allows us to specify a constraint on variables in a disjunctive normal form. States are mapped to the same value depending on whether they satisfy this constraint. By Definition 3.4 the equivalence classes of R on S are

$$\begin{aligned} [s]_R &= \{s \in S \mid F_{VarDNF}{}^{\top}(s) = \bullet\} = \{s\} && \text{if } d(s) \text{ true} \\ [s]_R &= \{s \in S \mid F_{VarDNF}{}^{\top}(s) = \bot\} && \text{otherwise} \end{aligned}$$

The set of states S' of ${\mathcal{M}_{VarDNF}}^\top$ is the union of the equivalence classes of R. It is $S' = \bigcup_{s \in S} [s]_R =: S_1 \cup S_2$ where

$$S_1 := \{\{s\} \mid s \in S,\, d(s) \text{ true}\} = \{\{s\} \mid s \in S,\, {F_{VarDNF}}^\top(s) = \bullet\} = \bigcup_{s \in S \setminus S_2} \{\{s\}\}$$

$$S_2 := \{\{s \in S \mid d(s) \text{ false}\}\} = \{\{s \in S \mid {F_{VarDNF}}^\top(s) = \bot\}\}.$$

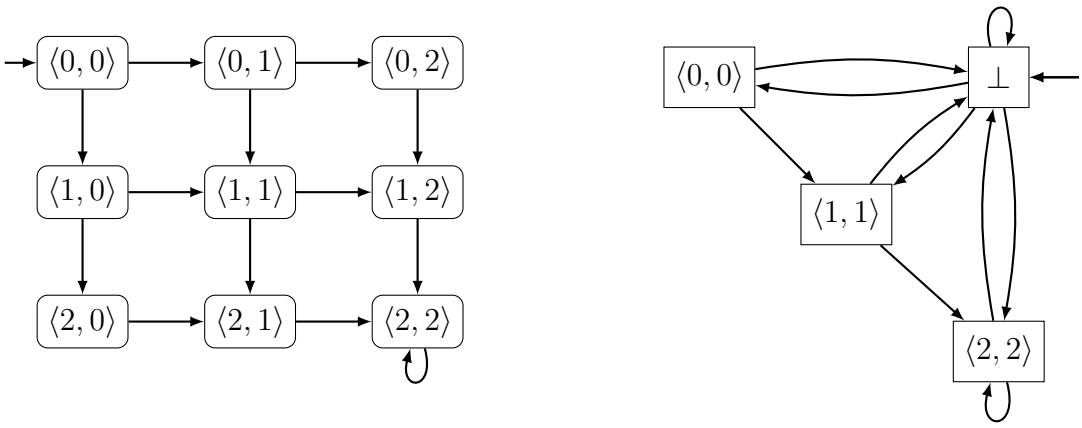


Figure 14: Simplified representations of $\mathcal{M}$ $S = \{\langle x, y\rangle \mid x, y \in \{0, 1, 2\}\}$ (left) and the view ${\mathcal{M}_{VarDNF}}^\top(d(s))$ where $d(s) = ((x = 0) \wedge (y = 0)) \vee ((x = 1) \wedge (y = 1)) \vee ((x = 2) \wedge (y = 2))$ on it (right)

Similarly, a view based on a conjunctive normal formal can be defined that may be more convenient, depending on the requirement/restriction.

Definition 4.23. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and

$$\begin{aligned} c(s) =& ((x_{s,i_1} \doteqdot a_{i_1}) \vee \cdots \vee (x_{s,i_{l_1}} \doteqdot a_{i_{l_1}})) \wedge \\ & ((x_{s,i_{l_1+1}} \doteqdot a_{i_{l_1+1}}) \vee \cdots \vee (x_{s,i_{l_2}} \doteqdot a_{i_{l_2}})) \wedge \\ & \cdots \\ & ((x_{s,i_{(l_{m-1})+1}} \doteqdot a_{i_{(l_{m-1})+1}}) \vee \cdots \vee (x_{s,i_{l_m}} \doteqdot a_{i_{l_m}})) \end{aligned}$$

proposition logical formula in disjunctive normal form where

- $x_{s,i_k}, a_{i_k} \in VarEval_{\mathcal{M}}[S, Var_{\mathcal{M}}]$ with $i_k \in \mathbb{N}$ and $k \in \{1, \ldots, l_m\}$
- $l_1 < l_2 < \cdots < l_m$ are natural numbers and m is the number of clauses in $c(s)$

The view ${\mathcal{M}_{VarCNF}}^\top$ is defined by its grouping function ${F_{VarCNF}}^\top$ where $\tilde{F}^\top_{VarCNF} : \tilde{S} \to M$ with

$$\tilde{F}^\top_{VarCNF}(s) = \begin{cases} \bullet, & \text{if } c(s) \text{ is true} \\ \bot, & \text{otherwise} \end{cases}$$

where $M := \{\bullet, \bot\}$.

The only difference between the view ${\mathcal{M}_{VarCNF}}^{\top}$ and the view ${\mathcal{M}_{VarDNF}}^{\top}$ is whether the respective formulas are in conjunctive or disjunctive normal form. Since any formula in conjunctive normal form can be transformed into a formula in disjunctive normal form, and vice versa [21, Theorem 4.12], neither of the views can perform an action that the other cannot. Thus there is no difference in expressiveness, but there may be a difference in size. Therefore, both views have been implemented and formalized. For finite MDPs they provide an interface to perform any possible grouping on the state set using parameter values. Equality, equivalence classes, and the new state set behave and are constructed analogously to the view ${\mathcal{M}_{VarDNF}}^{\top}$.

The views discussed above reduce the MDP in a very precise but also manual manner, because it not only dictates the variable but also its value. A more general approach is to specify only the variable but not its value. This way states will be grouped that have the same value for that variable, but without any specification about the value for that variable.

Definition 4.24. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $x \in Var_{\mathcal{M}}$. The view $\mathcal{M}_{Var:a}$ is defined by its grouping function $F_{Var:a}$ where $\tilde{F}_{Var:a} : \tilde{S} \to M$ with

$$s \mapsto (x, VarEval_{\mathcal{M}}(s, x))$$

and $M := \{\{x\} \times VarEval_{\mathcal{M}}[S, Var_{\mathcal{M}}]\} \cup \{\bullet\}$.

This grouping function maps each state to a pair containing the given variable and its value in that state. The variable is included to enable composition with several $\mathcal{M}_{Var:a}$. Hence for $s_1, s_2 \in S$ it is $F_{Var:a}(s_1) = F_{Var:a}(s_2)$ if and only if they are mapped to the same value $a \in M$. Hence the equivalence classes of R are

$$[s]_R = \{s' \in S \mid VarEval_{\mathcal{M}}(s') = VarEval_{\mathcal{M}}(s)\}.$$

By Definition 3.5 the set $S' := \bigcup_{s \in S} [s]_R$ is the set of states of $\mathcal{M}_{Var:a}$.

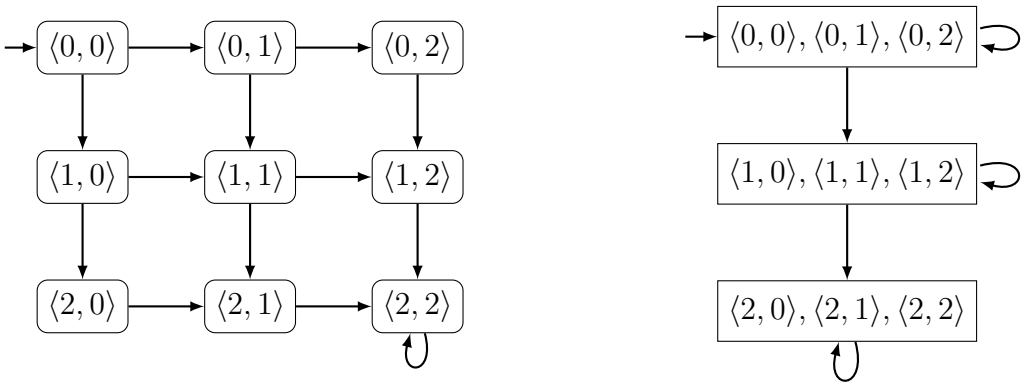


Figure 15: Simplified representations of $\mathcal{M}$ $S = \{\langle x, y\rangle \mid x, y \in \{0, 1, 2\}\}$ (left) and the view $\mathcal{M}_{Var:a}(x)$ on it (right).

Example 4.12. In Figure 15 we can observe the view of $\mathcal{M}_{Var:a}$ on $\mathcal{M}$. States are grouped in which x has the same value. It is $x \in \{0, 1, 2\}$. The states

$\langle 0,0\rangle, \langle 0,1\rangle, \langle 0,2\rangle$ are mapped to the pair $(x,0)$, because in all of them it is $x = 0$. In the same way $\langle 1,0\rangle, \langle 1,1\rangle, \langle 1,2\rangle$ are grouped due to $x = 1$ and $\langle 2,0\rangle, \langle 2,1\rangle, \langle 2,2\rangle$ due to $x = 2$.

4.1.6 Property Values and Model Checking Results

Apart from direct and indirect components of an MDP there are other values associated with states. These can be given by a reward structure or be results of model checking. A reward structure simply assigns values to each state. Model checking results are also available per state. We abstract from the exact type of value or its origin and formalize this notion with an arbitrary function that assigns a value to each state and utilizes it to define a view. Note that there can be multiple such functions (e.g., a reward structure and multiple model checking results) for an MDP.

Definition 4.25. The function $f : S \to \mathbb{R}$ is called *property function*. It maps each state to a property value.

Definition 4.26. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$, $r \in \mathbb{R}$ a granularity, $f : S \to \mathbb{R}$ a property function given and $Z = \bigcup_{i\in\mathbb{Z}} Z_i$ an by r induced partition on $\mathbb{R}$ where

$$Z_i = \begin{cases} [i \cdot r - \frac{r}{2}, i \cdot r + \frac{r}{2}), & \text{if } i > 0 \\ (i \cdot r - \frac{r}{2}, i \cdot r + \frac{r}{2}), & \text{if } i = 0 \\ (i \cdot r - \frac{r}{2}, i \cdot r + \frac{r}{2}] & \text{if } i < 0 \end{cases}$$

The view $\mathcal{M}_f$ is defined by its grouping function F_f where $\tilde{F}_f : \tilde{S} \to M$ with

$$s \mapsto r \cdot i \qquad \text{where } f(s) \in Z_i \in Z$$

and $M = \mathbb{R} \cup \{\bullet\}$.

The smaller the granularity r, the more elements the partition has, and vice versa. In the implementation, for each state s, the value $r \cdot i$ of $f(s)$ is determined directly by rounding. The formalization as in the definition above was chosen because it clearly formalizes the rounding behavior. Two states s_1, s_2 are grouped if $F_f(s_1) = F_f(s_2)$. Thus, the resulting equivalence classes are

$$\begin{aligned} [s]_R &= \{s' \in S \mid F_f(s') = F_f(s) =: r \cdot i\} \\ &= \{s' \in S \mid f(s), f(s') \in Z_i\} \end{aligned}$$

By Definition 3.5 the set $S' := \bigcup_{s\in S} [s]_R$ is the set of states of $\mathcal{M}_f$.

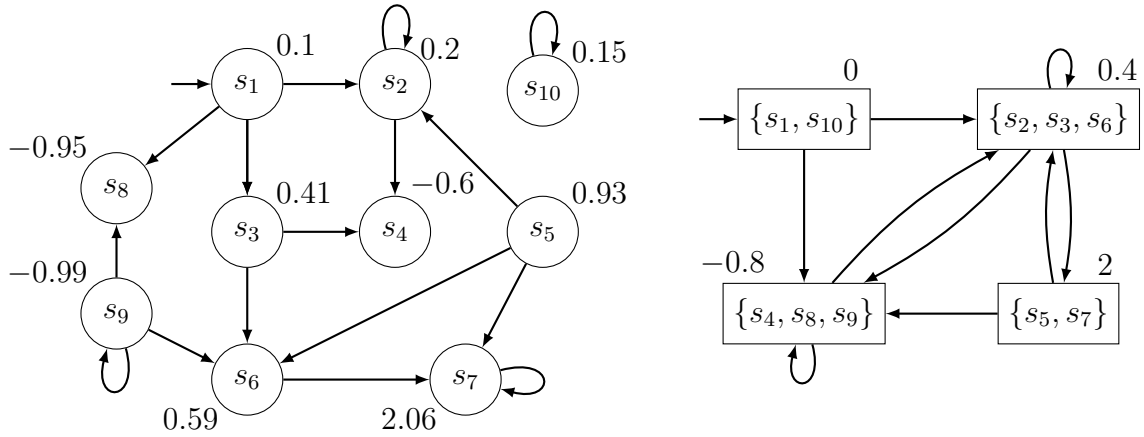


Figure 16: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_f(0.4)$ on it (right)

Example 4.13. In Figure 16 we observe the view $\mathcal{M}_f(0.4)$ on $\mathcal{M}$. The by $r = 0.4$ partition is $Z = \bigcup_{i\in\mathbb{Z}} Z_i$ where

$$Z_i = \begin{cases} [i \cdot 0.4 - 0.2, i \cdot 0.4 + 0.2) = [0.4i - 0.2, 0.4i + 0.2), & \text{if } i > 0 \\ (i \cdot 0.4 - 0.2, i \cdot 0.4 + 0.2) = (-0.2, 0.2), & \text{if } i = 0 \\ (i \cdot 0.4 - 0.2, i \cdot 0.4 + 0.2] = (0.4i - 0.2, 0.4i + 0.2], & \text{if } i < 0 \end{cases}$$

It is

$$\begin{aligned} f(s_4) = -0.6, f(s_8) = -0.95, f(s_9) = -0.99 \in Z_{-2} &= [-1.8, -2.2) \\ f(s_1) = 0.1, f(s_{10}) = 0.15 \in Z_0 &= (-0.2, 0.2) \\ f(s_2) = 0.2, f(s_3) = 0.41, f(s_6) = 0.59 \in Z_1 &= [0.2, 0.6) \\ f(s_5) = 0.93, f(s_7) = 2.06 \in Z_5 &= [1.8, 2.2) \end{aligned}$$

Hence it is

$$\begin{aligned} F_f(s_4) = F_f(s_8) = F_f(s_9) = 0.4 \cdot (-2) &= -0.8 \\ F_f(s_1) = F_f(s_{10}) = 0.4 \cdot 0 &= 0 \\ F_f(s_2) = F_f(s_3) = F_f(s_6) = 0.4 \cdot 1 &= 0.4 \\ F_f(s_5) = F_f(s_7) = 0.4 \cdot 5 &= 2 \end{aligned}$$

and states are grouped as displayed in the simplified representation of $\mathcal{M}_f$.

4.2 Views Utilizing the MDP Graphstructure

4.2.1 Distance

Looking at distances in a graph can be very helpful to get an overview of a graph. Likewise it helps a lot with understanding the structure of an MDP. In order to

consider the distance between states in a an MDP we need to formalize it. For this we will introduce *paths*, which are conceptually very similar to execution fragments [1, Def. 2.4.].

Definition 4.27. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP. A (simple and finite) *path* P is a sequence $(s_1, \alpha_1, s_2, \alpha_2, \ldots, \alpha_n, s_{n+1})$ alternating between states and actions where $n \in \mathbb{N}, \{s_1, \ldots, s_n\} \subseteq S$ is a set of distinct states, $\{\alpha_1, \ldots, \alpha_n\} \subseteq Act$ and for all $i \in \{1, \ldots, n\}$ it is $(s_i, \alpha_i, s_{i+1}) \in \mathbf{P}$.
It is $first(P) := s_1$ and $last(P) := s_n$ and $P_{\mathcal{M}}$ the set of all paths in $\mathcal{M}$.

Definition 4.28. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP. A (simple and finite) *undirected path* $\bar{P}$ is a sequence $(s_1, \alpha_1, s_2, \alpha_2, \ldots, \alpha_n, s_{n+1})$ alternating between states and actions where $n \in \mathbb{N}, \{s_1, \ldots, s_n\} \subseteq S$ is a set of distinct states, $\{\alpha_1, \ldots, \alpha_n\} \subseteq Act$ and for all $i \in \{1, \ldots, n\}$ it is $(s_i, \alpha_i, s_{i+1}) \in \mathbf{P}$ or $(s_{i+1}, \alpha_i, s_i) \in \mathbf{P}$.
It is $first(\bar{P}) := s_1$ and $last(\bar{P}) := s_n$ and $\bar{P}_{\mathcal{M}}$ the set of all undirected paths in $\mathcal{M}$.

Definition 4.29. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and $P = (s_1, \alpha_1, s_2, \alpha_2, \ldots, \alpha_n, s_{n+1})$ be a path in $\mathcal{M}$. The number $n =: len(P)$ is called *length* of the path P. It corresponds to the number of transitions taken.

Definition 4.30. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP. The *distance* from a set of states $S_1 \subseteq S$ to a disjoint set of states $S_2 \subseteq S$ is length of the shortest path from a state $s_1 \in S_1$ to a $s_2 \in S_2$ or infinity if no such path exists. That is, if $S_1 \cap S_2 = \emptyset$ it is

$$dist(\mathcal{M}, S_1, S_2) := \begin{cases} \min\{len(P) \mid P \in P_{S_1 \to S_2}\}, & \text{if } P_{S_1 \to S_2} \neq \emptyset \\ \infty, & \text{otherwise} \end{cases}$$

where $P_{S_1 \to S_2} := \{P \in P_{\mathcal{M}} \mid first(P) \in S_1,\ last(P) \in S_2\}$.
If $S_1 \cap S_2 \neq \emptyset$, it is $dist(\mathcal{M}, S_1, S_2) := 0$.

Definition 4.31. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP. The *undirected distance* between disjoint $S_1, S_2 \subseteq S$ is the length of the shortest undirected path from a state $s_1 \in S_1$ to a state $s_2 \in S_2$ or infinity if no such path exists. That is, if $S_1 \cap S_2 = \emptyset$ it is

$$\overline{dist}(\mathcal{M}, S_1, S_2) := \begin{cases} \min\{len(\bar{P}) \mid \bar{P} \in P_{S_1 - S_2}\}, & \text{if } P_{S_1 - S_2} \neq \emptyset \\ \infty, & \text{otherwise} \end{cases}$$

where $P_{S_1 - S_2} := \{\bar{P} \in \bar{P}_{\mathcal{M}} \mid first(\bar{P}) \in S_1,\ last(\bar{P}) \in S_2\}$.
If $S_1 \cap S_2 \neq \emptyset$, it is $\overline{dist}(\mathcal{M}, S_1, S_2) := 0$.

For a given MDP $\mathcal{M}$ with state set S and $s \in S, \tilde{S} \subseteq S$ we write $dist(\mathcal{M}, \tilde{S}, s)$ for $dist(\mathcal{M}, \tilde{S}, \{s\})$ and $dist(\mathcal{M}, s, \tilde{S})$ for $dist(\mathcal{M}, \{s\}, \tilde{S})$. We do the same for $\overline{dist}$. The symbol ∞ is used to indicate that there is no path and therefore the distance is "infinite". We will compare against the symbol with equality to rule out whether there is a path from one set of states to the other. To apply the notion of distance to the entire MDP graph with a view, it must be done in a way where a distance

is computed for each state. Therefore, some set of states is needed, from which the distance for the remaining states is measured. Such a set could be for example the set of initial states or any other set of interest. In each view using distance it is denoted by $\bar{S}$. We will also consider a granularity $n \in \mathbb{N}$. The following view groups states that are reachable with the same number of transitions $\bar{S} \subseteq S$ considering the granularity n.

Definition 4.32. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\bar{S} \subseteq \tilde{S} \subseteq S$ and $n \in \mathbb{N}$. The view $\mathcal{M}_{\overrightarrow{dist}}$ is defined by its grouping function $F_{\overrightarrow{dist}}$ where $\tilde{F}_{\overrightarrow{dist}} : \tilde{S} \to M$ with

$$F_{\overrightarrow{dist}}(s) = \begin{cases} d - (d \bmod n), & \text{if } d \neq \infty \\ \infty, & \text{otherwise} \end{cases}$$

where $d = dist(\mathcal{M}, \bar{S}, s)$ and $M = \mathbb{N} \cup \{\infty\} \cup \{\bullet\}$.

The number n from the definition is the granularity of the distance cluster. For example, if the granularity is two ($n = 2$), the expression $d-(d \bmod n)$ evaluates to $0 - 0 = 0$ for $d = 0$, to $1 - 1 = 0$ for $d = 1$, to $2 - 0 = 2$ for $d = 2$, and to $3 - 1 = 2$ for $d = 3$, where the subtrahend is d mod 2. We see that $F_{\overrightarrow{dist}}$ maps $dist(\mathcal{M}, \tilde{S}, s)$, if not infinite, to the next smaller natural number divisible by n, which means that the granularity causes $F_{\overrightarrow{dist}}(s)$ to be a multiple of n.

By Definition 3.4 the equivalence classes of R on S are

$$\begin{aligned} &[s]_R = \{s' \in S \mid F_{\overrightarrow{dist}}(s') = F_{\overrightarrow{dist}}(s) = \tilde{d} \in \mathbb{N}\} && \text{if } dist(\mathcal{M}, \bar{S}, s) \neq \infty \\ &[s]_R = \{s \in S \mid F_{\overrightarrow{dist}}(s) = \infty\} && \text{otherwise} \end{aligned}$$

Thereby the obtained set of states S' of $\mathcal{M}_{x=a}$ is the union of the equivalence classes of R. It is $S' = \bigcup_{s \in S} [s]_R =: S_1 \cup S_2$ where

$$\begin{aligned} S_1 &:= \{\{s \in S \mid dist(\mathcal{M}, \bar{S}, s) = d - (d \bmod n) =: \tilde{d} \neq \infty\} \mid \tilde{d} \in F_{\overrightarrow{dist}}[S]\} \\ &= \{\{s \in S \mid F_{\overrightarrow{dist}}(\mathcal{M}, \bar{S}, s) = d - (d \bmod n) =: \tilde{d} \neq \infty\} \mid \tilde{d} \in F_{\overrightarrow{dist}}[S]\} \text{ and} \\ S_2 &:= \{\{s \in S \mid dist(\mathcal{M}, \bar{S}, s) = \infty\}\} = \{\{s \in S \mid F_{\overrightarrow{dist}}(s) = \infty\}\}. \end{aligned}$$

Example 4.14. In Figure 17 we can observe the views $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 1)$ (middle) and $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 2)$ (right) on $\mathcal{M}$. It is

$$\begin{aligned} &dist(\mathcal{M}, \{s_1, s_6\}, s_i) = 0 \text{ for } i \in \{1, 6\}, && dist(\mathcal{M}, \{s_1, s_6\}, s_i) = 1 \text{ for } i \in \{2, 3, 7, 8\}, \\ &dist(\mathcal{M}, \{s_1, s_6\}, s_i) = 2 \text{ for } i = 4, && dist(\mathcal{M}, \{s_1, s_6\}, s_i) = \infty \text{ for } i \in \{5, 9, 10\} \end{aligned}$$

This is how we obtain, the set of states of $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 1)$ as displayed in the figure. For $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 2)$ (right) the states $\{s_1, s_6\}$ and $\{s_2, s_3, s_7, s_8\}$ of $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 1)$ (middle) merge into one, as $F_{\overrightarrow{dist}}(s_i) = 0-(0 \bmod 2) = 0 = 1-(1 \bmod 2) = F_{\overrightarrow{dist}}(s_j)$ for $j \in \{1, 6\}$ and $i \in \{2, 3, 7, 8\}$. displayed in the figure. For $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 2)$ (right) the states $\{s_1, s_6\}$ and $\{s_2, s_3, s_7, s_8\}$ of $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 1)$ (middle) merge into one, as $F_{\overrightarrow{dist}}(s_i) = 0 - (0 \bmod 2) = 0 = 1 - (1 \bmod 2) = F_{\overrightarrow{dist}}(s_j)$ for $j \in \{1, 6\}$ and $i \in \{2, 3, 7, 8\}$.

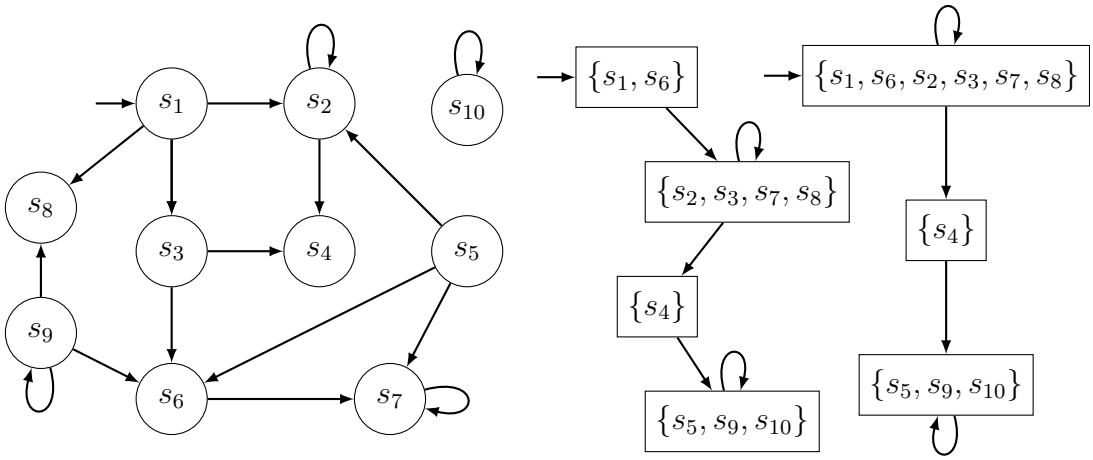


Figure 17: Simplified representations of $\mathcal{M}$ (left), the views $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 1)$ (middle) and $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 2)$ on it (right).

The view $\mathcal{M}_{\overrightarrow{dist}}$ groups states that are reachable with the same number of transitions from a given set of states. It can be used to give a perspective on the structure of an MDP, but cannot be used to deduce how a state or set of states was reached. How a given state was reached is a common case of interest, since there are states in a system that should be reached and others that should not or even must not be reached. To provide an aid to investigating how a state was reached, we define a view that groups states that have the same distance from a given state when traversing transitions backwards.

Definition 4.33. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\bar{S} \subseteq \tilde{S} \subseteq S$ and $n \in \mathbb{N}$. The view $\mathcal{M}_{\overleftarrow{dist}}$ is defined by its grouping function $F_{\overleftarrow{dist}}$ where $\tilde{F}_{\overleftarrow{dist}} : \tilde{S} \to M$ with

$$F_{\overleftarrow{dist}}(s) = \begin{cases} d - (d \bmod n), & \text{if } d \neq \infty \\ \infty, & \text{otherwise} \end{cases}$$

where $d = dist(\mathcal{M}, s, \bar{S})$ and $M = \mathbb{N} \cup \{\infty\} \cup \{\bullet\}$.

Note that the only difference between $\mathcal{M}_{\overleftarrow{dist}}$ and $\mathcal{M}_{\overrightarrow{dist}}$ is that it is $dist(\mathcal{M}, s, \bar{S})$ instead of $dist(\mathcal{M}, \bar{S}, s)$. By definition 4.30 this declares that paths from s to $\bar{S}$ are considered instead of paths from $\bar{S}$ to s. Thus, the distance from the set $\bar{S}$ is measured by traversing transitions backwards. This causes states that reach $\bar{S}$ with the same number of transitions to be grouped together. Nevertheless, on a formal level, equality, equivalence classes, and the new state set behave and are constructed analogously to the view $\mathcal{M}_{\overrightarrow{dist}}$.

For cases where the direction of transitions is of no further importance we define a view $\mathcal{M}_{\overline{dist}}$ utilizing $\overline{dist}$. This, causes that the direction of that transitions regularly have is ignored.

Definition 4.34. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\bar{S} \subseteq \tilde{S} \subseteq S$ and $n \in \mathbb{N}$. The view $\mathcal{M}_{\overline{dist}}$ is defined by its grouping function $F_{\overline{dist}}$ where $\tilde{F}_{\overline{dist}} : \tilde{S} \to M$

with

$$F_{\overline{dist}}(s) = \begin{cases} d - (d \bmod n), & \text{if } d \neq \infty \\ \infty, & \text{otherwise} \end{cases}$$

where $d = \overline{dist}(\mathcal{M}, \bar{S}, s)$ and $M = \mathbb{N} \cup \{\infty\} \cup \{\bullet\}$.

The only difference of $\mathcal{M}_{\overline{dist}}$ to $\mathcal{M}_{\overrightarrow{dist}}$ is that $\overline{dist}$ is used instead of $dist$. Hence, we omit explaining behavior and construction of equality, equivalence classes and the new state set.

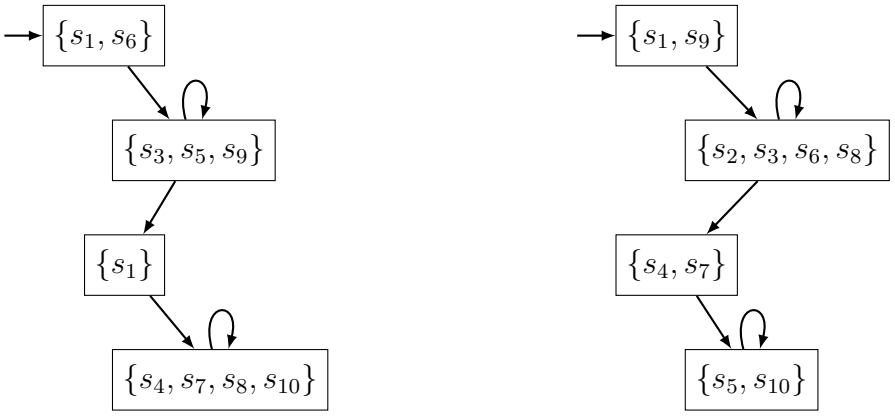


Figure 18: Simplified representations of the views $\mathcal{M}_{\overleftarrow{dist}}(\{s_1, s_6\}, 1)$ (middle) and $\mathcal{M}_{\overline{dist}}(\{s_1, s_9\}, 1)$ (right) on the $\mathcal{M}$ from Figure 17.

In the implementation, a simple breadth-first search is used to determine the distance values from a given state set to each state. There is no need for an algorithm like Dijkstra, since the weights on the edges are probabilities, not values that can be interpreted as cost or distance. The implementation ensures that for each state s there is a pair (s, d) in $distance(\mathcal{M}, \tilde{S}, n)$, i.e. ∞ if there is no path to a given state. The algorithm in Java code using functional programming can be found in the appendix (**??**). It implements all three views based on distance in a single java class.

4.2.2 Cycles

A cycle is a structure that can exist in every graph. The concept of cycles is not limited to MDPs or related models used in model checking. The thesis aims to discuss views that utilize domain specific knowledge or general concepts that are of special relevance when exploring an MDP. In general cycles are worth exploring because they may lead to unwanted infinite results in model checking. Additionally, when seeking views, the main cycle idea can be adjusted to include MDPs characteristics. To formalize views based on cycles, we need to formalize the cycle concept itself. We'll define it in a way that suits our specific domain the best.

Definition 4.35. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP. A (simple) *cycle* C in $\mathcal{M}$ is a sequence $(s_0, \alpha_0, s_1, \alpha_1, \ldots, \alpha_{n-1}, s_0)$ alternating between states and actions

where $n \in \mathbb{N}, \{s_0, \ldots, s_{n-1}\} \subseteq S$ is a set of distinct states, $\{\alpha_0, \ldots, \alpha_{n-1}\} \subseteq Act$ and for all $i \in \{0, \ldots, n-1\}$ it is $(s_i, \alpha_i, s_{i+1 \bmod n}) \in \mathbf{P}$.

In the following let $C = (s_0, \alpha_0, s_1, \alpha_1, \ldots, \alpha_{n-1}, s_0)$ be a cycle. For convenience we will write $s \in C$ if the state is contained in the cycle C and $\alpha \in C$ if the action is contained in in the cycle C. In words we will both write a state or action is *on* or *in* the cycle.

Definition 4.36. Let C be an cycle in $\mathcal{M}$ and S be the states set of $\mathcal{M}$. The number $|\{s \in S \mid s \in C\}| =: len(C)$ is called the *length* of a cycle.

Definition 4.37. Let $\mathcal{M}$ be an MDP. The set $C_{\mathcal{M},n} := \{C \mid len(C) \geq n\}$ declares the set of all cycles in $\mathcal{M}$ with a length of at least n.

With the formalization now complete, we will proceed to introduce several views that leverage the concept of cycles. Firstly, we will present views that make use of the overall cycle concept, and secondly, we'll integrate the cycle concept with specific knowledge about the subject. Cycles, in a general sense, are intriguing because unintended cycles can often result in infinite values when checking models. In many instances, these infinite values in model checks are undesired and stem from flawed model construction. Detecting these cycles can be helpful in diagnosing an MDP. Therefore, creating a view that solely identifies existing cycles is achievable.

Definition 4.38. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $n \in \mathbb{N}$. The view $\mathcal{M}_{\exists C}$ is defined by its grouping function $F_{\exists C}$ where $\tilde{F}^{\top}_{\exists C} : \tilde{S} \to M$ with

$$\tilde{F}^{\top}_{\exists C}(s) = \begin{cases} \bullet, & \text{if } \exists C \in C_{\mathcal{M},n} : s \in C \\ \bot, & \text{otherwise} \end{cases}$$

and $M = \{\top, \bullet\}$.

This view groups all states that are contained in a cycle with a length of at least n. Hence, this view can also finds self loops for $n \in \mathbb{N}$. It is to note that the affinity of a state to one or several respective cycles is lost. By Definition 3.4 the equivalence classes of R on S are

$$\begin{aligned} [s]_R &= \{s \in S \mid F_{\exists C}(s) = \bullet\} = \{s\} && \text{if } \exists C \in C_{\mathcal{M},n} : s \in C \\ [s]_R &= \{s \in S \mid F_{\exists C}(s) = \bot\} && \text{otherwise} \end{aligned}$$

The set of states S' of $\mathcal{M}_{\exists C}$ is the union of the equivalence classes of R. It is $S' = \bigcup_{s \in S} [s]_R =: S_1 \cup S_2$ where

$$\begin{aligned} S_1 &:= \{\{s\} \mid s \in S, \text{ if } \exists C \in C_{\mathcal{M},n} : s \in C\} \\ &= \{\{s\} \mid s \in S, F_{\exists C}(s) = \bullet\} = \bigcup_{s \in S \setminus S_2} \{\{s\}\} \\ S_2 &:= \{\{s \in S \mid \text{if } \exists C \in C_{\mathcal{M},n} : s \in C\}\} = \{\{s \in S \mid F_{\exists C}(s) = \bot\}\}. \end{aligned}$$

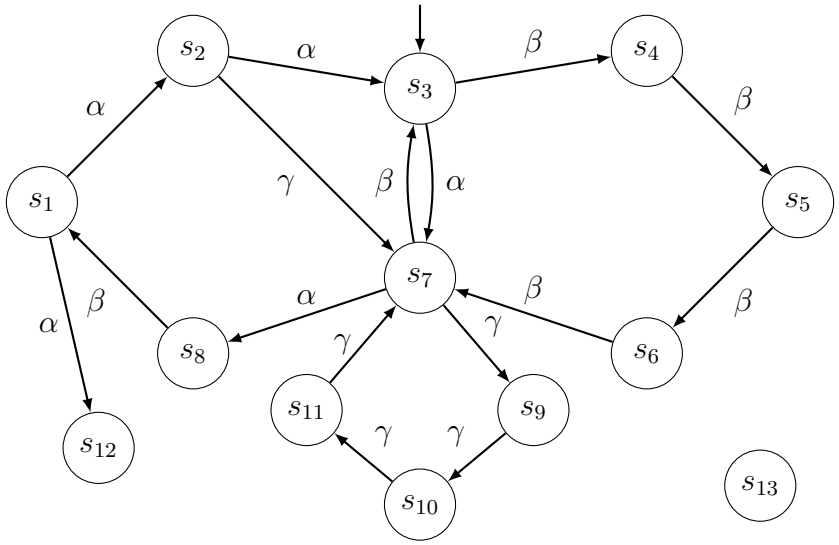


Figure 19: Simplified representation of $\mathcal{M}$

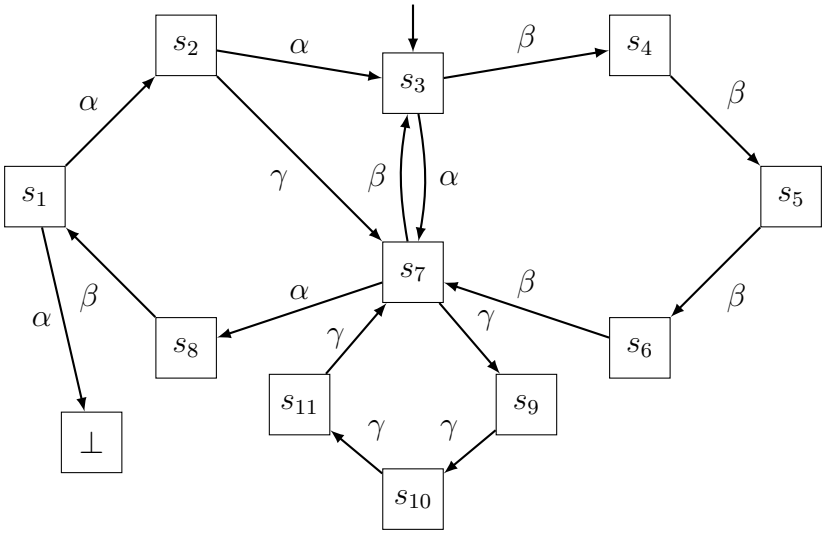


Figure 20: Simplified representations of the view $\mathcal{M}_{\exists C}$ on $\mathcal{M}$ from Figure 19. The states s_{12} and s_{13} are grouped since they are the only states not on a cycle.

Definition 4.39. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $n \in \mathbb{N}$. The view $\mathcal{M}_{\{C_n\}}$ is defined by its grouping function $F_{\{C_n\}}$ where $\tilde{F}_{\{C_n\}} : \tilde{S} \to M$ with

$$s \mapsto \{C \in C_{\mathcal{M},n} \mid s \in C\}$$

and $M = \mathcal{P}(C_{\mathcal{M},n}) \cup \{\bullet\}$.

This view groups states that have the same set of cycles they are contained in. Thus if C_1 and C_2 are distinct cycles and $s_1, s_2 \in C_1$ and $s_1 \in C_2$ but $s_2 \notin C_2$ they are not grouped. It suffices that a state is on one cycle that the other one is not on, in order for the states not being grouped. In graphs with many cycles this can lead to little grouping.

By Definition 3.4 the equivalence classes of R on S are

$$[s]_R = \{s' \in S \mid F_{\{C_n\}}(s') = F_{\{C_n\}}(s) = \{C_1, \ldots, C_n\} \subseteq C_{\mathcal{M},n}\}$$

By Definition 3.5 the set $S' := \bigcup_{s \in S} [s]_R$ is the set of states of $\mathcal{M}_{\{C_n\}}$.

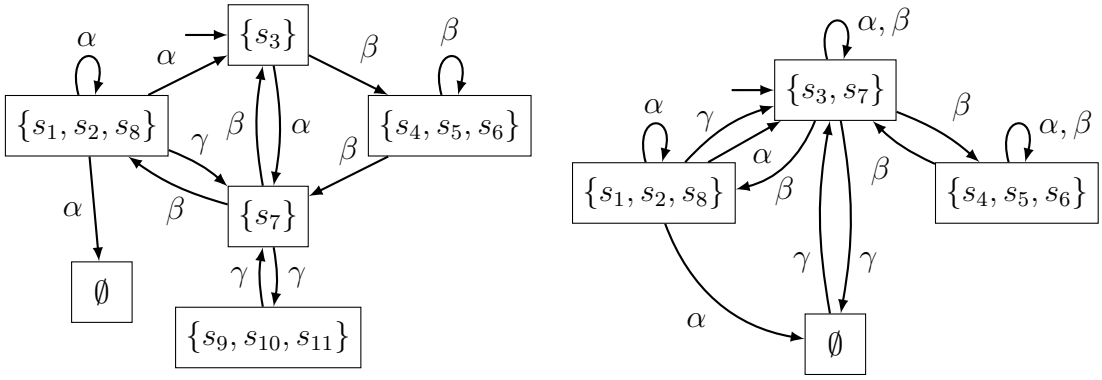


Figure 21: View $\mathcal{M}_{\{C_n\}}(2)$ (left) and view $\mathcal{M}_{\{C_n\}}(5)$ (right) on $\mathcal{M}$ from Figure 19

Example 4.15. In Figure 21 we can observe the views of $\mathcal{M}_{\{C_n\}}(2)$ and $\mathcal{M}_{\{C_n\}}(5)$ on $\mathcal{M}$ from Figure 19. In the simplified representation of an MDP, we find six cycles of size at least two:

$$\begin{aligned}
C_1 &= (s_1, \alpha, s_2, \alpha, s_3, \beta, s_4, \beta, s_5, \beta, s_6, \beta, s_7, \alpha, s_8, \beta, s_1) \\
C_2 &= (s_1, \alpha, s_2, \alpha, s_3, \alpha, s_7, \alpha, s_8, \beta, s_1) \\
C_3 &= (s_3, \beta, s_4, \beta, s_5, \beta, s_6, \beta, s_7, \beta, s_3) \\
C_4 &= (s_7, \gamma, s_9, \gamma, s_{10}, \gamma, s_{11}, \gamma, s_7) \\
C_5 &= (s_1, \alpha, s_2, \gamma, s_7, \alpha, s_8, \beta, s_1) \\
C_6 &= (s_3, \alpha, s_7, \beta, s_3)
\end{aligned}$$

View $\mathcal{M}_{\{C_n\}}(2)$ results as displayed as in the simplified representation on the left because because the following states are contained in exactly these cycles

$$\begin{aligned}
s_1, s_2, s_8 &\text{ in } C_1, C_2, C_5 \\
s_3 &\text{ in } C_1, C_2, C_3, C_5, C_6 \\
s_7 &\text{ in } C_1, C_2, C_3, C_4, C_5, C_6 \\
s_4, s_5, s_6 &\text{ in } C_1, C_3 \\
s_9, s_{10}, s_{11} &\text{ in } C_4
\end{aligned}$$

and s_{12} and s_{13} are contained in no cycle of size at least two, i.e. $F_{\{C_n\}}(s_{12}) = F_{\{C_n\}}(s_{13}) = \emptyset$. For the $\mathcal{M}_{\{C_n\}}(5)$ it is $C_{\mathcal{M},5} = \{C_1, C_2, C_3\}$. Thereby we have the following states being exactly contained in these cycles:

$$\begin{aligned}
s_1, s_2, s_8 &\text{ in } C_1, C_2 \\
s_3 &\text{ in } C_1, C_2, C_3 \\
s_7 &\text{ in } C_1, C_2, C_3 \\
s_4, s_5, s_6 &\text{ in } C_1, C_3
\end{aligned}$$

and $F_{\{C_n\}}(s) = \emptyset$ for all remaining states s.

The view above might be useful when having found cycles to see what cycles specifically exist. It might be interesting to find cycles that consist only of transitions of the same action. The following view accomplishes that.

Definition 4.40. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $n \in \mathbb{N}$. The view $\mathcal{M}_{\{C_{\alpha,n}\}}$ is defined by its grouping function $F_{\{C_{\alpha,n}\}}$ where $\tilde{F}_{\{C_{\alpha,n}\}} : \tilde{S} \to M$ with

$$s \mapsto \{C \in C_{\mathcal{M},n} \mid s \in C, \tilde{\alpha} \in C, \forall \alpha \in C : \alpha = \tilde{\alpha}\}$$

and $M = \mathcal{P}(C_{\mathcal{M},n}) \cup \{\bullet\}$.

The view specializes the view from Definition 4.39 in the sense that it additionally requires for all $C \in F_{\{C_n\}}$ that all actions occurring in the cycle are the same. The reasoning about the equality of the grouping function values, the obtained equivalence classes and the resulting set of states S' of the view is analogous to $\mathcal{M}_{\{C_n\}}$ and thus omitted.

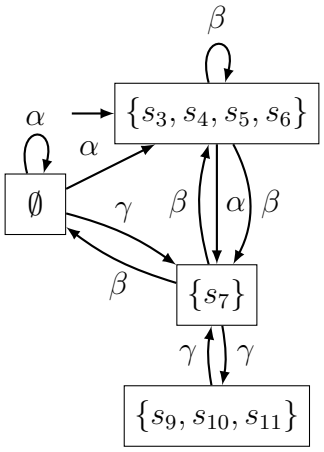


Figure 22: Simplified representation of the view $\mathcal{M}_{\cup\{s\}_C}$ on $\mathcal{M}$ from Figure 19

Example 4.16. In Figure 22 the simplified representation of $\mathcal{M}_{\{C_{\alpha,n}\}}(2)$ on $\mathcal{M}$ from Figure 19 can be observed. From cycles listed in the discussion of Figure 21 only C_3 and C_4 consist of transitions with the same actions, i.e. only $C_3, C_4 \in F_{\{C_{\alpha,n}\}}[S]$ for $n = 2$. The view $\mathcal{M}_{\{C_{\alpha,n}\}}$ in Figure 22 results because the following states are contained in exactly these cycles

$$\begin{aligned} s_3, s_4, s_5, s_6 &\text{ in } C_3 \\ s_7 &\text{ in } C_3, C_4 \\ s_9, s_{10}, s_{11} &\text{ in } C_4 \end{aligned}$$

and for all remaining states S it is $F_{\{C_{\alpha,n}\}}(s) = \emptyset$.

When discussing definition 4.39 we stated that little grouping can occur when mapping on the set of cycles in all of which the state is contained. Hence we provide the definition and implementation of a view that groups states even though their set of cycles is not equal, but there is sufficient similarity in the cycles they are on. This might be useful to find clusters of cycles.

Definition 4.41. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $n \in \mathbb{N}$. The view $\mathcal{M}_{\cup\{s\}_C}$ is defined by its grouping function $F_{\cup\{s\}_C}$ where $\tilde{F}_{\cup\{s\}_C} : \tilde{S} \to M$ with

$$s \mapsto \{\tilde{s} \in S \mid s, \tilde{s} \in C \in C_{\mathcal{M},n}\}$$

and $M = S \cup \{\bullet\}$.

With this view each state s is mapped to the set of states. The set contains all the states from cycles, where the state s is contained. By Definition 3.4 the equivalence classes of R on S are

$$[s]_R = \{s' \in S \mid F_{\{C_n\}}(s') = F_{\{C_n\}}(s) = \{s_1, \ldots, s_n\} \subseteq \{C_1, \ldots, C_n\} \subseteq C_{\mathcal{M},n}\}$$

By Definition 3.5 the set $S' := \bigcup_{s \in S} [s]_R$ is the set of states of $\mathcal{M}_{\{C_n\}}$.

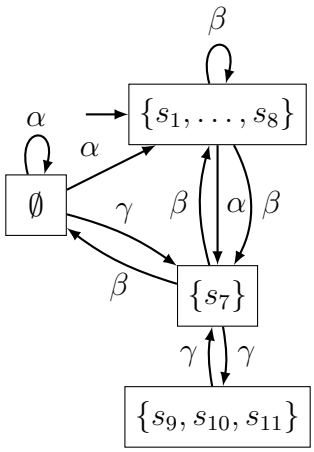


Figure 23: Simplified representation of the view $\mathcal{M}_{\cup\{s\}_C}(2)$ on $\mathcal{M}$ from Figure 19

Example 4.17. In Figure 23 the simplified representation of $\mathcal{M}_{\cup\{s\}_C}(2)$ on $\mathcal{M}$ from Figure 19 can be observed. Since it is $n = 2$ we consider the same cycles as in the discussion of Figure 21. For $i \in \{1, \ldots, 8\}$ it is $F_{\cup\{s\}_C}(s_i) \subseteq \{s_1, \ldots, s_8\}$, because they are all on cycle C_1. For $s \in \{s_1, \ldots, s_8\} \setminus \{s_7\}$ there is no cycle containing any other state than $\{s_1, \ldots, s_8\}$. Thus, for $i \in \{1, \ldots, 8\}$ it is $F_{\cup\{s\}_C}(s_i) = \{s_1, \ldots, s_8\}$. For the state s_7 it is $F_{\cup\{s\}_C}(s_7) = S \setminus \{s_{12}, s_{13}\}$ because of $s_7 \in C_1, C_4$ and s_{12}, s_{13} are contained in no cycle of length at least two. Also because s_{12}, s_{13} are contained in no cycle of length at least two it is $F_{\cup\{s\}_C}(s_{12}) = F_{\cup\{s\}_C}(s_{13}) = \emptyset$. Obviously for s_9, s_{10}, s_{11} it is $F_{\cup\{s\}_C}(s_i) = \{s_7, s_9, s_{10}, s_{11}\}$ because it is only $s_9, s_{10}, s_{11} \in C_4$ as s_7 denies them being on any other (*simple*) cycle. Thereby the grouping results in a view as depicted as simplified representation in Figure 23.

In practice there exist several cycle finding algorithms, to determine the set $C_{\mathcal{M},n_{\mathcal{M},0}}$. The actual implementation relies on algorithms of the Java library jGraphtT [22] namely the Algorithm Szwarcfiter and Lauer [23]. It has a theoretical complexity of $O(V + EC)$. If in $C_{\mathcal{M},n_{\mathcal{M},n}}$ n was greater one, the found cycles are then filtered, retaining those with a minimal length of n.

4.2.3 Strongly Connected Components

Strongly connect components (SCCs) are are interesting when exploring an MDP because they group states where may happen some complex operation. Moreover, finding strongly connected components is way more efficient than finding cycles. Since every cycle also is a strongly connected component finding strongly connected components can be seen as finding a set that contains cycles. Therefore it is feasible to consider a view utilizing strongly connected components. Deviating from most definitions we will define SCC not as a subgraph but only as the set of its nodes. Moreover the definition is written in the terms of an MDP.

Definition 4.42. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and $\tilde{S} \subseteq S$. A set $T \subseteq S$ is called *strongly connected component* if for all $s, s' \in T$ either it holds

$$\exists P \in P_{\mathcal{M}} : first(P) = s \wedge\ last(P) = s'$$

or

$$s = s'.$$

The set of all strongly connected components of $\mathcal{M}$, that contain at least n states, is denoted with $SCC_{\mathcal{M},n}$.

Since the strong connection is an equivalence relation, the SCCs are equivalence classes and hence disjoint. To find SCCs Tarjan's and Sahir's algorithm is the classic [24]. In the implementation an improved variant from Gabow [25] is used supplied by the jGraphtT library.

Definition 4.43. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $n \in \mathbb{N}$. The view $\mathcal{M}_{scc}$ is defined by its grouping function F_{scc} where $\tilde{F}_{scc} : \tilde{S} \to M$ with

$$s \mapsto \{T \in SCC_{\mathcal{M},n} \mid s \in T\}$$

and $M = SCC_{\mathcal{M},n} \cup \{\bullet\}$.

This view groups all states together that are in the same SCC. Note that for all states s, $F_{scc}\ (s)$ is a singleton set because SCCs are disjoint. That is, each state is mapped to the set only containing its strongly connected component or to the empty set if its SCC has less than n elements. In the view $\mathcal{M}_{scc}$ two states $s_1, s_2 \in S$ are grouped if $F_{scc}(s_1) = F_{scc}(s_2) \neq \bullet$. Hence the equivalence classes are:

$$[s]_R = \{s' \in S \mid F_{scc}(s') = F_{scc}(s) =: T \in SCC_{\mathcal{M},n}\} \qquad \text{for } s \in T$$

By Definition 3.5 the set $S' := \bigcup_{s \in S} [s]_R$ is the set of states of $\mathcal{M}_{\{C_n\}}$.

Example 4.18. In Figure 24 the simplified representation of $\mathcal{M}_{scc}(2)$ (right) on $\mathcal{M}$ (left) can be observed.. The SCCs in $\mathcal{M}$ are $\{s_1\}, \{s_2, s_3, s_4, s_5\}, \{s_6, s_7, s_8, s_9\}, \{s_{10}, s_{11}, s_{12}\}$ and $\{s_{13}\}$. Because the SCC of state s_{13} has not cardinality two but one it is mapped to the empty set and would be grouped with any other s whoms strongly connected component has less than two elements.

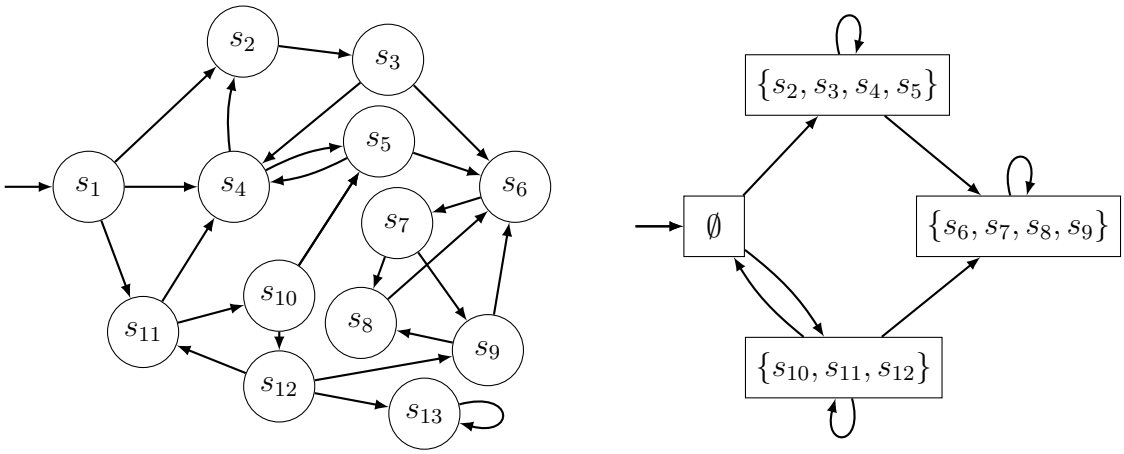


Figure 24: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{scc}(2)$ on it (right)

A special kind of strongly connected components is the the bottom strongly connected component. Thus, they can be interesting for deadlock detection and liveness analysis.

Definition 4.44. Let T be a SCC. A *bottom strongly connected component* (BSCC) is a SCC where it holds that:

$$\forall s \in T : \forall (s, \alpha, s') \in \mathbf{P} : s' \in T$$

That is, from T there is no state reachable outside of T. The set of all bottom strongly connected components of $\mathcal{M}$, that contain at least n states, is denoted with $BSCC_{\mathcal{M},n}$.

Bottom strongly connected components are of relevance because they pose a terminal structure of an $\mathcal{M}$ that can not be left.

Definition 4.45. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and $\tilde{S} \subseteq S$. The view $\mathcal{M}_{bscc}$ is defined by its grouping function F_{bscc} where $\tilde{F}_{bscc} : \tilde{S} \to M$ with

$$s \mapsto \{T \in BSCC_{\mathcal{M},n} \mid s \in T\}$$

and $M = BSCC_{\mathcal{M},n} \cup \{\bullet\}$.

The view $\mathcal{M}_{bscc}$ groups all states together that are in the same BSCC. In a sense, it is a specialization of $\mathcal{M}_{scc}$, because instead of all SCCs the set is no restricted to the ones where there is no outgoing transition to another SCC. Hence, equality, equivalence classes and the state set behave and are constructed very similarly.

Example 4.19. In Figure 25 the simplified representation of $\mathcal{M}_{bscc}(0)$ (right) on $\mathcal{M}$ (left) can be observed. The SCCs in $\mathcal{M}$ are $\{s_1\}, \{s_2, s_3, s_4, s_5\}, \{s_6, s_7, s_8, s_9\}, \{s_{10}, s_{11}, s_{12}\}$ and $\{s_{13}\}$. The only ones where there is no state with an outgoing transition to a state in another SCC are $\{s_6, s_7, s_8, s_9\}$ and $\{s_{13}\}$. Hence, these two sets of states become a new state $\mathcal{M}_{bscc}$ while all remaining states are mapped to $\emptyset$ and thereby grouped together.

In the implementation strongly connected components are determined using the algorithm of Gabow, afterwards filtering those SCCs that have only transitions to states within the SCC. Only the bottom strongly connected components remain.

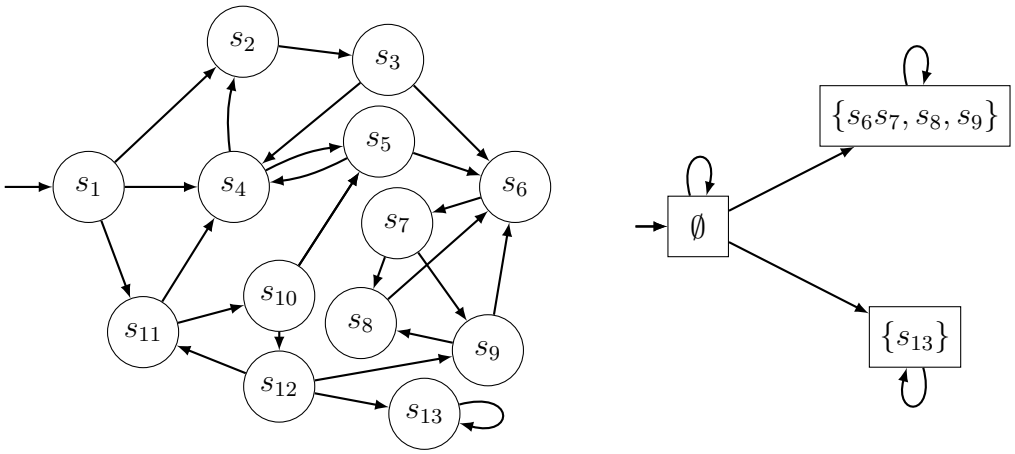


Figure 25: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{bscc}$ (0) on it (right)

5 View Implementation

The focus of this chapter is to delve into the implementation details of the *PMC-Vis* web application, particularly concentrating on the mechanisms behind views and their integration within the system. The application primarily serves the purpose of exploring and visualizing MDPs. Views are used for preprocessing data and give simplified perspectives on MDPs.

The PMC-Vis project is a web-based application designed for MDP exploration and visualization. Since it is a web-based application the project can be divided into two main components: the frontend and the backend. The frontend is responsible for rendering visualizations and providing various customization options for viewing MDPs. The backend supplies data to the frontend in JSON format. It consists of three parts: several prism models, the Java server, and multiple databases. The Java server interacts with the prism models, controls them, parses them, and creates a dedicated database for each model. These databases primarily store the structure of MDPs, divided into two tables: one for states and another for transitions.

Information such as property values are stored in dedicated columns in the respective state table. Whenever a new property is introduced, a new column is added to the database, assigning property values to each state. Similarly, values obtained from the grouping function of a particular view are stored in designated columns. Figure 26 shows an overview of the structure of the project.

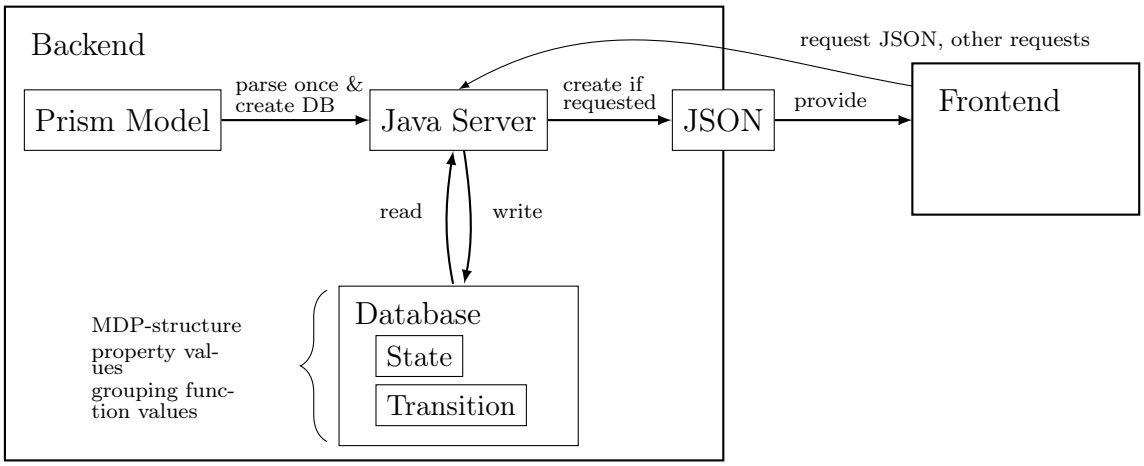


Figure 26: Project Structure, showing interaction and internal structure of frontend and backend

Views are implemented within the server on the backend side in the `prism.core.views` package. Each view is represented by a Java class. While individual views often have their own dedicated class, sometimes multiple similar views are implemented within a single class. All views inherit from the abstract class `View`. This abstract class contains common attributes and methods required by all views. Many of these are used for testing and I/O. In Listing 2 an overview of relevant attributes and methods is shown. The method is `buildView()` is of particular importance as it computes the mappings of the grouping function and updates the corresponding state table in the database. The process involves 1) checking prerequisites, 2) creating a new col-

umn for storing grouping results, 3) applying the grouping function, which returns the mapped values in the form of a list of SQL queries and 4) executing these SQL queries to insert the computed values (see Listing 3).

```
public abstract class View implements Namespace { // M_θ

    protected final ViewType type; // similar to identifier θ

    protected final Model model; // MDP M

    protected long id;

    protected Set<Long> relevantStates; // S̃, Def. view

    protected Map<String,Set<String>> stateRestriction
    = new HashMap<>(); // Z, Def. selective composition


    protected boolean semiGrouping = true; // • ∈ F_θ[S] allowed
    // true: remaining states without property grouped
    // false: remaining states without property NOT grouped

    protected enum BinaryMode {SHOW, HIDE} // Δ ∈ {⊤,⊥}
    // HIDE: Group states that have the property
    // SHOW: Group states that do NOT have the property

    protected BinaryMode binaryMode = BinaryMode.SHOW; // Δ = ⊤
    // declares binary mode, only queried in binary views

    private void setRelevantStates(Map<String, Set<String>>
       stateRestriction) { ... }

    public void buildView() { ... }

    protected abstract List<String> groupingFunction() throws
       Exception; // F̃_θ, Def. detached grouping function
```

Listing 2: Most relevant attributes and methods of class `View`

Creating a new view entails implementing the grouping function, the method `getCollumn()`, and any necessary private attributes. This approach matches the formal definition of views and ensures consistency in the process of generating views, given that each view is essentially defined by its grouping function.

Parallel composition of views is achieved by simply creating another view, resulting in the grouping function entries being written to a new column in the database. Selective composition involves setting a restriction (`stateRestriction`) that corresponds to Z from Definition 3.11. This restriction is realized with a map, mapping column names to lists of allowed values. In contrast to the formalization this restriction is then used to generate an SQL Query, that selects states from the database that meet this requirement. This is achieved by setting the where clause of the SQL statement to a boolean formulae in conjunctive normal form that expresses the requirement.

```
public void buildView() {
    try {
        // 1. View specific checks
        if (!viewRequirementsFulfilled()) return;

        // 2. Create new Column
        model.getDatabase().execute(String.format("ALTER TABLE %s
            ADD COLUMN %s TEXT DEFAULT %s",
        model.getStateTableName(), getCollumn(), Namespace.
            ENTRY_C_BLANK));

        // 3. Compute grouping function mapping
        List<String> toExecute = groupingFunction();

        // 4. Write mapping to database
        model.getDatabase().executeBatch(toExecute);

    } catch(Exception e){
        throw new RuntimeException(e);
    }
}
```

Listing 3: Implementation of `buildView()` function

The concept of disregarding views is implemented a little less general as in the formalization. In this context, the variable `semigrouping` plays a crucial role, indicating whether something should be grouped or not. For categorizing views the disregarded value is fixed empty set or string. That is, only for states that would otherwise be mapped to an empty set or string, it can be set whether they are to be grouped. For instance, in the case of the view $\mathcal{M}_{scc}$ with a parameter "n" equal to 3, the variable `semigrouping` determines whether states in an SCC with less than three states are grouped together. For binary views, an enumeration value decides whether $\top$ or $\bot$ should possibly be disregarded (not grouped), in the case of the variable `semigrouping` being true.

To create a view, one accesses `localhost:8080/<model_id>/view:<viewname>?param=<param_val_1>&param=<param_val_2>..."` via a browser, assuming the backend and frontend are operational. Afterward, accessing `localhost:3000/?id=<model_id>/` displays the graphical view representation. The accessing of `localhost:8080` causes the call of `createView()`, which creates a new view by calling its constructor, saves it in the internal list of views and finally calls `build()` on it, which causes the the grouping function values of the view being written to the database. When accessing `localhost:3000` the frontend calls the backend which then creates and provides the JSON file to the frontend with the information stored in the database. This includes the saved information about the grouping function mappings.

The implementation of views utilizes an internal graph structure that was particularly essential for views that employ grouping based on the structural properties of the MDP graph. To facilitate this, the application utilizes the jGraphtT library [22], which not only provides graph structures but also offers many common graph algo-

rithms. This library was selected because it is the most common, most up to date java library for graphs with the best documentation and broadest functionality. The MDP itself is represented by the class `MdpGraph` that inherits from the class `DirectedWeightedPseudgraph` from the jGraphtT library. A directed weighted pseudograph has been chosen because it is directed, allows weights, self loops and multiple edges between nodes, as they occur in MDPs, where the weights are set to the transition probabilities, edges are transitions and nodes are states.

To maintain a lightweight graph, nodes and vertices are represented as long values, referring to state and transition IDs. Information about states and transitions is accessible via hashmaps within the `MdpGraph` class. These hashmaps facilitate modular access to essential information for view functionalities, such as action strings for transitions or variable values for states.

Apart from implementation of views and by them required classes and data structures, several functionalities have been implemented to allow the following actions at runtime without restarting the server:

- Display current view information (Parameter values etc.) and built views
- Rebuild view with new parameters
- Remove views by providing id or name ($||^{-1}$, from Definition 3.10)

These rely on several string parsing methods that have been implemented. The latest version of the project PMC-Vis is available at `https://imld.com/pmc-vis`. The latest artifact of the project is available at `https://zenodo.org/record/8172531`.

6 Comparison and Evaluation

In this chapter we want to show how views can be utilized in three different usecases. In subsection 6.4 we will consider performance aspects. In the usecases we will see screenshots made in the to this time current version PMC-Vis. (Grouped) States are represented by grey rectangles. The blue labels on them are the mapped values from the grouping function. These do not always exactly match the ones of the formalization. Especially it is to note that `__BLANK__` $= \bullet$. If parallel composition is used in some form, the values of the grouping functions in the label are separated with `|` and are in the same order as they have been composed with the operator $||$. If we formally have $\mathcal{M}_{\theta_1} || \mathcal{M}_{\theta_2} = \mathcal{M}_{\theta_1 || \theta_2}$ and for some grouped states $\tilde{S}$ it holds for all of them $\mathcal{M}_{\theta_1 || \theta_2}(\tilde{s}) = (a, b)$ where $\tilde{s} \in \tilde{S}$, then in the screenshot of PMC-Vis the grouped state of the view receives the label `a|b`.

6.1 Explore Modules

One Purpose of views is to simplify MDPs to make them better understandable. Due to the state explosion problem already rather simple systems can become very hard to oversee. As an example consider the concurrency problem Dining Philosophers, where n philosophers have to share n-1 forks and each of them needs two to eat. They are sitting on a round table with forks between them. Each of them only has access to fork to their left and right, if it is not already occupied. When representing the problem with an MDP the choice of which fork the philosopher tries to pick is made at random with an uniform distribution. The respective prism File is shown in the appendix (A.2).

Already for only three philosophers the MDP has 956 states. When looking at the graphical representation from PMC-Vis in Figure 27 it appears to be little helpful for understanding and exploring the graph due to its sheer size. Although the graph is large, this MDP is still rather small. With already five philosophers the MDP has about 100 000 states and with 10 philosophers the resulting MDP has more than 8 billion states. The view $\mathcal{M}_{Var:a}$ can help to only show the behavior of some Philosophers and hiding the behavior of the remaining ones.

The view $\mathcal{M}_{Var:a}(\texttt{p1})$ groups all states that have the same value of (`p1`) ignoring the values of the remaining variables. That is, the values of `p2` and `p3` are hidden, which results in only showing the module of philosopher `p1` (Figure 28). This may help immensely if only a specific module is of interest or the reaming modules have the same structure, as it is the case here with Dining Philosophers.

It is also possible to use the view $\mathcal{M}_{Var:a}$ to see the interleaved behavior of two or more modules. To see the interleaved behavior of `p1` and `p2`, we use parallel composition $\mathcal{M}_{Var:a}(\texttt{p1}) || \mathcal{M}_{Var:a}(\texttt{p2})$. This results in states s_1, s_2 being grouped if $(F_{Var:a}(s_1 \,|\, \texttt{p1}), F_{Var:a}(s_1 \,|\, \texttt{p2})) = (F_{Var:a}(s_2 \,|\, \texttt{p1}), F_{Var:a}(s_2 \,|\, \texttt{p2}))$. Hence, only the value of `p3` is hidden which results exactly in the desired interleaved model (Figure 29).

In general the views ${\mathcal{M}_{VarDNF}}^{\top}$ and ${\mathcal{M}_{VarCNF}}^{\top}$ are very powerful since they allow to perform every possible operation on variables, because the state space is finite, for MDPs in practice.

Figure 27: Screenshot in PMC-Vis of approximately half the MDP $\mathcal{M}$

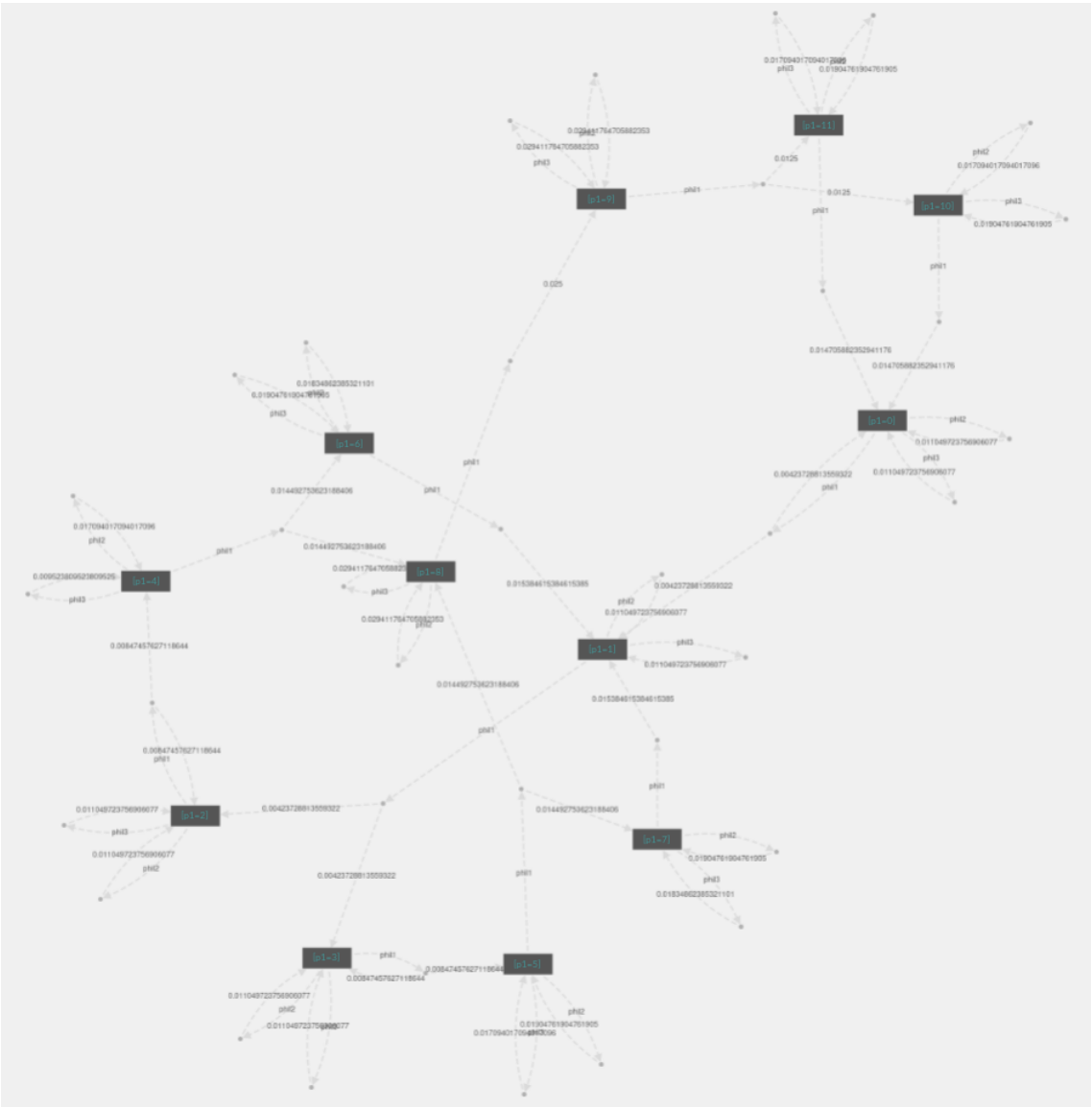


Figure 28: Screenshot in PMC-Vis of view $\mathcal{M}_{Var:a}(\texttt{p1})$

Figure 29: Screenshot in PMC-Vis of view $\mathcal{M}_{Var:a}(\texttt{p1}) \,||\, \mathcal{M}_{Var:a}(\texttt{p2})$

6.2 Investigate why illegal states can be reached

In this section we want to show how views can be used for debugging. In this specific case we assume that we observed unwanted behavior or model checking results. We know that some states - that is some assignment of variables - are not allowed. We will check if these states are in fact not reachable.

We will consider a small MDP, that we have been given the PRISM file for. It represents two systems that intend to send information via a unshareable medium. With a probability of 0.8 a system can establish a connection and with probability of 0.2 establishing a connection will fail. After an established connection access to the medium shall only be granted, if it is not occupied by the other system. If the medium is not occupied the system starts sending, otherwise it waits until the medium is free. The termination of the transmission is modeled with probabilities. There is 50 percent chance of terminating the transmission and a 50 percent chances of continuing. The number 0 represents that a system is trying to establish a connection, the number 1 represents that a system established the connection, the number 2 represents that the system is sending.

The state $\langle 2, 2\rangle$ should not be reachable, since it represents the situation of the two systems occupying the unshareable medium at the same time. We will check if this state in fact is not reachable by using the $\mathcal{M}_{VarCNF}(c(s))$ where $c(s) := (x = 2) \wedge (y = 2)$ (Figure 30). Note that we are not using view *not* in disregarding form.

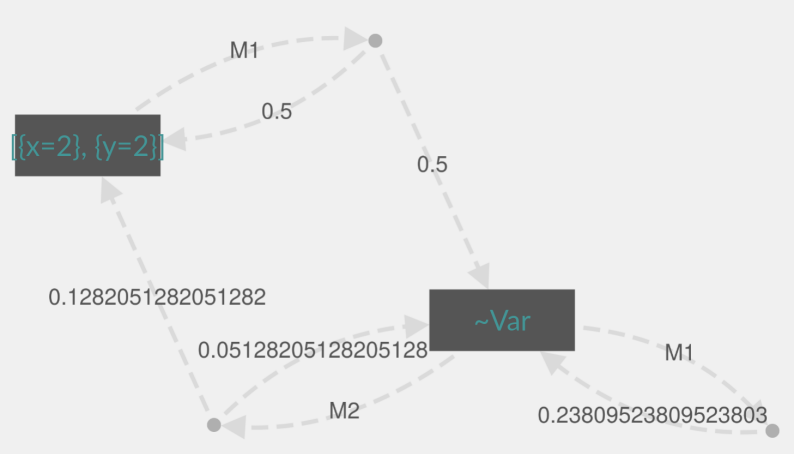


Figure 30: Screenshot in PMC-Vis of view $\mathcal{M}_{VarCNF}(c(s))$ where `~Var` refers to $\bot$.

We observe, that this state is reachable, because it is shown and only reachable states are shown at all in PMC-Vis. The question is how and why. To get this information, it should be investigated by which state this critical state $\langle 2, 2\rangle$ has been reached. One way of accomplishing that is to look into the database that stores states and transitions (Table 2). For this very simple MDP, this already seems to be of little help.

A better approach might be to look at the PRISM model file (figure 4). People experienced in working with prism models may be able to find the problem quickly, especially since this is a rather small MDP. With less experienced people, and especially with larger and more complicated models, finding a problem becomes much more difficult. So let us see how views can help us.

transition_id	state_out	action	probabilityDistribution	property_0	scheduler_0	property_1	scheduler_1
1	3	M1	6:1.0	0.0	1.0	0.0	1.0
2	3	M2	3:0.8;4:0.2	0.0	1.0	0.0	1.0
3	7	M1	1:0.5;7:0.5	0.0	1.0	0.0	1.0
4	7	M2	6:0.5;8:0.5	0.0	1.0	0.0	1.0
5	0	M1	0:0.8;3:0.2	0.0	1.0	0.0	1.0
6	0	M2	0:0.8;1:0.2	0.0	1.0	0.0	1.0
7	4	M1	7:1.0	0.0	1.0	0.0	0.0
8	4	M2	5:1.0	1.0	0.0	1.0	1.0
9	4	M2	3:0.5;5:0.5	0.5	0.0	0.5	0.0
10	8	M1	2:0.5;8:0.5	0.0	1.0	0.0	1.0
11	1	M1	1:0.8;4:0.2	0.0	1.0	0.0	1.0
12	1	M2	2:1.0	0.0	1.0	0.0	1.0
13	1	M2	0:0.5;2:0.5	0.0	1.0	0.0	1.0
14	2	M1	2:0.8;5:0.2	0.2	1.0	0.2	1.0
15	6	M1	0:0.5;6:0.5	0.0	1.0	0.0	1.0
16	6	M2	6:0.8;7:0.2	0.0	1.0	0.0	1.0

Table 2: Transitions table form the database, where states are numerated with `state_out` $= x \cdot 3^1 + y \cdot 3^0$ for a state $\langle x, y \rangle$

```
mdp

module M1
x : [0..2] init 0;

[] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);
[] x=1 & y!=2 -> (x'=2);
[] x=2 -> 0.5:(x'=2) + 0.5:(x'=0);
endmodule

module M2
y : [0..2] init 0;

[] y=0 -> 0.8:(y'=0) + 0.2:(y'=1);
[] y=1 & x!=2 -> (y'=2);
[] y=1 -> 0.5:(y'=2) + 0.5:(y'=0);
endmodule
```

Listing 4: PRISM model file, where modules define the two systems mentioned in the text

First, we will use the view $\mathcal{M}_{\overleftarrow{dist}}$ from this state (Figure 31 (left)). This shows the structure of the MDP in terms of how this state is reached. That is, states are grouped that require the same number of transitions to reach the critical state $\langle 2, 2 \rangle$. The natural number indicates the number of transitions needed to reach it. The state `~Var|1` represents a group of states that can reach $\langle 2, 2 \rangle$ with a single

transition. We are interested in which states it contains, so that we know which state of the original MDP reaches $\langle 2, 2 \rangle$. Since the current version of the project does not yet implement expanding grouped states to their contained states on a visual level, we will use a custom view that emulates this feature.

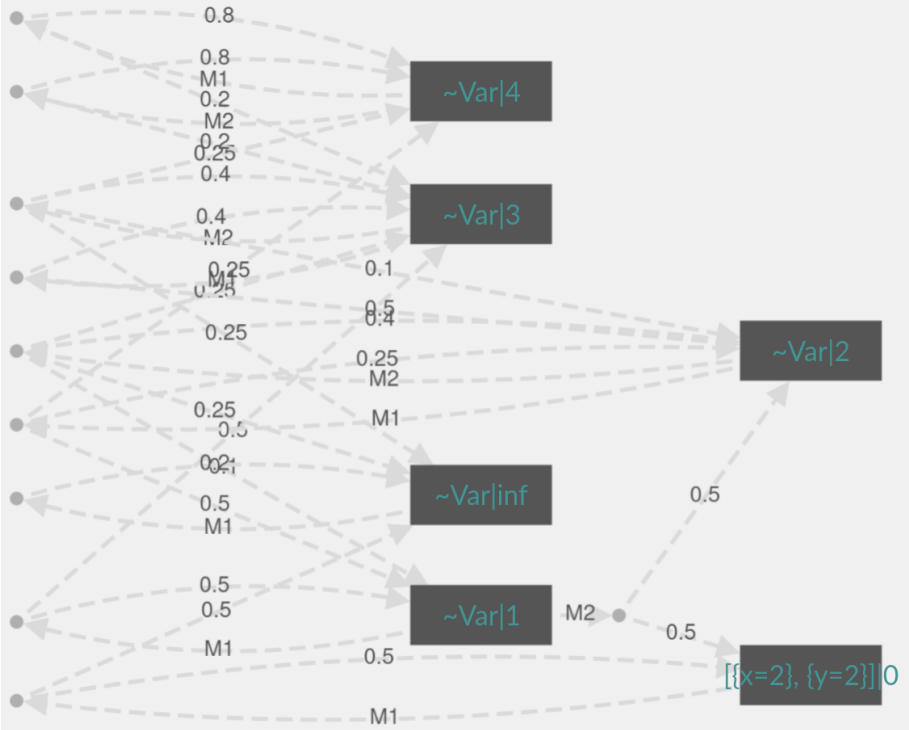


Figure 31: Screenshot in PMC-Vis of view $\mathcal{M}_{VarCNF}(c(s)) \,||\, \mathcal{M}_{\overrightarrow{dist}}$ where `~Var` refers to $\perp$.

Definition 6.1. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$ and $n \in \mathbb{N}$. The view $\mathcal{M}_{id}$ is defined by its grouping function F_{id} where $\tilde{F}_{id} : \tilde{S} \to M$ with

$$\tilde{F}_{id}(s) = s$$

and $M = S \cup \{\bullet\}$.

We will use partial application with $\mathcal{M}_{VarCNF}(c(s)) \,||_Z\, \mathcal{M}_{id}$ where $Z = \{(F_{\overrightarrow{dist}}, 1)\}$ to expand this state. The MDP graph then looks like in 32. It can be seen that the state contains only one state with the id seven. In the current version of the implementation it is not possible to get the parameter values. With the database file we obtain that this is the state where $x = 2$ and $y = 1$. Since $x = 2$ means that system 1 is sending and system 2 is waiting, we see that it is possible for the second system to start sending on the medium although it is already occupied by the first system!

If we look at the prism file (Listing 4), we can see why this is the case. In the last line of the second module, when $y = 1$, there is a 50% chance that $y = 2$ will be entered. This line was originally intended to terminate the transfer. From $y = 1$ it should not be possible to enter $y = 2$ if $x = 2$. After fixing this, the state no longer appears when using $\mathcal{M}_{VarDNF}{}^{\top}(c(s))$.

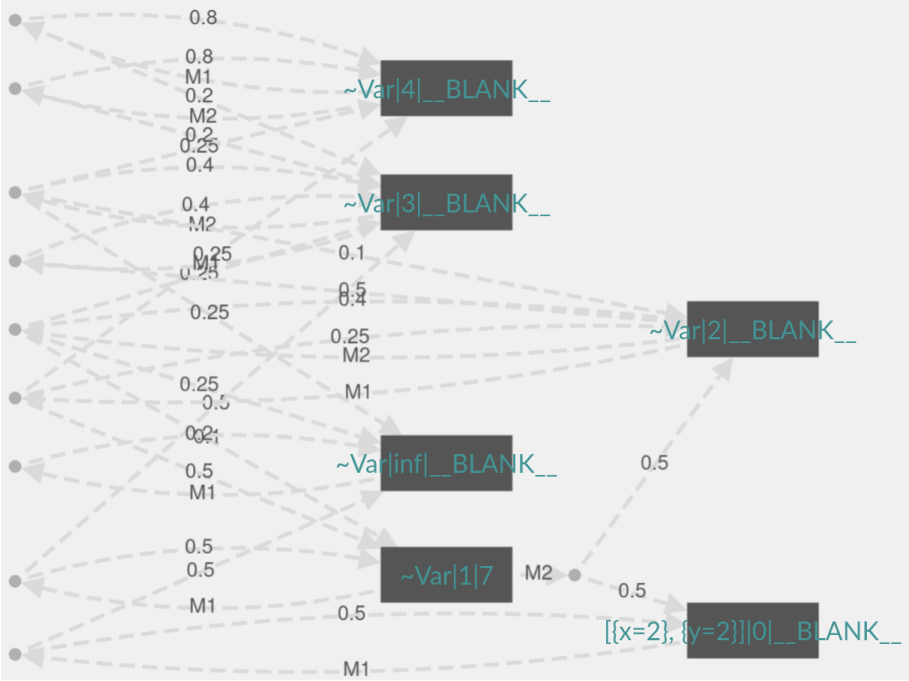


Figure 32: Screenshot in PMC-Vis of view $\mathcal{M}_{VarCNF}(c(s)) \,||\, \mathcal{M}_{\overrightarrow{dist}} \,||_Z \, \mathcal{M}_{id}$ where $Z = \{(F_{\overrightarrow{dist}}, 1)\}$ and `~Var` refers to $\bot$.

6.3 Understand and Debug MDP

In this subsection, we want to take a look at a more complex use case of how views can help us understand and fix a given MDP. We refer to the MDP with as described by the PRISM file in Listing 5.

First, we will gain some understanding of the model. We already saw in Section 6.1 that variables can help a lot in understanding a MDP. Using $\mathcal{M}_{Var:a}(\texttt{time})$ we see that the MDP has a limited time behavior (Figure 33 left). If we apply $\mathcal{M}_{Var:a}(\texttt{phases})$ we see that the system works in phases (Figure 33 right). If we look at $\mathcal{M}_{Var:a}(\texttt{time}) \,||\, \mathcal{M}_{Var:a}(\texttt{phases})$, we see that the phases are repeated for each iteration of time (Figure 34 right). However, we still have no information about the behavior of the system in these phases. Since this model has actions, we will consider the view $\mathcal{M}_{\overrightarrow{Act(s)_{\approx}}}$ (Figure 35). We see that we have sets of states that only have transitions with the action `[reconfigure]` outgoing, the action `[working]` outgoing, the actions `[configure1]`, `[configure2]`, `[end_reconfigure]` outgoing, the action `[working]` outgoing, or the action `[end]` outgoing. By interpreting the names of the actions, we see that this system seems to have a configuration phase, a working phase, a reconfigure phase, and an end phase. The grouping seems very similar to $\mathcal{M}_{Var:a}$ (`phases`). Therefore, we consider $\mathcal{M}_{\overrightarrow{Act(s)_{\approx}}} \,||\, \mathcal{M}_{Var:a}(\texttt{phases})$ (Figure 36). In fact, the enumeration coincides with the grouping of outgoing actions, except for `[end]` and `[reconfigure]`.

So we know that the system works for `phase` $= 0$, configures for `phase` $= 1$, and stops configuring for `phase` $= 2$. This process is repeated six times until `time` $= 5$.

In general, we learned that this system runs several times, each time choosing certain configurations before it runs. Its reward function rewards states that

```
mdp

const int c_max_time = 5;

label "end" = (time = c_max_time);

module reconfiguration
activity: [0..1] init 0;
config1 : [0..4] init 0;
config2 : [0..4] init 0;

[reconfigure] (activity=0) -> (activity'=1);

[configure1 ] (activity=1) -> (config1 '=0);
[configure1 ] (activity=1) -> (config1 '=1);
[configure1 ] (activity=1) -> (config1 '=2);
[configure1 ] (activity=1) -> (config1 '=3);
[configure1 ] (activity=1) -> (config1 '=4);

[configure2 ] (activity=1) -> (config2 '=0);
[configure2 ] (activity=1) -> (config2 '=1);
[configure2 ] (activity=1) -> (config2 '=2);
[configure2 ] (activity=1) -> (config2 '=3);
[configure2 ] (activity=1) -> (config2 '=4);

[end_reconfigure] (activity=1) -> (activity'=0);
endmodule

module phases
phases: [0..2] init 0;

[reconfigure] (phases=0) -> (phases'=1);
[end_reconfigure] (phases=1) -> (phases'=2);
[working] (phases=2) -> (phases'=0);

endmodule

module timer
time: [0..c_max_time] init 0;

[working] (time < c_max_time) -> (time'=time+1);
[reconfigure] (time < c_max_time) -> true ;

[end] (time=c_max_time) -> true ;
endmodule

rewards "Utility"
[working] true : config1 *3 + config2 *2;
endrewards
```

Listing 5: PRISM model file

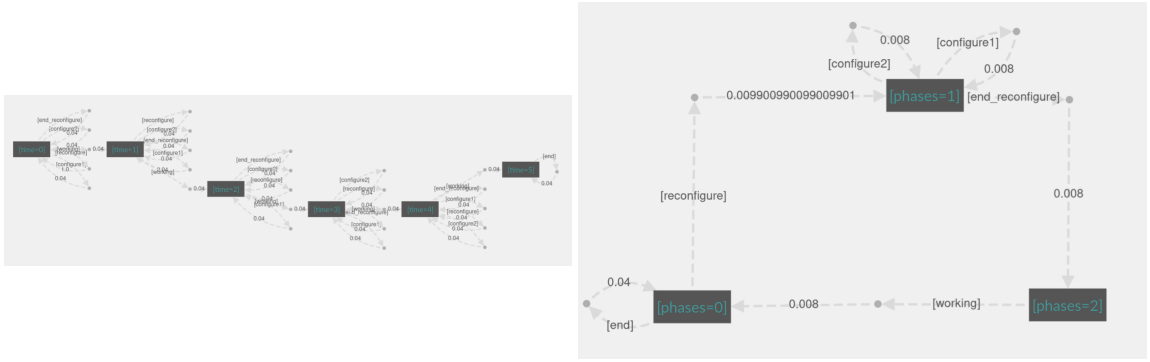


Figure 33: Screenshot in PMC-Vis of view $\mathcal{M}_{Var:a}(\texttt{time})$ (left) and view $\mathcal{M}_{Var:a}(\texttt{phases})$ (right). Larger versions of these images in appendix (**??**)

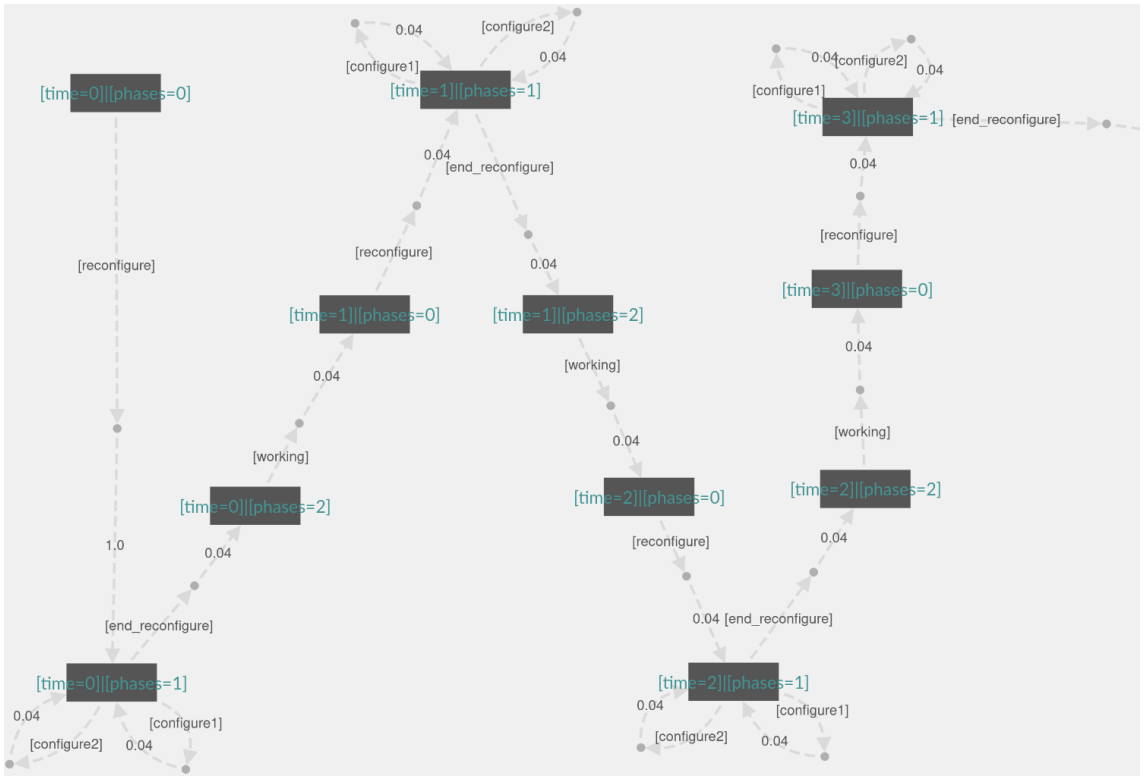


Figure 34: Screenshot in PMC-Vis of view $\mathcal{M}_{Var:a}(\texttt{time}) \,||\, \mathcal{M}_{Var:a}(\texttt{phases})$

work with a better configuration. A classic model checking value is to determine $Exp^{\min}_{utility}(\Diamond \texttt{end})$ and $Exp^{\max}_{utility}(\Diamond \texttt{end})$. For $Exp^{\min}_{utility}(\Diamond \texttt{end})$ we obtain 0, for $Exp^{\max}_{utility}(\Diamond \texttt{end})$ we obtain ∞. Since the system is modeled for a finite time and chooses from a finite set of configurations each time, it is undesirable behavior for $Exp^{\max}_{utility}(\Diamond \texttt{end})$ to be infinite. We will now show how views can help to find the cause.

Such behavior of infinite $Exp^{\max}_{utility}(\Diamond \texttt{end})$ is often caused by cycles. One feasible idea would be to use a view with cycles. A less resource-intensive way to find sets containing cycles is to find strongly connected components, since each cycle is a strongly connected component. The view $\mathcal{M}_{scc}$ shows that there are quite

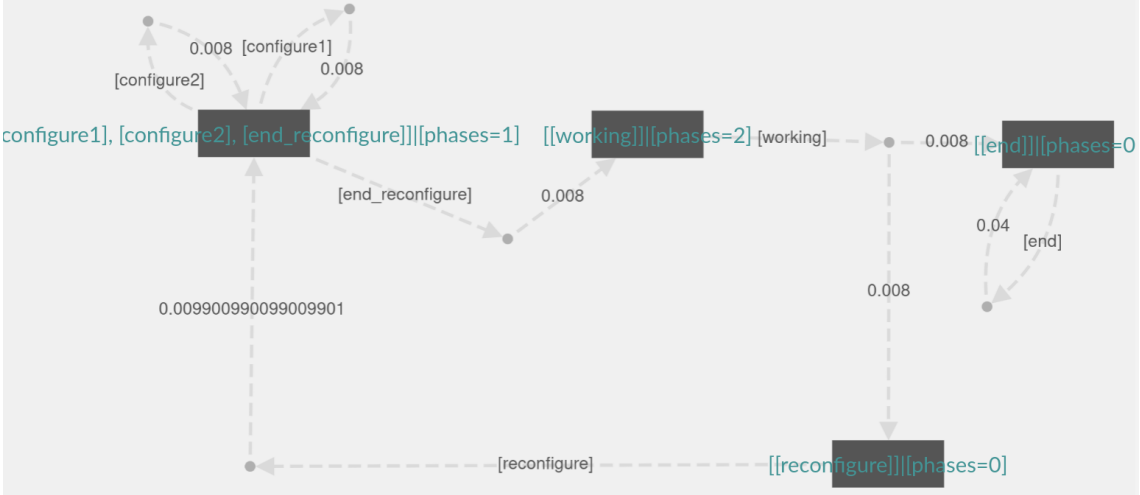


Figure 35: Screenshot in PMC-Vis of view $\mathcal{M}_{\overrightarrow{Act(s)\approx}}$

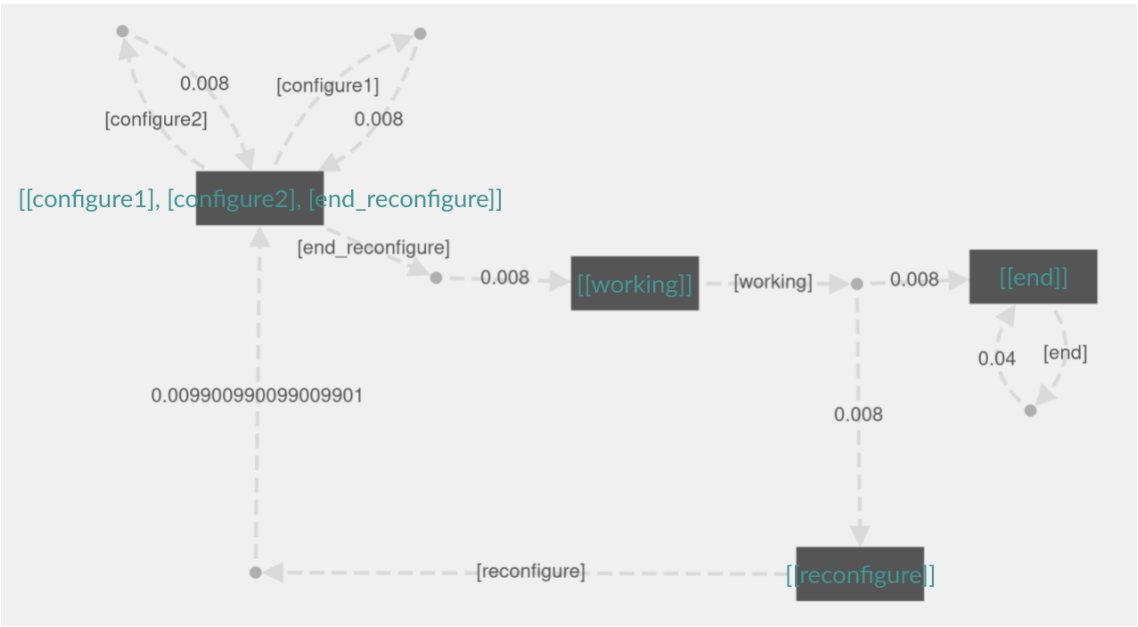


Figure 36: Screenshot in PMC-Vis of view $\mathcal{M}_{Var:a}(\texttt{time}) \,||\, \mathcal{M}_{Var:a}(\texttt{phases})$

large strongly connected components (Figure 37). To find out where these are, we look at the parallel composed view on MDP with $\mathcal{M}_{scc} \,||\, \mathcal{M}_{\overrightarrow{Act(s)\approx}}$ (Figure 38). We see that the strongly connected components are in the configuration phase. Looking at the Prism file, we see that we can change the configuration as often as we like. Thus, there is no guarantee that a configuration can only be selected once. This leads to infinite paths in the MDP, which in turn leads to infinite maximum expectations. This can be fixed by ensuring that a configuration can only be selected once. A simple way to do this is to sequentially select the two configurations: First `[configuration1]` is selected, then `[configuration2]`, and finally the configuration phase is ended (`[end_configuration]` is the only available action left) (Figure 6).

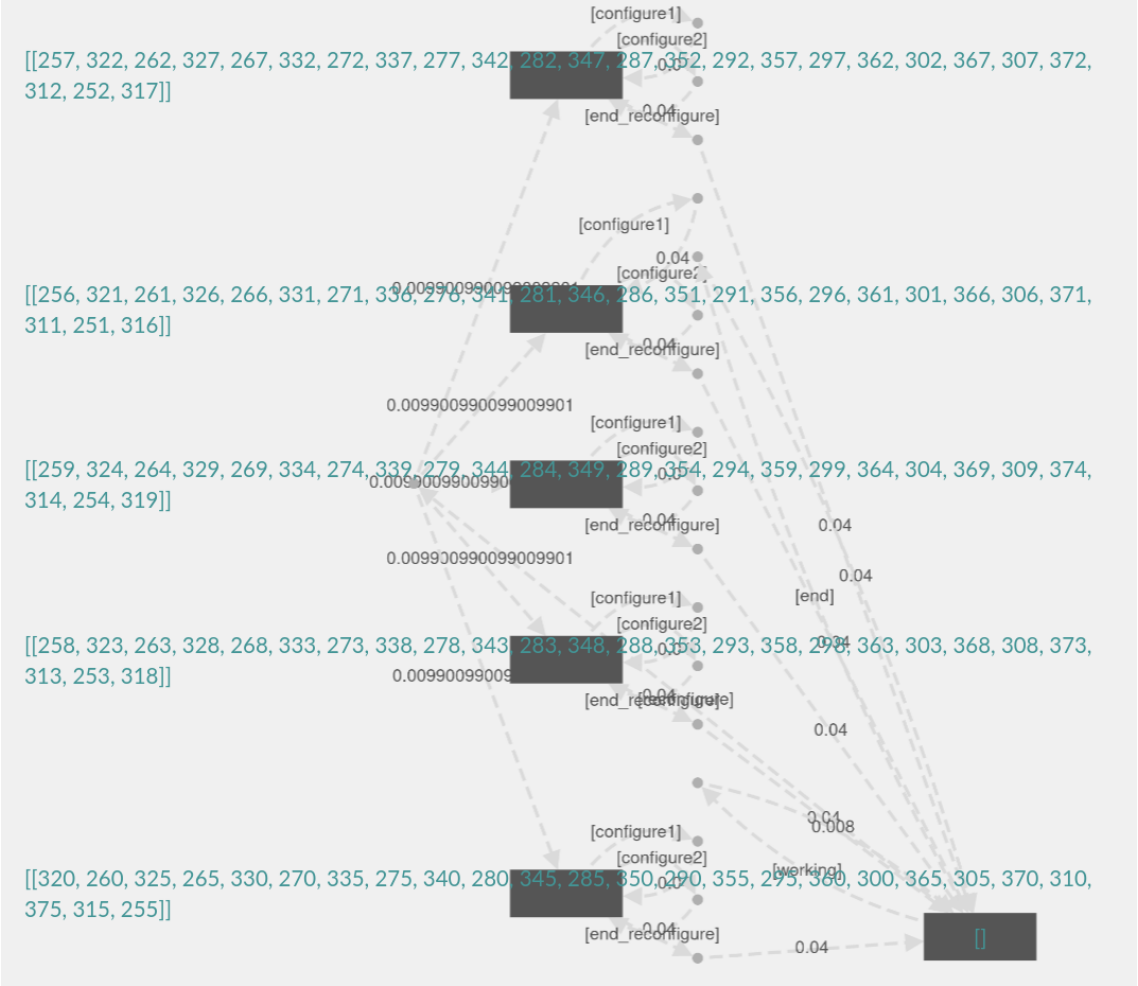


Figure 37: Screenshot in PMC-Vis of view $\mathcal{M}_{scc}$

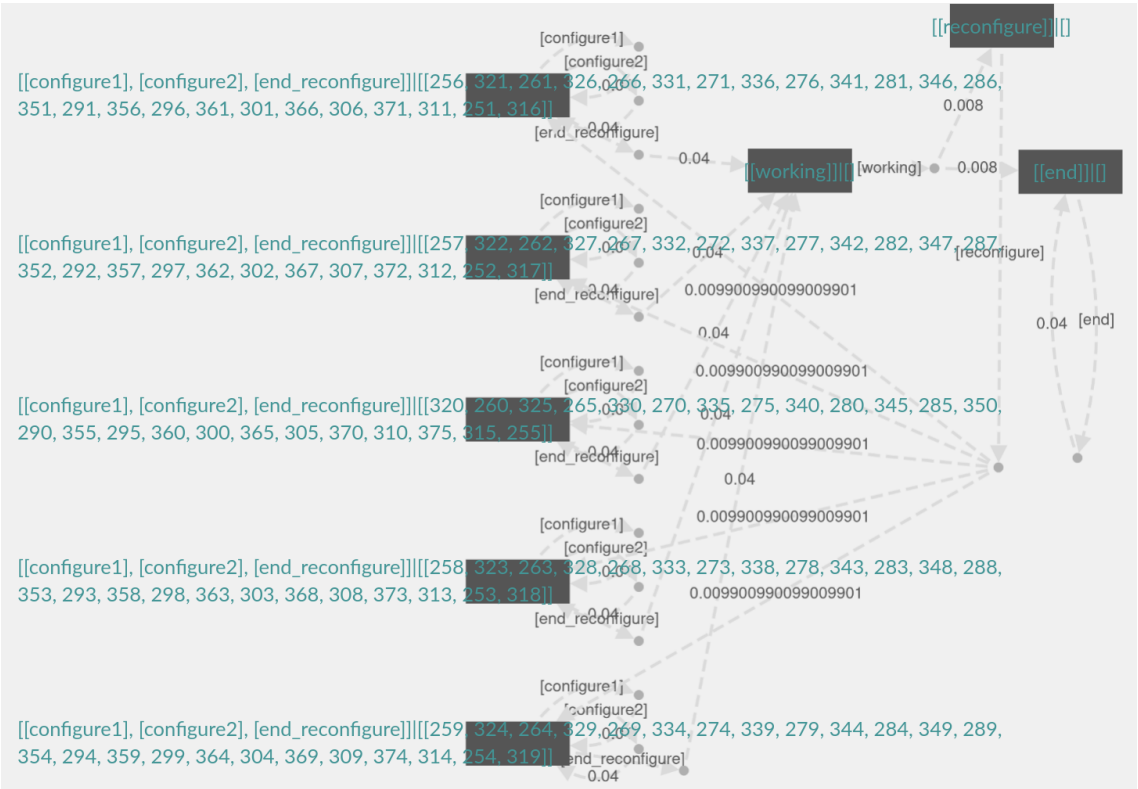


Figure 38: Screenshot in PMC-Vis of view $\mathcal{M}_{scc} || \mathcal{M}_{\overrightarrow{Act(s)}_{\approx}}$

```
mdp

const int c_max_time = 5;

label "end" = (time = c_max_time);

module reconfiguration
activity: [0..3] init 0;
config1 : [0..4] init 0;
config2 : [0..4] init 0;

[reconfigure] (activity=0) -> (activity'=1);

[configure1 ] (activity=1) -> (config1 '=0) & (activity'=2);
[configure1 ] (activity=1) -> (config1 '=1) & (activity'=2);
[configure1 ] (activity=1) -> (config1 '=2) & (activity'=2);
[configure1 ] (activity=1) -> (config1 '=3) & (activity'=2);
[configure1 ] (activity=1) -> (config1 '=4) & (activity'=2);

[configure2 ] (activity=2) -> (config2 '=0) & (activity'=3);
[configure2 ] (activity=2) -> (config2 '=1) & (activity'=3);
[configure2 ] (activity=2) -> (config2 '=2) & (activity'=3);
[configure2 ] (activity=2) -> (config2 '=3) & (activity'=3);
[configure2 ] (activity=2) -> (config2 '=4) & (activity'=3);

[end_reconfigure] (activity=3) -> (activity'=0);
endmodule

module phases
phases: [0..2] init 0;

[reconfigure] (phases=0) -> (phases'=1);
[end_reconfigure] (phases=1) -> (phases'=2);
[working] (phases=2) -> (phases'=0);

endmodule

module timer
time: [0..c_max_time] init 0;

[working] (time < c_max_time) -> (time'=time+1);
[reconfigure] (time < c_max_time) -> true ;

[end] (time=c_max_time) -> true ;
endmodule

rewards "Utility"
[working] true : config1 *3 + config2 *2;
endrewards
```

Listing 6: Fixed model file

6.4 Performance

Views will be used on MDPs, that may have millions of states. For this reason, in this section we will consider the time to create a new view. In addition, we will take a look at the build time of the internal graph structure (`mdpGraph`), which is based on the jGraphtT library, in terms of its build time and memory usage. The tests were performed on a Dell XPS 9370 (16 GB RAM, Intel Core i7-8550U) with Manjaro KDE. Only necessary tools were open during the test: Firefox and IntelliJ.

Time was measured with self-written classes `Timer` and `TimeSaver` using the package `java.time.LocalTime`. The class implementation and an example use case are provided in the appendix (A.4). Memory usage was measured using the Java Object Layout (JOL) tool provided by openJDK. The method used was `totalSize()` in `org.openjdk.jol.info.GraphLayout`. The benchmark uses a single scalable MDP (prism model file in the appendix). The sizes considered are 50, 100, 500, 1 000, 5 000, 10 000, 50 000, 100 000, 500 000, and 1 000 000. Using a single scalable MDP means that execution times and memory usage may vary with different graph structures of the MDP. This is especially true for views based on the graph structure. Therefore, in this section we will give an overview of the expected values for build times and memory usage rather than a detailed analysis.

First, we will look at the time required to create views. All views are created 100 times for each considered size of the MDP. Their execution times for each size are then averaged. As we know from the 5 chapter, creating a view involves instantiating the object and calling the build function. The instantiation only involves setting a handful of private attributes. Since the number of attributes of a view is fixed and their size is not related to the size of the MDP, the instantiation time is negligible. Building `mdpGraph` is also part of instantiating a view if it has not yet been built on the model, but will be considered separately. Therefore, we will focus on the time taken by the `buildView()` method. As we know from the section 5, the build `buildView()` method is divided into four parts: 1) prechecks, 2) creating a new column, 3) executing grouping function 4) writing the results to the database. First, we will consider the computation time of the grouping function, since the time for the actions performed in steps 1), 2) and 4) are almost identical or identical, with respect to the different sizes of MDP.

In Figure 39 we see the averaged execution times (100 executions) of `groupingFunction` for the different views. It can be seen that the execution times are very similar overall and behave linearly with the number of states of the MDP. For MDPs with up to 1 million states for no view, the execution time of its grouping function exceeds 1.5 seconds. Variations in the microsecond range are to be expected for very small MDPs. In general, the performance results as shown indicate very good scalability in terms of views being usable on large models. This very good performance is largely due to the graph structure provided by jGraphtT.

In Figure 40 we see the execution time of the grouping function compared to executing the generated SQL statements (writing results/mappings to database) and the creation of the `mdpGraph`. We see that the execution time of the grouping function is the least time consuming operation, except for the prechecks. The most time is taken by writing the results to the database, where even building the `mdpGraph` is faster. The times for writing to the database refer to the current database imple-

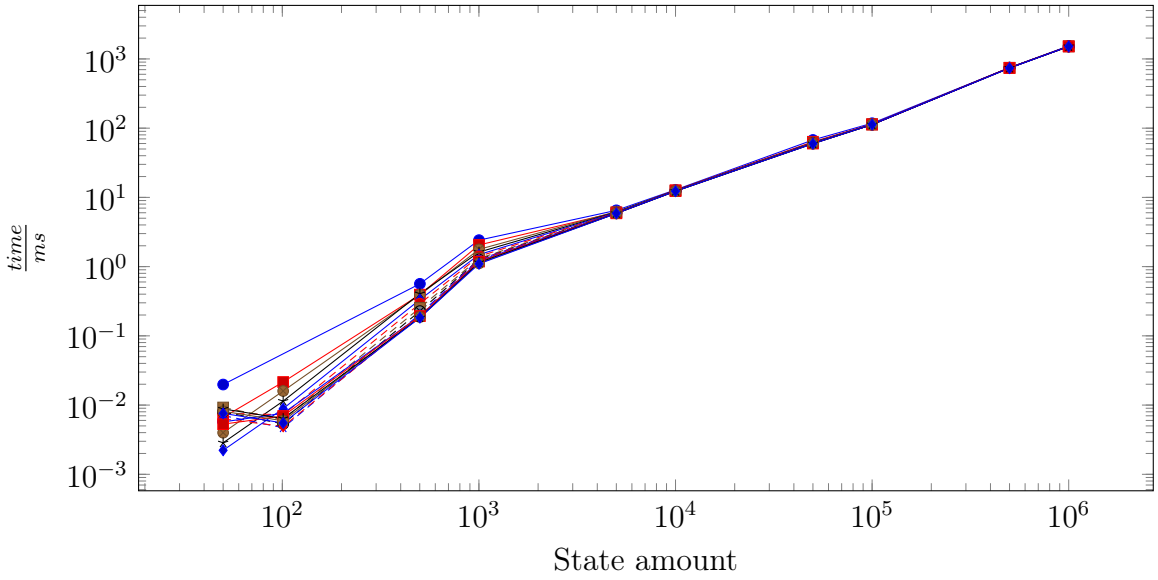


Figure 39: Average grouping function computation times

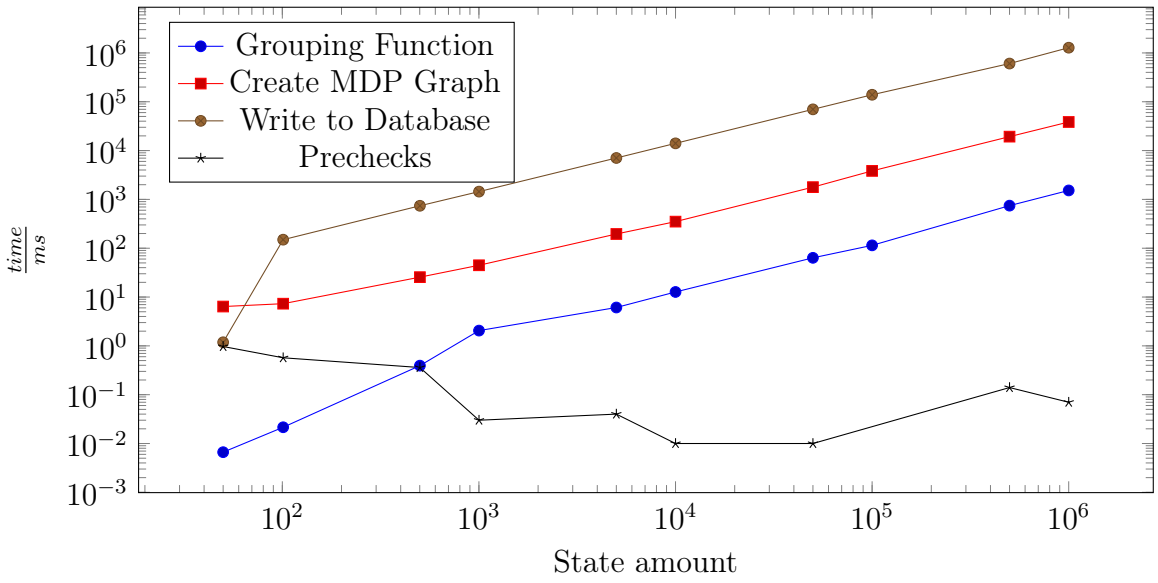


Figure 40: Average times for grouping function computation (1 representative from Figure 39), writing to the database and building the `mdpGraph`

mentation of PMC-Vis, which may change in the future.

As stated before, the performance of computing grouping functions depends heavily on the implemented graph structure (`mdpGraph`). This graph structure is held in memory and becomes quite large for MDPs with about a million states. Figure 41 shows the measured depth size (including referenced objects) of the `mdpGraph`. As expected, the size of the `mdpGraph` is linear to the size of the MDP. Additionally, it can be seen that up to 1.3 GB of memory is used for the graph alone when the MDP reaches about 1 million states. Depending on the operating system, the amount of memory built into the machine (PC) and the currently available memory, it is possible to use the `mdpGraph` even for large MDPs. However, it should be noted

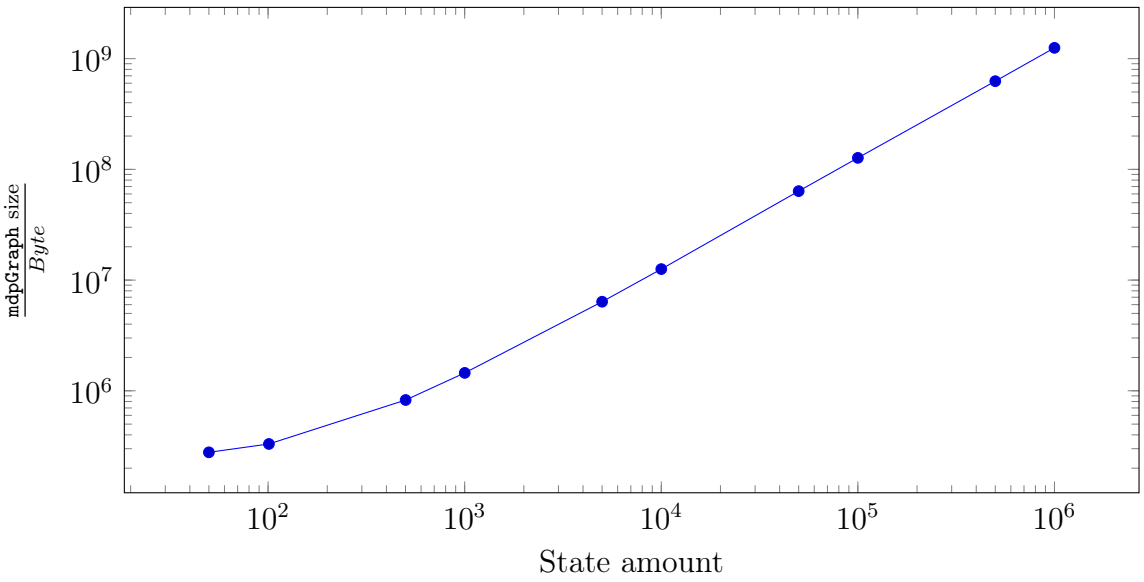


Figure 41: Size of `mdpGraph` for different model sizes

that this is only the size of the built graph object. When reading from the database to build the `mdpGraph`, more memory is needed because the objects containing the information from the database also remain temporarily in memory. PMC-Vis itself will also use memory, as will other possibly unrelated processes running on the system. On the machine used for testing, building the `mdpGraph` for MDPs with about 1 million states and creating views on them was still reasonable, but also close to the memory limit of the system. The `mdpGraph` could not be created with the test machine when the MDP used for testing was configured to five million states.

In general, we conclude that the computation of grouping functions is quite fast and scalable. The construction times of the `mdpGraph` are reasonable, but entail a high memory consumption. Database accesses are the most time consuming portion of the view creation time.

7 Outlook

In this bachelor thesis we proposed a concept to preprocess data for visualization, which we called view. We presented, defined and discussed types and operations on views and proposed example views. For some of them, use cases were presented to show their applicability in practice. Apart from the formalization, all features of views of views and all proposed views have been implemented in the context of the PMC-Vis project. The implementation is based on a fast but memory hungry graph structure provided by the jGraphtT library.

In conclusion, the formalization concisely describes the notion of a concept view, which helps to understand the implementation and may be useful for the development of other simplification approaches in model checking related systems. For the proposed example views, it should be noted that the author had little experience in working with MDPs. Therefore, most of the views are not motivated by how a view might be useful when working with MDP, but rather what simplifications are possible on MDPs, taking into account its components and structure. The use cases shown are only a small selection of what might be possible. The real practicability will be proven in practice. Also, MDP-experienced users who actually work with views in practice will most likely raise other ideas for views that might be helpful.

In addition to practical advancements for views and their use cases, there is still room for concept exploration from a theoretical perspective. For one, considering views with probabilities would have been beyond the scope of a bachelor thesis, so the probabilities present in MDPs were not explored. Furthermore, there may be advanced concepts from other disciplines that can be used to gain simplifications on MDPs.

The provided implementation serves as a solid framework to implement further views. As mentioned before, it is quite memory intensive. The MDP used for testing with 5 million states could not be built on the test machine due to lack of memory. Thus, for a given amount of available memory, the data structure imposes a hard limit on the applicability of views in practice if the MDP exceeds a certain size. This problem can be solved by not using the data structure (`mdpGraph`), but by accessing the database directly. This allows views to be used on almost arbitrarily large MDPs. While this is a great advantage, its downside is an immense performance loss (increase in build times) compared to the used `mdpGraph`, because database accesses with the current version of PMC-Vis are orders of magnitude slower than accessing the information in the `mdpGraph`. This is due to the fact that the `mdpGraph` consists of very efficient data structures such as hash maps and hash sets. In addition, the `mdpGraph` and its data structures are stored in memory, while accessing the database means accessing the machine's memory, which is in principle much slower than accessing memory. This leads to immense execution times when creating a view, especially if stateful database accesses are to be performed.

References

[1] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[2] Robert Görke, Tanja Hartmann, and Dorothea Wagner. Dynamic graph clustering using minimum-cut trees. In *Algorithms and Data Structures: 11th International Symposium, WADS 2009, Banff, Canada, August 21-23, 2009. Proceedings 11*, pages 339–350. Springer, 2009.

[3] Andreas Kerren, Helen Purchase, and Matthew O Ward. *Multivariate Network Visualization: Dagstuhl Seminar# 13201, Dagstuhl Castle, Germany, May 12-17, 2013, Revised Discussions*, volume 8380. Springer, 2014.

[4] C. Nobre, M. Meyer, M. Streit, and A. Lex. The state of the art in visualizing multivariate networks. *Computer Graphics Forum*, 38(3):807–832, jun 2019.

[5] Jimmy Johansson and Camilla Forsell. Evaluation of parallel coordinates: Overview, categorization and guidelines for future research. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):579–588, 2016.

[6] Salvador García, Sergio Ramírez-Gallego, Julián Luengo, José M. Benítez, and Francisco Herrera. Big data preprocessing: methods and prospects. *Big Data Analytics*, 1:1, 2016.

[7] Suad A Alasadi and Wesam S Bhaya. Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, 12(16):4102–4107, 2017.

[8] SS Baskar, L Arockiam, and S Charles. A systematic approach on data preprocessing in data mining. *Compusoft*, 2(11):335, 2013.

[9] Fang Zhou, Sebastien Malher, and Hannu Toivonen. Network simplification with minimal loss of connectivity. In *2010 IEEE international conference on data mining*, pages 659–668. IEEE, 2010.

[10] Henry Soldano and Guillaume Santini. Graph abstraction for closed pattern mining in attributed networks. In *ECAI 2014*, pages 849–854. IOS Press, 2014.

[11] Alex Arenas, Jordi Duch, Alberto Fernández, and Sergio Gómez. Size reduction of complex networks preserving modularity. *New Journal of Physics*, 9(6):176, 2007.

[12] Wai Shing Fung, Ramesh Hariharan, Nicholas JA Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 71–80, 2011.

[13] Ning Ruan, Ruoming Jin, and Yan Huang. Distance preserving graph simplification. In *2011 IEEE 11th International Conference on Data Mining*, pages 1200–1205. IEEE, 2011.

[14] Yuanbo Li, Qirun Zhang, and Thomas Reps. Fast graph simplification for interleaved-dyck reachability. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 44(2):1–28, 2022.

[15] Sean Yaw, Richard S Middleton, and Brendan Hoover. Graph simplification for infrastructure network design. In *International Conference on Combinatorial Optimization and Applications*, pages 576–589. Springer, 2019.

[16] Arend Rensink and Eduardo Zambon. Pattern-based graph abstraction. In *Lecture Notes in Computer Science*, pages 66–80. Springer Berlin Heidelberg, 2012.

[17] Francesco Bonchi, Gianmarco De Francisci Morales, Aristides Gionis, and Antti Ukkonen. Activity preserving graph simplification. *Data Mining and Knowledge Discovery*, 27:321–343, 2013.

[18] Iovka Boneva, Arend Rensink, Marcos E Kurban, and Jörg Bauer. Graph abstraction and abstract graph transformation. *Measurement Science Review-MEAS SCI REV*, 1, 2007.

[19] Marta Kwiatkowska, Gethin Norman, and David Parker. Verifying randomized distributed algorithms with prism. In *Workshop on advances in verification (WAVE'00)*, 2000.

[20] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[21] Christel Baier, Manuela Berg, and Walter Nauber. Formale systeme ws 2011/2012 skript zur vorlesung. 08 2011.

[22] Dimitrios Michail, Joris Kinable, Barak Naveh, and John V. Sichi. Jgrapht—a java library for graph data structures and algorithms. *ACM Transactions on Mathematical Software*, 46(2):1–29, may 2020.

[23] Jayme L Szwarcfiter and Peter E Lauer. A search strategy for the elementary cycles of a directed graph. *BIT Numerical Mathematics*, 16(2):192–204, 1976.

[24] Micha Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67–72, 1981.

[25] Harold N Gabow. Path-based depth-rst search for strong and biconnected components. *Information Processing Letters*, 2000.

A Appendix

A.1 Java Code of Distance Views

```
protected List<String> groupingFunction() throws Exception {
    List<String> toExecute = new ArrayList<>();

    // Compute reachability score
    Set<Long> visited = new HashSet<>();
    Set<Long> visiting = new HashSet<>();
    long distance = 0;

    // initialise states with expression
    if (identifierExpression.equals("init")){
        Set<Long> subsetInitStates = model.getInitialStates()
        .stream()
        .filter(stateId -> relevantStates.contains(stateId))
        .collect(Collectors.toSet());
        visiting.addAll(subsetInitStates);
    } else {
        Set<Long> subsetStates = model.getStatesByExpression(model.
           parseSingleExpressionString(identifierExpression).
           toString())
        .stream()
        .filter(stateId -> relevantStates.contains(stateId))
        .collect(Collectors.toSet());
        visiting.addAll(subsetStates);
    }
    MdpGraph mdpGraph = model.getMdpGraph();

// determine distance from Expression states (both ways)
while(!visiting.isEmpty()){
    Set<Long> toVisit = new HashSet<>();

    for (Long stateID : visiting){
        long curr = distance - Math.floorMod(distance, granularity)
           ;

        // create SQL statement
        toExecute.add(String.format("UPDATE %s SET %s = '%s' WHERE
           %s = '%s'", model.getStateTableName(), getCollumn(),
           curr, ENTRY_S_ID, stateID));

        Set<Long> reachableStates;

        // see all outgoing (depending Direction Mode)
        switch (direction) {
            case BACKWARD:
            reachableStates = mdpGraph.incomingEdgesOf(stateID)
            .stream()
            .map(mdpGraph::getEdgeSource)
            .filter(stateId -> relevantStates.contains(stateId))
            .collect(Collectors.toSet());
            break;
```

```
            case DIRECTIONLESS:
            reachableStates =
            mdpGraph.outgoingEdgesOf(stateID)
            .stream()
            .map(mdpGraph::getEdgeTarget)
            .filter(stateId -> relevantStates.contains(stateId))
            .collect(Collectors.toSet());
            reachableStates.addAll(
            mdpGraph.incomingEdgesOf(stateID)
            .stream()
            .map(mdpGraph::getEdgeSource)
            .filter(stateId -> relevantStates.contains(stateId))
            .collect(Collectors.toSet())
            );
            break;

            case FORWARD: default:
            reachableStates = mdpGraph.outgoingEdgesOf(stateID)
            .stream()
            .map(mdpGraph::getEdgeTarget)
            .filter(stateId -> relevantStates.contains(stateId))
            .collect(Collectors.toSet());
        }

        for (Long idReachableState : reachableStates) {
            if (!(visited.contains(idReachableState) ||
               visiting.contains(idReachableState))){
                toVisit.add(idReachableState);
            }
        }
    }

    visited.addAll(visiting);
    visiting = toVisit;
    distance++;
    }

    // compute not reachable states
    Set<Long> not_reachable = new HashSet<>(model.getAllStates())
    .stream()
    .filter(stateId -> relevantStates.contains(stateId))
    .collect(Collectors.toSet());
    not_reachable.removeAll(visited);

    // compute SQL statements for not reachable states
    for (long stateID : not_reachable){
        String reachability = semiGrouping ? ENTRY_C_BLANK : "inf";
        toExecute.add(String.format("UPDATE %s SET %s = '%s' WHERE
           %s = '%s'", model.getStateTableName(), getCollumn(),
           reachability, ENTRY_S_ID, stateID));
    }
    return toExecute;
}
```

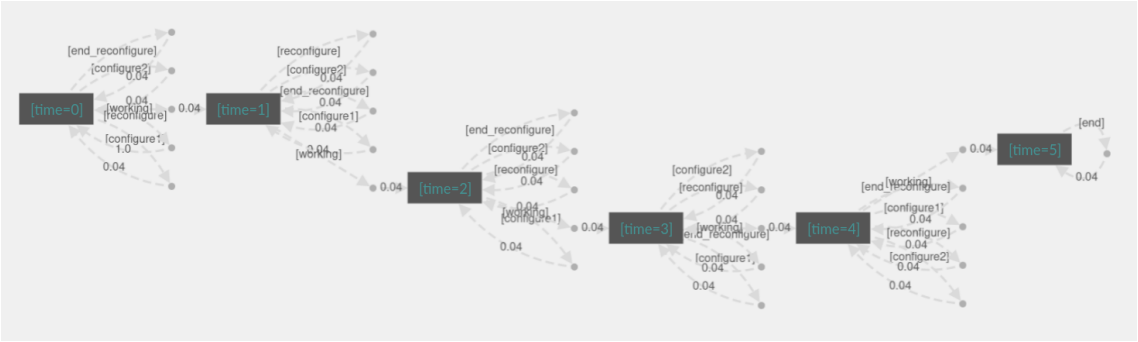
Listing 7: grp Function java

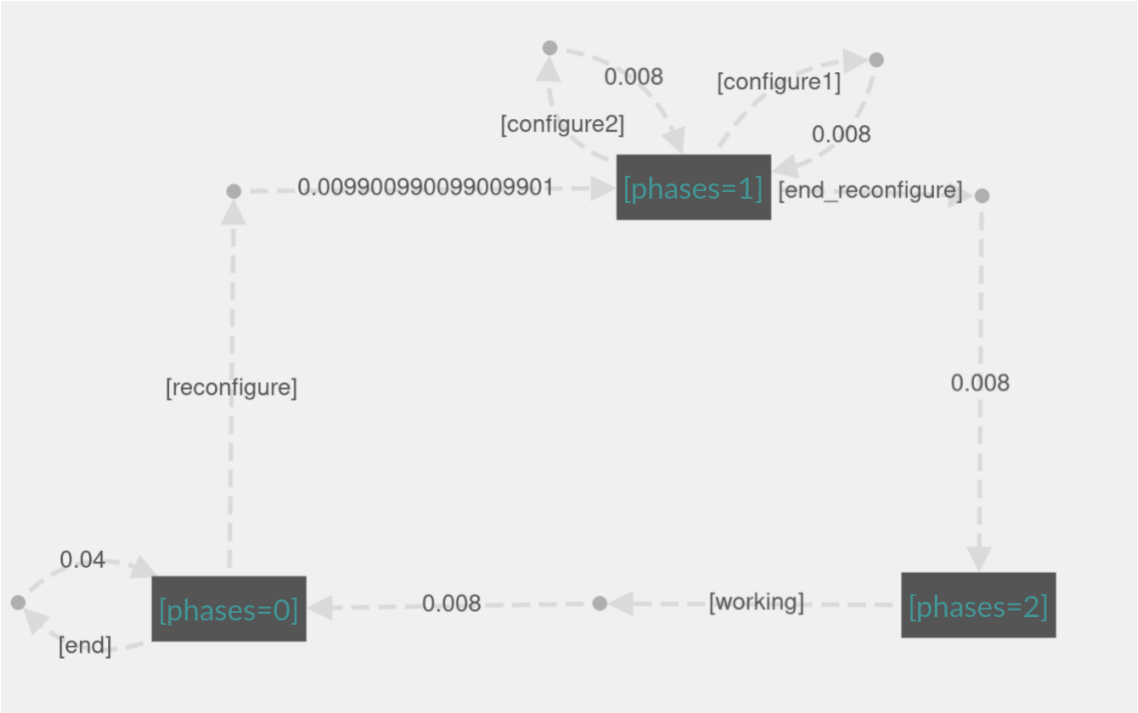
A.2 PRISM-File for MDP of Dining Philosophers

```
formula lfree = (p2>=0&p2<=4)|p2=6|p2=10;
formula rfree = (p3>=0&p3<=3)|p3=5|p3=7|p3=11;

module phil1

p1: [0..11];

[] p1=0 -> (p1'=0); // stay thinking
[] p1=0 -> (p1'=1); // trying
[] p1=1 -> 0.5 : (p1'=2) + 0.5 : (p1'=3); // draw randomly
[] p1=2 &  lfree -> (p1'=4); // pick up left
[] p1=2 &  !lfree -> (p1'=2); // left not free
[] p1=3 &  rfree -> (p1'=5); // pick up right
[] p1=3 &  !rfree -> (p1'=3); // right not free
[] p1=4 &  rfree -> (p1'=8); // pick up right (got left)
[] p1=4 & !rfree -> (p1'=6); // right not free (got left)
[] p1=5 &  lfree -> (p1'=8); // pick up left (got right)
[] p1=5 & !lfree -> (p1'=7); // left not free (got right)
[] p1=6  -> (p1'=1); // put down left
[] p1=7  -> (p1'=1); // put down right
[] p1=8  -> (p1'=9); // move to eating (got forks)
[] p1=9  -> (p1'=10); // finished eating and put down left
[] p1=9  -> (p1'=11); // finished eating and put down right
[] p1=10 -> (p1'=0); // put down right and return to think
[] p1=11 -> (p1'=0); // put down left and return to think

endmodule

// construct further modules through renaming
module phil2 = phil1 [ p1=p2, p2=p3, p3=p1 ] endmodule
module phil3 = phil1 [ p1=p3, p2=p1, p3=p2 ] endmodule
```

Listing 8: PRISM model file for MDP of Dining Philosophers

A.3 Large Figures from Figure 33 in Chapter 6





A.4 Timer Class and TimeSaver Class with Usage Example

```
package prism.misc;

import java.time.LocalTime;

public class Timer implements AutoCloseable {
    TimeSaver timeSaver;

    public Timer(TimeSaver timeSaver){
        this.timeSaver = timeSaver;
        timeSaver.storeStartTime(LocalTime.now());
    }

    @Override
    public void close() {
        timeSaver.storeEndTime(LocalTime.now());
    }
}

}
```

Listing 9: Timer.java

```
        import prism.misc.Timer;
        import prism.misc.TimeSaver;

        class Testtimer {
            public static void main(String[] args) {

                TimeSaver timeSaver = new TimeSaver();

                for (int i = 0; i < 100; i++) {
                    try(Timer timer = new Timer(timeSaver)) {
                        testFunction();
                    }
                }
                System.out.println(timeSaver.getAverageDuration());

            }

            private void testFunction(){
                // do something
            }
        }
```

Listing 10: Example for measuring time with Timer

```java
package prism.misc;

import java.util.*;
import java.time.LocalTime;
import java.time.Duration;

public class TimeSaver {

    private List<LocalTime> startTimes = new ArrayList<>();

    private List<LocalTime> endTimes = new ArrayList<>();

    public double getAvgDurationInMs() {
        long amountSamples;
        if (startTimes.size() != endTimes.size()) {
            throw new RuntimeException("startTimes.size() !=
                endTimes.size()");
        } else {
            amountSamples = startTimes.size();
        }
        long durationAccLong = 0;
        for (int i = 0; i < amountSamples; i++) {
            durationAccLong += getDurationInMsAt(i);
        }
        return ((double)durationAccLong) / ((double)amountSamples);
    }

    public long getDurationInMsAt(int i) {
        return Duration.between(startTimes.get(i), endTimes.get(i))
            .toMillis();
    }

    public void storeStartTime(LocalTime startTime) {
        this.startTimes.add(startTime);
    }

    public void storeEndTime(LocalTime endTime) {
        this.endTimes.add(endTime);
    }

}
```

Listing 11: TimeSaver.java