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 23, 2023

Dipl. Inf Max Korn (Theoretical Computer Science)

Contents

Abstract

Lorem ipsum

1 Introduction

The modern world heavily relies on ICT (Information and Communication Technology) 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 live saving ones utilized in medicine. Apart from performance, provided features and functionality, one of the most relevant aspects of such systems is their faulty free behavior, when active or running. The effect of a system fault can reach from a small disturbance in the user experience to inoperability of the whole system, the effect of which could be an annoyed user, a financial damage of several millions or a lost live. To prevent these possibly severe negative effects, methods to verify the correct behavior of a system are needed.

Apart from in practice often used approaches like peer review and testing for software and emulation and simulation for hardware, formal methods can be used as well 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 utilized in various ways for verification: the state space can be explored exhaustively, only specific 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 also stochastic phenomena are occurring. The major limitation for algorithms running on the set of states of models, in order to model check them, is the state explosion problem. The state explosion problem states that the number of states grow exponentially in the number of variables in a program graph or the number of components in concurrent systems [1, ch. 2.3]. Already simple system models can lead to complex system behavior and an immense amount of states. This is not only a problem for algorithms but also humans who need to understand, analyze and review 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]. A prominent approach is multiple coordinated view setups featuring parallel coordinates plots (PCPs) [5]. But even these techniques have their limitations for very large amounts of data as they are easily reachable with models used in model checking. The pure data volume is left unaffected.

In this thesis we want to preprocess data, that is then utilized in 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 issues.

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 size of single samples (feature selection, space transformations) or the amount of samples (instance reduction). There are several methods to reduce the amount 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 of 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 simplify 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 for 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 the other stepwise 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, which will be called view, that is similar to abstraction [1, pp. 499]. It defines an equivalence relation on the set of states which share certain traits. Its intention is to show humans interacting with the visualization as much information possible in 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 on where and how the proposed views of 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. Moreover there will be an overview about performance and scalability of the proposed views.

2 Preliminaries

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

Firstly 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 every state there may be several possible transitions to be taken. The choice of which one is take is done nondeterministically. That is the outcome of the selection can not be know a priori. It is especially not following any probability distribution. Hence there can not be made any statement about the likelihood of a transition being selected.

In contrast with *Markov Chains* this nondeterministic behavior is replaced with a probabilistic one. That is for every state there exists a probability distribution that describes the chance of a transition being selected. There are no actions and there is 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. For finite $\mathcal{M}$, the *size* of $\mathcal{M}$, denoted *size*($\mathcal{M}$), is the number of states plus the number of pairs $(s, s') \in S \times S$ with $\mathbf{P}(s, s') > 0$. OMIT??

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 likelihood that the system evolution starts in s. All states s with $\iota_{init}(s) > 0$ are considered *initial states*. States s' with $\mathbf{P}(s, s') > 0$ are viewed as possible successors of the state s. The operational behavior is as follows. An initial state s_0 is yielded by the initial distribution ι_{init}. 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) permit both probabilistic and nondeterministic choices. An MDP thereby is 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 set of ι_{init} with nondeterministic choice at the beginning. For sake of simplicity there is just one single distribution. The operational behavior is as follows. A starting state s_0 yielded by ι_{init} with $\iota_{init}(s_0) > 0$. From there on nondeterministic choice of enabled action takes place followed by a probabilistic choice of a state. The action fixed in the step of nondeterministic selection. Any 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 for $s_1, s_2 \in S$ and $\alpha \in Act$ we will write $(s_1, \alpha, s_2) \in \mathbf{P}$ if and only if $\mathbf{P}(s_1, \alpha, s_2) > 0$, that is to say if there is a non-zero probability of evolving from state s_1 to state s_2 with action α. Analogously we will write $s \in \iota_{init}$ if and only if is an initial state ($\iota_{init}(s) > 0$). 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 a state has an outgoing action α 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 and transition probability function of 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.

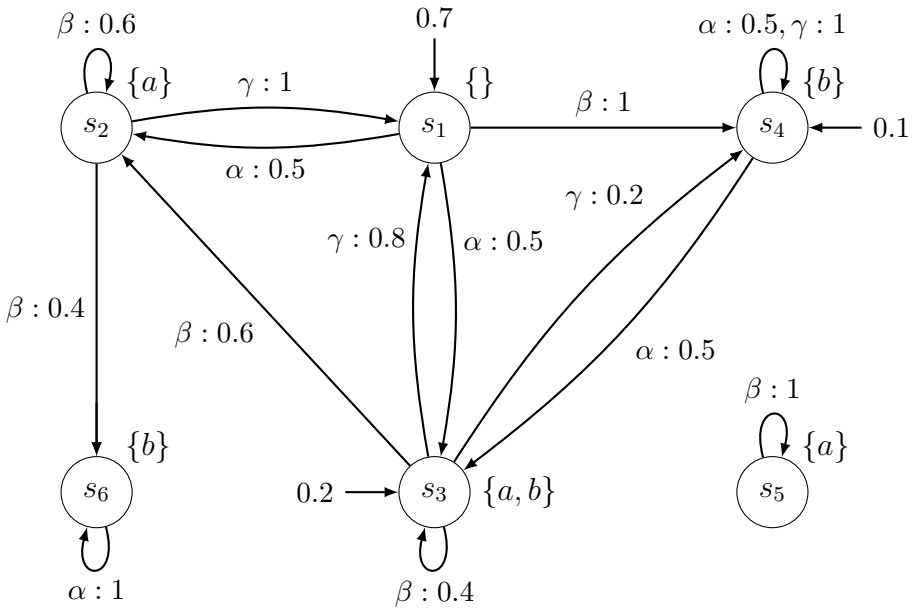

Figure 1: Simplified representation of $\mathcal{M}$ (left) and the view ${\mathcal{M}_I}^\top$ on it(left)

In the implementation and evaluation in 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, which interact with each other. A module consists of variables and commands. The (current) values of all the 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

```
mdp

module onlymodule

x : [1..6];

[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


init
    x=1 | x=3 | x=4
endinit
```

Listing 1: PRISM model file for the MDP $\mathcal{M}$ given by Figure 1 initials not by probability!. 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.

functions. An `update` represents a transition in the model which normally reflects a change of state. As a state is defined by the value 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 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 has to be in parentheses and separated with `&` from other assignments. The `action` can be included optionally for labeling the command or for synchronization purposes. In Listing 1 the model file for the MDP $\mathcal{M}$ from Figure 1 is shown.

Actions cause synchronization when included in several modules. If for example `action1` is included in two commands, which are in distinct modules these will be chosen simultaneously. If the guard of one of the commands with `action1` is not true, neither of the commands can be chosen [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. Thereby the original MDP is retained.

In the preliminaries transition systems and Markov Chains were listed as simpler version of MDPs. Roughly speaking transition systems and MDPs are special MDPs namely 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 of consideration.That is why we will define views on MDPs. Only for specific views and their implementation it is to be kept in mind that if they regard an action or the probability distribution of an action in a state it is not applicable to transitions systems or MCs respectively.

3.1 Grouping Function

The conceptional 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 only 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 their codomain of F_θ and $\tilde{F}_\theta$ respectively. The identifier θ normally hints 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 if that value is not the unique symbol $\bullet$. The mapping to the symbol $\bullet$ will be used whenever a state is supposed to be excluded from the grouping. The definition offers an easy way of creating groups of states. The exact mapping depends on the desired grouping. 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 3.3. 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 $\bullet$.

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 $R := \{(s_1, s_2) \in S \times S \mid F_\theta(s_1) = F_\theta(s_2) \notin \xi\} \cup \{(s, s) \mid s \in S\}$

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 $F_\theta(s_2) = F_\theta(s_1) \in \xi$ or $F_\theta(s_2) \neq F_\theta(s_1)$. In both cases it follows from definition of R that $(s_1, s_2) \notin R$. If $F_\theta(s_2) \in \xi$ it directly follow 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 $(s_1, s_2) \notin R$. Hence when considering transitivity and symmetry we assume $F_\theta(s_1), F_\theta(s_2) \notin \xi$. If $F_\theta(s_1) = F_\theta(s_2)$ it is obviously also $F_\theta(s_2) = F_\theta(s_1)$, which means if $(s_1, s_2) \in R$ it follows that $(s_2, s_1) \in R$. Hence R is symmetric. The property directly conveyed from equality. In the same way transitivity directly conveys 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 respective 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 $\tilde{F}_\theta : \tilde{S} \to M$ it 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

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

- $\iota_{init}{}' : S/R \to [0, 1]$ 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 $\tilde{S} = S$. 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 will 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 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 is, because a state in a view is an equivalence class and it may be that in it there are several states with the same outgoing action. Hence, summing up the probabilities for two given equivalence classes and action α may yield a value greater than one (no probability anymore). This is why the sum of probabilities has to be normalized, by dividing by the number of states that have the action α outgoing.

Note that the definition is in a most general form in the sense that if in a view a property accounts to one piece 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 $\mathcal{M}_\theta$ on the MDP $\mathcal{M}$ with $S = \{s_1, \ldots, s_6\}$ in Figure 1. The $\mathcal{M}_\theta$ view 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 $\tilde{S} = S \backslash \{s_2\}$ 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 $F_\theta(s_2) = F_\theta(s_6) = \bullet \in \xi$. Because in $\mathcal{M}$ it is

$$\begin{array}{ll} \iota_{init}(s_1) = 0.7 & \iota_{init}(s_4) = 0.1 \\ \iota_{init}(s_3) = 0.2 & \iota_{init}(s_5) = 0 \end{array}$$

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

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

Because there are no other other outgoing transitions from s_1 or s_3 with action β in $\mathcal{M}_\theta$ it is

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

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 outgoing from a state in $[s_1]_R$ and incoming to a state in $[s_1]_R$, $[s_2]_R$ or $[s_4]_R$ respectively. 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.

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. Given a grouping function $F_\theta : S \to M$ or a detached grouping function $\tilde{F}_\theta : \tilde{S} \to M$

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 $F_S = \{\top, \bot\}$ 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 of expanding grouped states and show the ones they contain. Hence, at runtime it can be decided which states are to be shown, rather than in the phase of preprocessing.

The notion of a categorizing view is that categorizes states to groups, which have a certain property. One could argue that binary views as well categorize states in

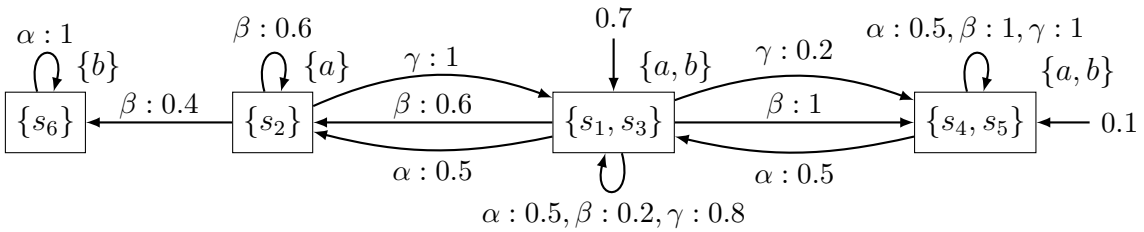

Figure 2: View $\mathcal{M}_\theta$ on MDP $\mathcal{M}$ from Figure 1

two groups that have in common to share having or not having a certain property. We defined categorizing views as in Definition 3.6 so that a view either is binary or categorizing to distinguish between the intention of a view.

For binary $\mathcal{M}_\theta$ we already presented three cases in which a view is called binary. We will now further elaborate on those cases and generalize the concept and also enable it with categorizing views. When looking at the Elements of the set $\{\{\top, \bot\}, \{\bullet, \bot\}, \{\top\, \bullet\}\}$ we observe that for distinct s, t in the set it is $|s| - |s \cap t| = 1$, i.e. s and t only differ in one element. We declare the case when $\bullet \in M(F_\theta)$ a special case, because instead of assignment of $\top$ or $\bot$, the symbol $\bullet$ is assigned, to avoid grouping on states with this property. In general it may be that grouping on states that would be mapped to a certain value should not be grouped. To accomplish 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 - maybe amongst others if $\tilde{S} \subset S$. For this thesis we will only consider disregarding views for $n = 1$. Normally we will define binary views as the disregarding ${\mathcal{M}_\theta}^{\top}$ and perceive them as filters that show use states that have that property. 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 it suffices to give one Definition for binary view. For categorizing views we normally wont use disregarding views.

3.4 Composition of Views

In essence views are a simplification generated from MDP. It seems rather obvious that the composition of views is a very practical feature, in order 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 partaking view also takes effect in the composed view. Moreover it is to be generated in a way that the effect of each individual view can be reverted from the composed one.

The fundamental concept for composition, which 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 a set of grouping function is combined to a new grouping function in one single step.

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 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

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

Then it is

$$F_{v||u}(s_1) = (F_v(s_1), F_u(s_1)) = (b, a)$$
$$F_{v||u}(s_2) = (F_v(s_2), F_u(s_2)) = (y, x)$$

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

We assume that whenever a view or grouping function is parallel composed this 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 $||$ in order 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 $\mathcal{M}_{\theta_1 || \theta_2 || \ldots || \theta_n}$ be a parallel composed view with its grouping function $F_{\theta_1 || \theta_2 || \ldots || \theta_n}$, 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 if

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

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 get to know some views that might be useful in real world applications.
When considering views we only want to take into account those that utilize properties of MDPs or that do computations that are also feasible on normal graphs but are of explicit relevance MDPs.

4.1 Views Utilizing MDP Characteristics

In this subsection we will introduce some views the are purely based on the components of an MDP. Their will neither be computations on the graph-structure of an MDP nor computations using the result vector.

4.1.1 Atomic Propositions

One of the least involved approaches to create a view is to base it on the atomic propositions that are 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 $\tilde{F}_{AP} : \tilde{S} \to M, s \mapsto L(s)$.

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 $F_{AP}(s_1) = F_{AP}(s_2) \iff L(s_1) = L(s_2)$. According to Definition 3.4 for $\tilde{s} \in S$ it is $[\tilde{s}]_R = \{s \in S \mid L(s) = L(\tilde{s})\}$.

By this 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.

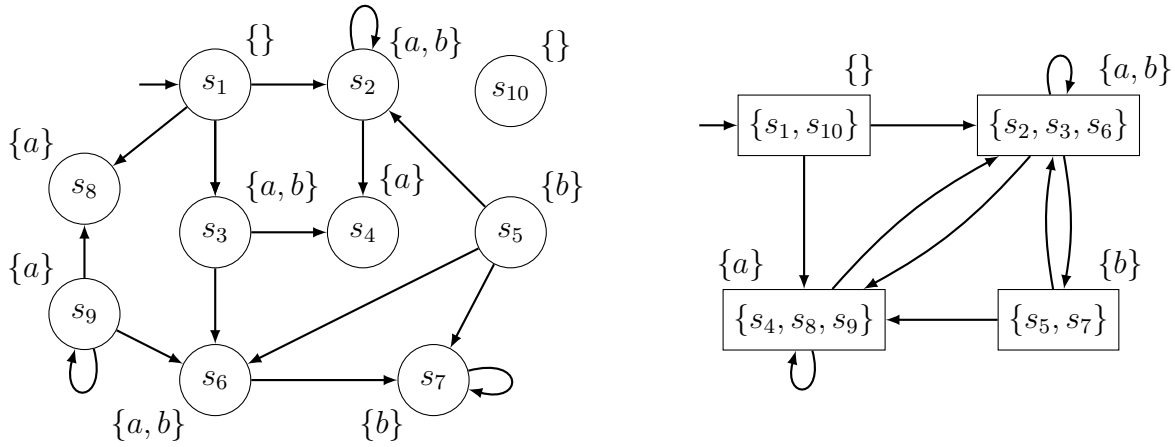

Figure 3: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{AP}$ on it (right)

In Figure 3 we can observe the effect of $\mathcal{M}_{AP}$. 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 might seem rather simple because essentially it only performs $F_{AP} := L$ it is the most powerful one. This is because every view presented in the following is reducible to this one. That is because a grouping function essentially asserts an atomic proposition to every state, namely the value with respect to the considered property of a given view. The reduction can be realized by replacing the labeling function with the grouping function of the resepective 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.

4.1.2 Initial States

An a little more involved idea than directly using a given function is to utilize the set of initial states. We can group states that have a probability greater zero, that they are started from. In practice this might 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 $I := \{s \in S \mid s \in \iota_{init}\}$. 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 $M := \{\bullet, \bot\}$.

For $s_1, s_2 \in S$ it is ${F_I}^\top(s_1) = {F_I}^\top(s_2)$ if and only if $s_1, s_2 \notin I$ or $s_1 = s_2$. According to Definition 3.4 it is

$$\begin{aligned} [s]_R &= \{s \in S \mid {F_I}^\top(s) = \bullet\} = \{s\} && \text{for } s \in I \text{ and} \\ [s]_R &= \{s \in S \mid {F_I}^\top(s) = \bot\} && \text{for } s \notin I. \end{aligned}$$

By this 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.

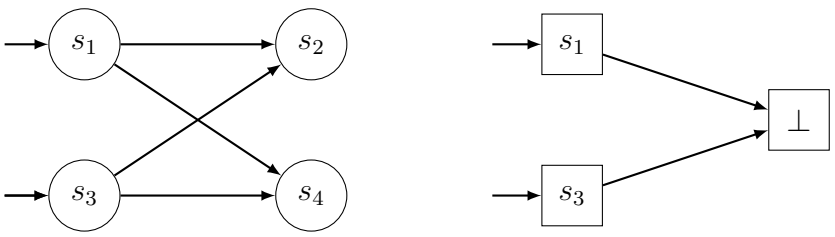

Figure 4: Simplified representations of $\mathcal{M}$ (left) and the view ${\mathcal{M}_I}^\top$ on it (right)

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 interprocesscommunication und synchronization. In this subsection we will provide and discuss some views utilizing actions on transitions that are outgonig 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 $M := \{\bullet, \bot\}$.

For $s_1, s_2 \in S$ it is ${F_{\exists\overrightarrow{\alpha}}}^\top(s_1) = {F_{\exists\overrightarrow{\alpha}}}^\top(s_2)$ if and only if there exist $s_a, s_b \in S$ with $(s_1, \alpha, s_a), (s_2, \alpha, s_b) \in \mathbf{P}$ (i.e. they have α as outgoing action). In accordance with Definition 3.4 it is

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

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

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

In Figure 5 we can observe the effect of ${\mathcal{M}_{\exists\overrightarrow{\alpha}}}^\top(\alpha)$. 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 grouping states that only *have* outgoing actions we could also 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.

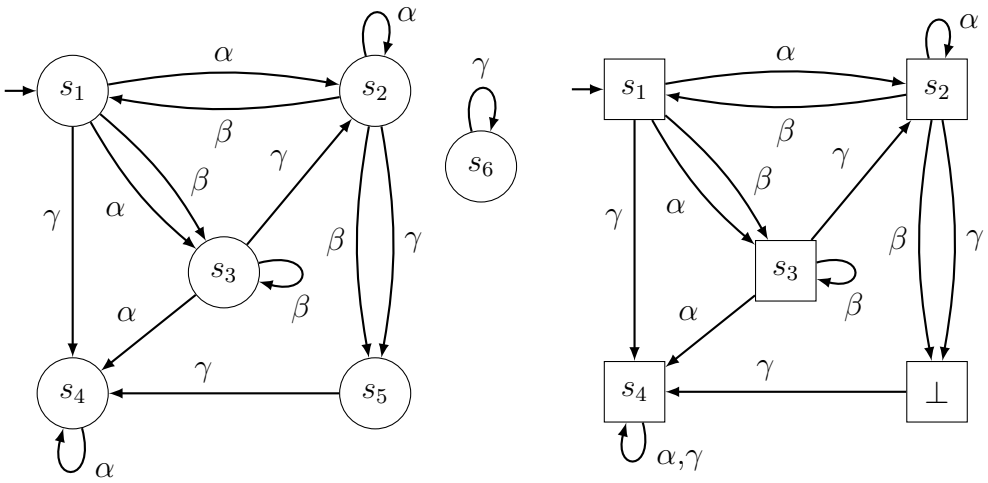

Figure 5: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\exists\overrightarrow{\alpha}}{}^{\top}(\alpha)$ on it (right)

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

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

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

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

is a first order logic predicate.

The predicate $Q_{n\leq\overrightarrow{\alpha}}(s, s_1, \ldots, s_n)$ requires that there are transitions with action α to n distinct states.

For $s_1, s_2 \in S$ it is $F_{n\leq\overrightarrow{\alpha}}{}^{\top}(s_1) = F_{n\leq\overrightarrow{\alpha}}{}^{\top}(s_2)$ if and only if there exist distinct $s_{a_1}, \ldots, s_{a_n} \in S$ and distinct $s_{b_1}, \ldots, s_{b_n} \in S$ so that $(s_1, \alpha, s_{a_1}), \ldots, (s_1, \alpha, s_{a_n}) \in \mathbf{P}$ and $(s_2, \alpha, s_{b_1}), \ldots, (s_2, \alpha, s_{b_n}) \in \mathbf{P}$ or $s_1 = s_2$. According to Definition 3.4 it is

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

By this 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

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

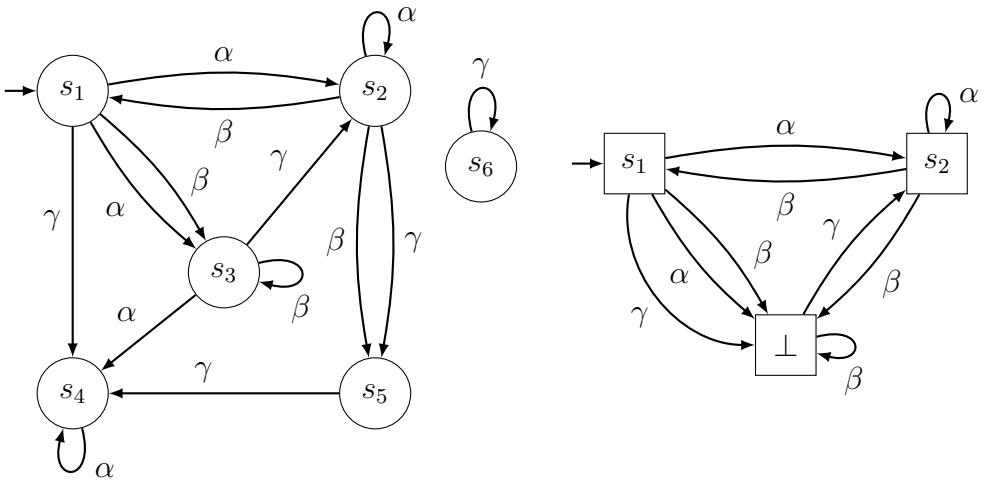

Figure 6: Simplified representations of $\mathcal{M}$ (left) and the view ${\mathcal{M}_{n\leq\overrightarrow{\alpha}}}^{\top}(\alpha, 2)$ on it (right).

In Figure 6 we can observe the effect of ${\mathcal{M}_{n\leq\overrightarrow{\alpha}}}^{\top}(\alpha, 2)$. 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 $s_3, \ldots, s_6$ are grouped.

In a similar fashion we define view that groups states where at most a certain number of times a given action is outgoing.

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.

It ensures that if there are one more than n outgoing transitions with an action α at least two of the states where the transitions end are in fact the same. Since this is required for all possible combinations of $n+1$ states by the grouping function, only states that have at most n outgoing actions will be assigned with $\bullet$ by the grouping function. 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 ${\mathcal{M}_{n\leq\overrightarrow{\alpha}}}^{\top}$.

In Figure 7 we can observe the effect of ${\mathcal{M}_{\overrightarrow{\alpha}\leq n}}^{\top}(\alpha, 1)$. In the simplified representation of an MDP on the left the action α is outgoing zero times from s_5, s_6, one

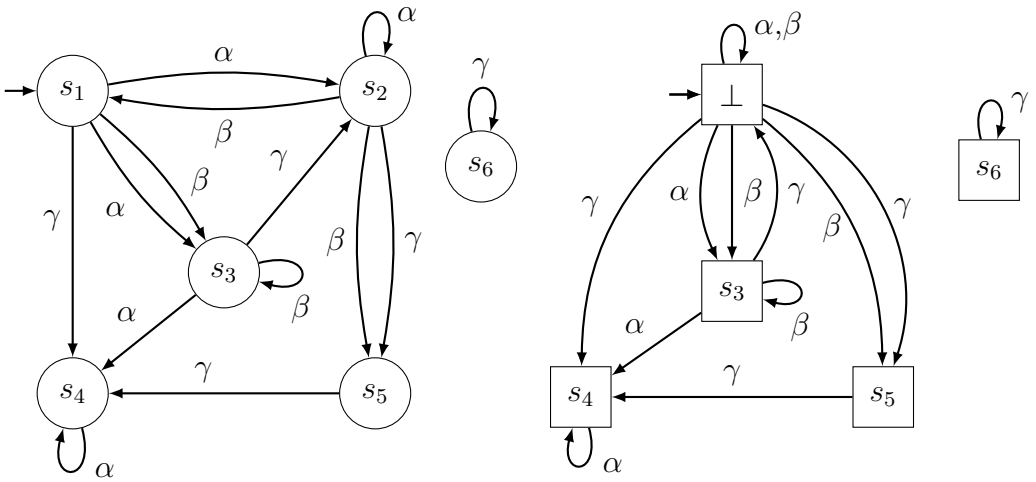

Figure 7: Simplified representations of $\mathcal{M}$ (left) and the view ${\mathcal{M}_{\overrightarrow{\alpha}\leq n}}^{\top}(\alpha, 1)$ on it (right)

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

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

where $M := \{\bullet, \perp\}$ and $m, n \in \mathbb{N}$ are the minimal and maximal number of transitions with action α in order for 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.

We already know that 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 minimal and maximal amount of outgoing transitions with an action α respectively. 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 for this that the number of outgoing actions is *in the span*.

For a given state s and action α we set

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

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

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

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 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

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

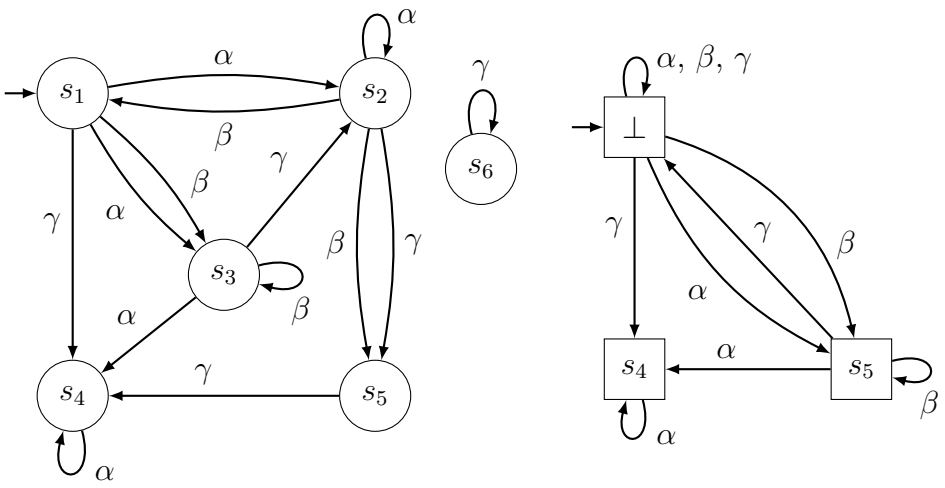

Figure 8: Simplified representations of $\mathcal{M}$ (left) and the view ${\mathcal{M}_{m\leq\vec{\alpha}\leq n}}^{\top}(1, \alpha, 1)$ on it (right)

In Figure 8 we can observe the view of ${\mathcal{M}_{m\leq\vec{\alpha}\leq n}}^{\top}(1, \alpha, 1)$ 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 about states and group them based on whether they meet these requirements it also possible to group states that are very similar or even identical in regard to their outgoing action. Firstly we will consider the case where the set of outgoing actions and their quantity is identical.

Definition 4.7. 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 $\tilde{F}^{\top}_{\overrightarrow{Act(s)_=}} : \tilde{S} \to M$ with

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

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

The grouping function asserts to each state a set of pairs. Note that a pair is contained in the set for each action $\alpha \in Act$. In case there is no outgoing transition from state s with an action α it is $(\alpha, 0) \in F_{\overrightarrow{Act(s)_=}}$. For $s_1, s_2 \in S$ it is $F_{\overrightarrow{Act(s)_=}}(s_1) = F_{\overrightarrow{Act(s)_=}}(s_2)$ if and only if s_1 and s_2 are mapped to the same set of pairs. By Definition 3.4 the obtained equivalence classes of R are

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

According to 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 as usual structured in accordance with the Definition 3.5.

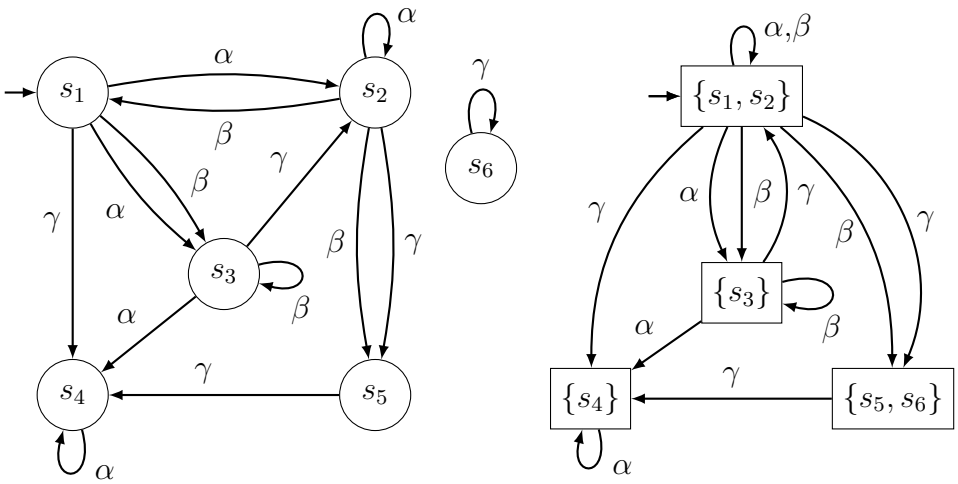

Figure 9: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\overrightarrow{Act(s)_=}}$ on it (right)

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. Hence, these two pairs are each grouped to a new to new states. The remaining states are not grouped with another state. The equivalences classes containing them are singleton sets.

As mentioned earlier a weak variant of the OutActionsIdent view is also conceivable. To ease readability for $s \in S$ we write $\overrightarrow{Act}(s)$ for the set $\{\alpha \in Act \mid \exists s' \in S : (s, \alpha, s') \in \mathbf{P}\}$

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

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

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

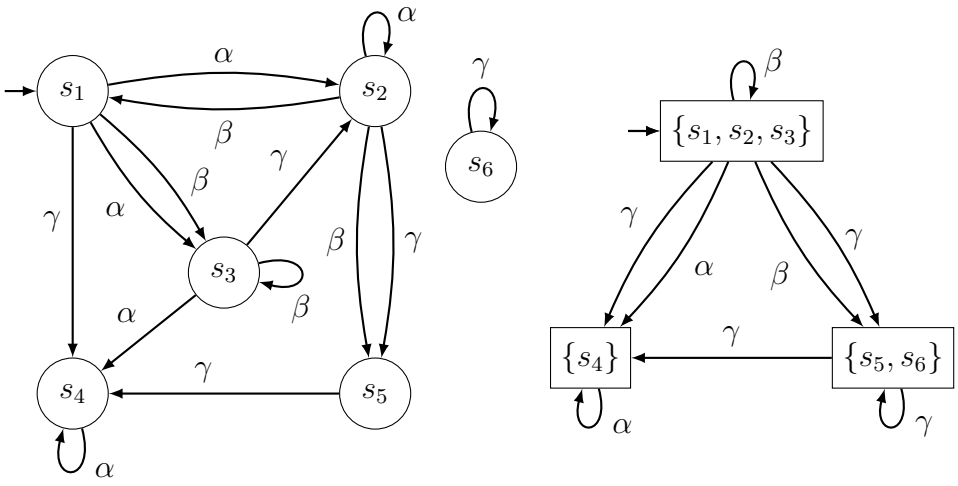

Figure 10: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\overrightarrow{Act}(s)\approx}$ on it (right)

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

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

$$\begin{aligned}[\tilde{s}]_R &= \{s \in S \mid F_{|\overrightarrow{Act}(s)|}(s) = F_{|\overrightarrow{Act}(s)|}(\tilde{s})\} \\ &= \{s \in S \mid \overrightarrow{Act}(s) = \overrightarrow{Act}(\tilde{s})\}\end{aligned}$$

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

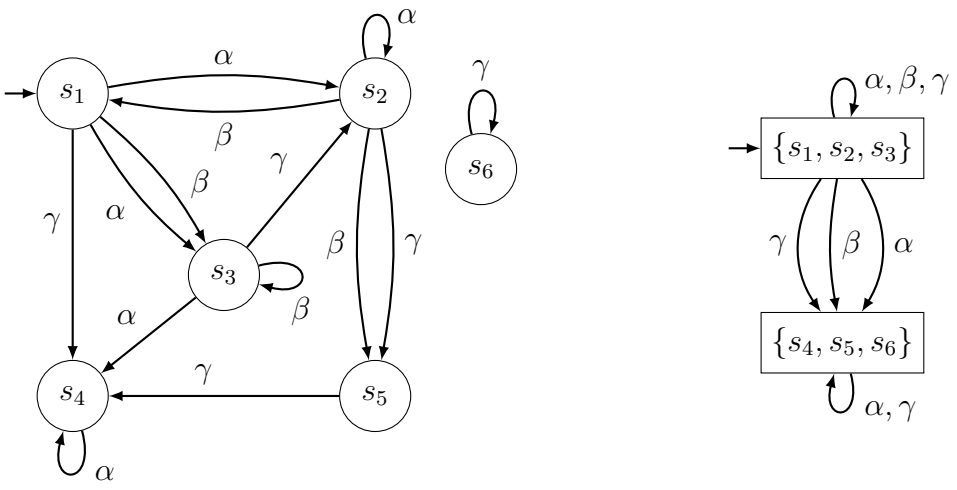

Figure 11: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{|\overrightarrow{Act}(s)|}$ on it (right)

In Figure 11 we can observe the view of $\mathcal{M}_{|\overrightarrow{Act}(s)|}$ 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. That is why each of these states become 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 sepcialcase of there only being one single ougoing action is worth a distinct view, since it hides all not nondeterministic 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 groups on states that *have* the property. This is the case because the intention of the view is to hide S in which there is no nondeterministic selection of an action.

Two states $s_1, s_2 \in S$ are grouped if and only if ${F_{\overrightarrow{|Act(s)|_1}}}^{\perp}(s_1) = {F_{\overrightarrow{|Act(s)|_1}}}^{\perp}(s_2)$. Hence the the equivalence classes of R are:

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

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

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

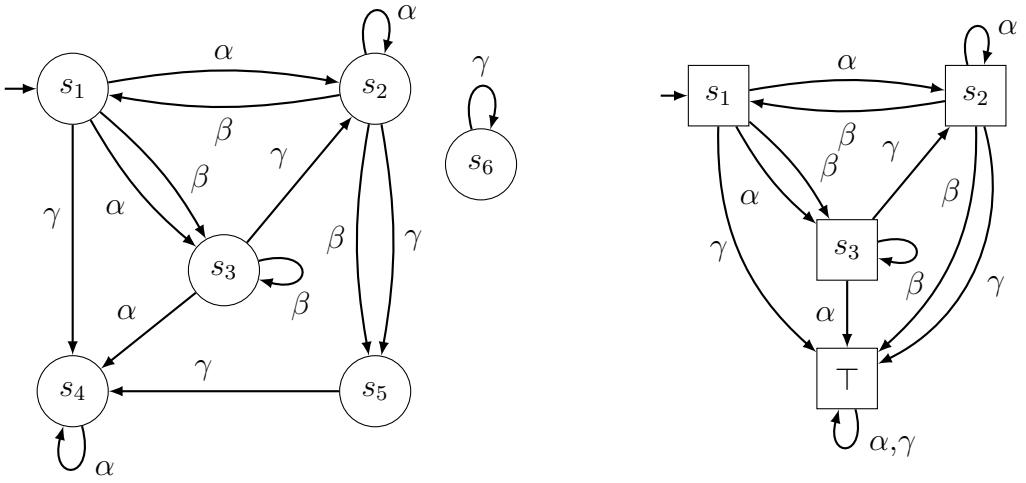

Figure 12: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\overrightarrow{|Act(s)|}}$ on it (right)

In Figure 12 we can observe the view of ${\mathcal{M}_{\overrightarrow{|Act(s)|_1}}}^{\perp}$ 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

Analogously to Outgoing Actions views of utilizing incoming actions are feasable. Since there is no difference apart from the definitions itself, 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 $F_{\exists \overleftarrow{\alpha}}{}^{\top}$ 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 maximal 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 minimal and maximal 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 $M := \mathbb{N} \cup \{\bullet\}$.

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 importance 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 carried is usually not atomic but rather consists of several pieces. For instance when considering a computer program at a given moment during execution all its currently available variables will have some value, the stack will have a certain structure and the program counter points to a specific instruction. Systems in general at a certain moment of their behavior have several properties that in total pose the current state of the system. That is in practice each state is actually derived from a possible variable assertion. Choosing the respective variable assertion 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 like an integer represents the actual state. When formalizing this in practice used approach several options come into consideration. Available MDP components like atomic propositions or the set of states could be used to contain this information. There could be a subset of atomic propositions that declares that a certain variable has a certain value. There had to be taken care of that for each variable in each state there is only one atomic proposition declaring its value. The set of states could be used in the sense that the states itself are complex objects containing the information. A third option is to define a set of variables and an evaluation function that are induced by the MDP.

Definition 4.18. 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 4.19. 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. 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 4.20. 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 $\mathcal{M}_{x=a}$ groups states that share the same value for a given variable. For s_1, s_2 it is ${F_{x=a}}^{\top}(s_1) = {F_{x=a}}^{\top}(s_2)$ if and only if $VarEval_{\mathcal{M}}(s_1, x) = VarEval_{\mathcal{M}}(s_2, x)$ or $s_1 = s_2$. The obtained equivalence classes 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

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

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 the requirement of several variables equaling or not equaling specified values this can be achieved by using parallel composition.

To allow even more flexibility a view can be used that 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 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 $\div$ to be an

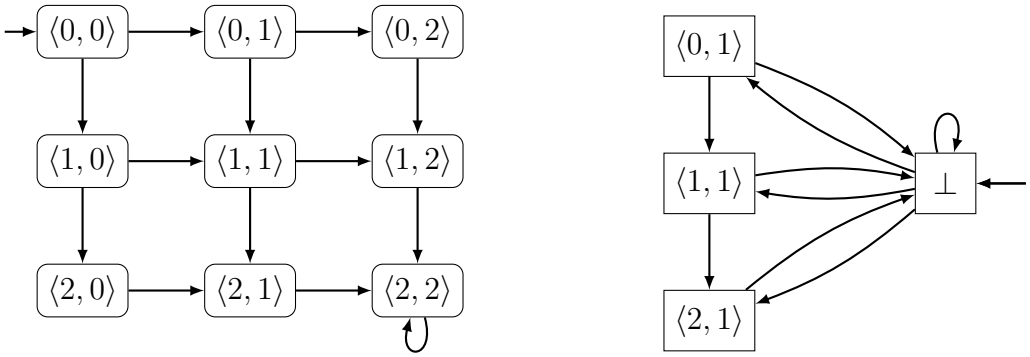

Figure 13: Simplified representations of $\mathcal{M}$ with the state set $S = \{\langle x, y\rangle \mid x, y \in \{0, 1, 2\}\}$ (left) and the view $\mathcal{M}_{x=a}(y = 1)$ on it (right)

element of the set $\{=, \neq\}$. That is to say, whenever it is used each time written it is a representative for either the symbol $=$ or $\neq$. It allows to write one symbol whenever $=$ and $\neq$ could or should be possible. Moreover 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 could be negated or not negated.

Definition 4.21. 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 requirement in disjunctive normal form about variables. States are mapped to the same value depending on whether or not they meet this requirement.

The view ${\mathcal{M}_{VarDNF}}^{\top}$ groups states that share the same value for a given variable. For s_1, s_2 it is ${F_{VarDNF}}^{\top}(s_1) = {F_{VarDNF}}^{\top}(s_2)$ if and only if $VarEval_{\mathcal{M}}(s_1, x) = VarEval_{\mathcal{M}}(s_2, x)$ or $s_1 = s_2$. The obtained equivalence classes 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

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

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

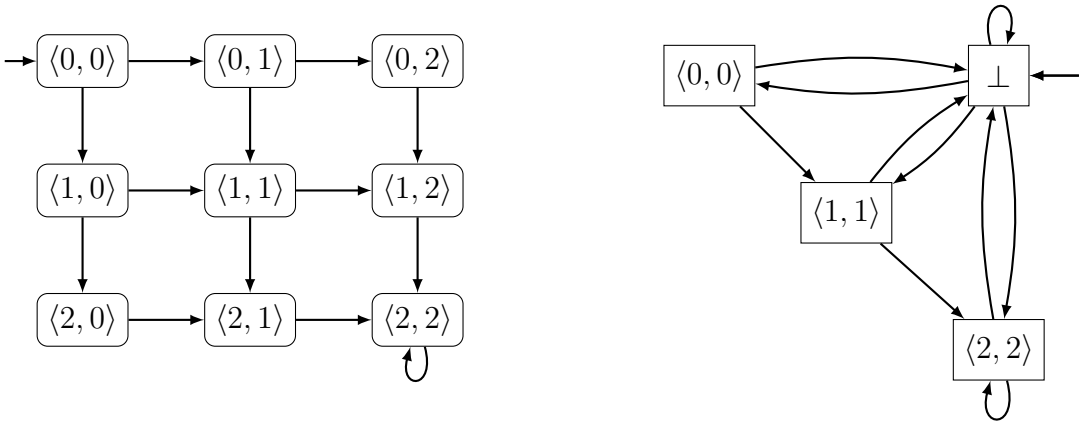

Figure 14: Simplified representations of $\mathcal{M}$ $S = \{\langle x, y\rangle \mid x, y \in \{0, 1, 2\}\}$ (left) and the view ${\mathcal{M}_{VarDNF}}^\top$ with $d(s) = ((x = 0) \wedge (y = 0)) \vee ((x = 1) \wedge (y = 1)) \vee ((x = 2) \wedge (y = 2))$ on it (right)

Analogously a view based on a conjunctive normal formal can be defined that may be more convenient, depending on the query.

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

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

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 from the view ${\mathcal{M}_{VarCNF}}^{\top}$ to the view ${\mathcal{M}_{VarDNF}}^{\top}$ is whether the respective formulae is in conjunctive or disjunctive normal form. Since each formulae in conjunctive normal form can be transformed to a formulae in disjunctive normal form and vice versa CITATION?? neither on of the views can perform an action the other can not. Hence there is no difference in expressivity, but there may be one in size. Therefore both views have been implemented and formalized. For finite MDPs they present an interface to performing arbitrary groupings on the state set utilizing 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 before 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 stipulate only the variable but not its value. This way states will be grouped that have the same value for that variable with no regard to the actual value of that variable.

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

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

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

With this grouping function we directly map to the value of the variable. 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

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

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

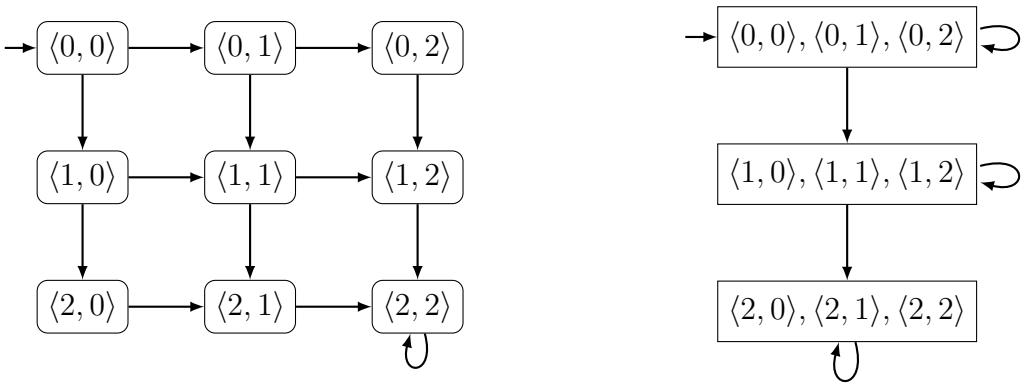

Figure 15: Simplified representations of $\mathcal{M}$ $S = \{\langle x, y\rangle \mid x, y \in \{0, 1, 2\}\}$ (left) and the view $\mathcal{M}_{Var:a}(x)$ on it (right).

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 $x \in \{0, 1, 2\}$. Hence, the states $\langle 0, 0\rangle, \langle 0, 1\rangle, \langle 0, 2\rangle$ are grouped because in all of them it is $x = 0$. For the same reason $\langle 1, 0\rangle, \langle 1, 1\rangle, \langle 1, 2\rangle$ grouped with $x = 1$ and $\langle 2, 0\rangle, \langle 2, 1\rangle, \langle 2, 2\rangle$ with $x = 2$.

4.1.6 Property Values and Model Checking Results

Apart from direct and indirect components of an MDP there are other values linked to states. These can be given by a reward structure or results from model checking. A reward structure simply assigns values to each state. Results from model checking are also available statewise. 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 utilize it to define a view. Note that there might be several such functions (e.g.reward structure and several model checking results) for an MDP.

Definition 4.24. The function $f : S \to \mathbb{R}$ is called *property function* (generated by a model checker). It maps each state to a property value.

Definition 4.25. 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 the by r induced partition $Z = \bigcup_{i \in \mathbb{Z}} Z_i$ 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 $M = \mathbb{R} \cup \{\bullet\}$.

With this view a partition on the image of f is built, based on the granularity r. 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 directly determined using 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 $F_f(s_1) = F_f(s_2) \neq \bullet$. Hence, the obtained equivalence classes are

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

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

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

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

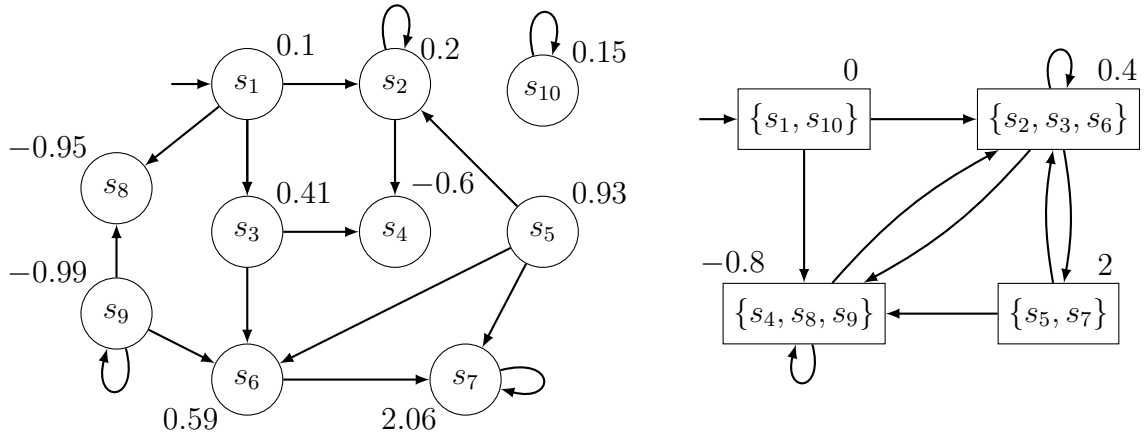

Figure 16: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_f(0.4)$ on it (right)

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 Utilizing the MDP Graphstructure

4.2.1 Distance

Considering 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 nodes we will need to formalize it. For this we will introduces *paths* which conceptually are very similar to execution fragments [1, Def. 2.4.].

Definition 4.26. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP. A (simple and finite) *path* P is a sequence $(s_0, \alpha_0, s_1, \alpha_1, \ldots, \alpha_n, s_{n+1})$ 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 4.27. 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 $(s_0, \alpha_0, s_1, \alpha_1, \ldots, \alpha_n, s_{n+1})$ 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 4.28. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and $P = (s_0, \alpha_0, s_1, \alpha_1, \ldots, \alpha_n, s_{n+1})$ be a path in $\mathcal{M}$. The number $n =: len(P)$ is called *length* of the path P.

Definition 4.29. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP. The *distance* between disjoint $S_1, S_2 \subseteq S$ is the 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

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

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 $\overrightarrow{dist}(\mathcal{M}, S_1, S_2) := 0$.

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

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

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

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 $\overrightarrow{dist}(\mathcal{M}, \tilde{S}, s)$ short for $\overrightarrow{dist}(\mathcal{M}, \tilde{S}, \{s\})$ and $\overrightarrow{dist}(\mathcal{M}, s, \tilde{S})$ short for $\overrightarrow{dist}(\mathcal{M}, \{s\}, \tilde{S})$. We do so in the same way for $\overline{dist}$ and $\overleftarrow{dist}$. For a view that applies to the whole and utilizes distance it only makes sense to consider a given set of states from which one the distance is measured. An intuitive choice for such set is the set of initial states. To determine the distance a simple breadth first search is used. There is no algorithm

such as Dijkstra need, since there are no edge weights apart from the probabilities. The implementation ensures that for every state s there exists a pair (s, d) in $distance(\mathcal{M}, \tilde{S}, n)$. The following view groups states that have the same distance to the set measured with the amounts of transitions necessary to reach the the closest $\tilde{S}$ 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 $d = \overrightarrow{dist}(\mathcal{M}, \bar{S}, s)$ and $M = \mathbb{N} \cup \{\infty\} \cup \{\bullet\}$.

The number n from the definition is the granularity of the distance cluster. For $n = 2$ we have for $d = 0 : 0 - 0 = 0$, for $d = 1 : 1 - 1 = 0$, for $d = 2 : 2 - 0 = 2$ and for $d = 3 : 3 - 1 = 2$ where each time the subtrahend is d mod 2. We see that, $F_{\overrightarrow{dist}}$ maps $\overrightarrow{dist}(\mathcal{M}, \tilde{S}, s)$ to next smaller natural number that is dividable by n, which means the granularity causes that $F_{\overrightarrow{dist}}(s)$ is a multiple of n.

In $\mathcal{M}_{\overrightarrow{dist}}$ two states $s_1, s_2 \in S$ are grouped if and only if $F_{\overrightarrow{dist}}(s_1) = F_{\overrightarrow{dist}}(s_2)$. Hence the equivalence classes in R are

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

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

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

In Figure 15 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

Can not get rid of left indent below in left minipage

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

This 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 $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 1)$ merge into one, as $F_{\overrightarrow{dist}}(s_i) = 0 - (0 \bmod 2) = 0 - 0 = 0 = 1 - 1 = 1 - (1 \bmod 2) = F_{\overrightarrow{dist}}(s_j)$ for $j \in \{1, 6\}$ and $i \in \{2, 3, 7, 8\}$.

A probable usecase of a view is to find out how a certain state is or was reached. For this we define a view that group states that have the same distance from a given state, when only traversing transitions backwards.

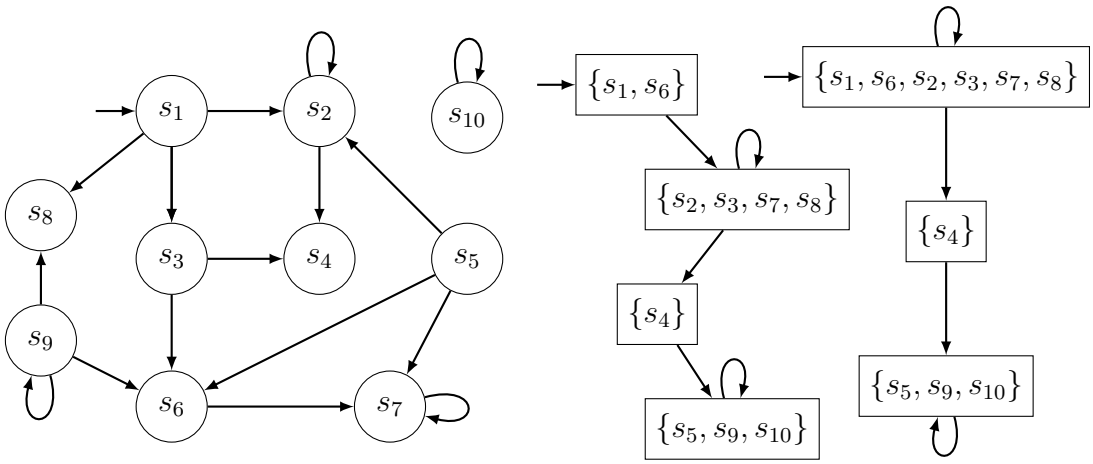

Figure 17: Simplified representations of $\mathcal{M}$ (left), the views $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 1)$ (middle) and $\mathcal{M}_{\overrightarrow{dist}}(\{s_1, s_6\}, 2)$ on it (right). CANNOT SHIFT RIGHT DIAGRAM FURTHER TO THE RIGHT

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

Note that the only difference of $\mathcal{M}_{\overleftarrow{dist}}$ to $\mathcal{M}_{\overrightarrow{dist}}$ is that $\overleftarrow{dist}$ is used instead of $\overrightarrow{dist}$, with the only difference of $\overrightarrow{dist}$ and $\overleftarrow{dist}$ being that $first(P)$ and $last(P)$ are swapped. Hence, 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 $\overline{dist}$.

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 $M = \mathbb{N} \cup \{\infty\} \cup \{\bullet\}$.

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

4.2.2 Cycles

A cycle is a structure that can exist in every graph. The concept of cycles is not specific to MDPs or any of its more specialized variants. The purpose of this thesis

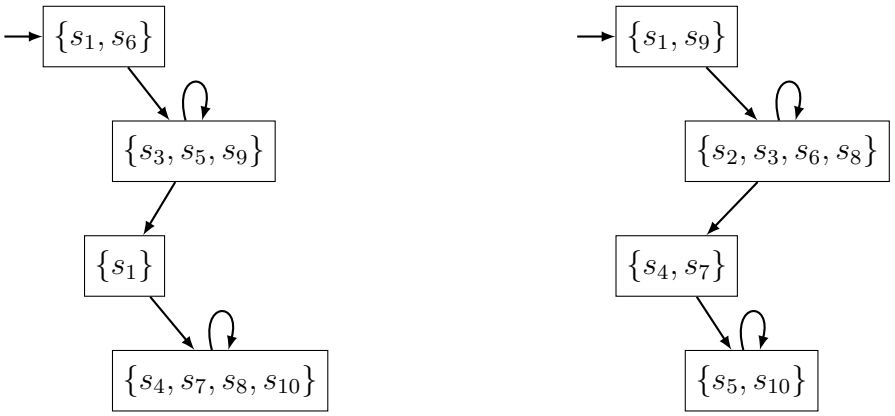

Figure 18: Simplified representations of the views $\mathcal{M}_{\overleftarrow{dist}}(\{s_1, s_6\}, 1)$ (middle) and $\mathcal{M}_{\overleftarrow{dist}}(\{s_1, s_9\}, 1)$ (right) on the $\mathcal{M}$ from Figure 17.

is to discuss views that utilize domain specific knowledge or if a general concept is of special relevance when exploring an MDP. The former and the latter apply on cycles.

Formalizing views based on cycles requires some formalization of the concept cycle. We will use a domain specific definition that will serve us the most.

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 $(s_i, \alpha_i, s_{i+1 \bmod n}) \in \mathbf{P}$.

When the actions in the cycle are of no further importance, we will omit them only writing a sequence of states. In the following let $C = (s_0, \alpha_0, s_1, \alpha_1, \ldots, \alpha_{n-1}, s_0)$ be a cycle. For conveniences 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. Let C_1 and C_2 be cycles. $C_1 \cup C_2 := \{s \in S \mid s \in C_1 \text{ or } s \in C_2\}$.

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.

In practice there exist several cycle finding algorithms. The function $findCycles(\mathcal{M}, n)$ is an abstraction for one of these algorithms being used. The actual implementation relys on algorithms of the Java library jGraphtT [21] namely the Algorithm Szwarcfiter and Lauer - $O(V + EC)$ and Tiernan - $O(V.constV)$ CITATION!!

With the formalization done we will introduce some views utilizing the concept cycles. For one we will combine the notion cycle with domain specific knowledge for the other we will consider how and why cycles in MDPs are in general of relevance. We will begin with the latter. Cycles in general are of interest because in unintended

cycles in many cases may lead to infinite values for model checking results. While it depends on the actual model in many cases infinite values in model checking are unwanted and caused to faulty construction of the model. Finding cycles hence may help in debug an MDP. Therefore a view that simply finds existing cycles is feasible.

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 cycle with a length of at least n. It is to note that the affinity of a state to one or several respective cycles is lost. For s_1, s_2 they are grouped $F_{\exists C}(s_1) = F_{\exists C}(s_2) \neq \bullet$. Hence, the obtained equivalence classes 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

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

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.

Two states s_1, s_2 are grouped if $F_{\{C_n\}}(s_1) = F_{\{C_n\}}(s_2) \neq \bullet$. Hence, the obtained equivalence classes are

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

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

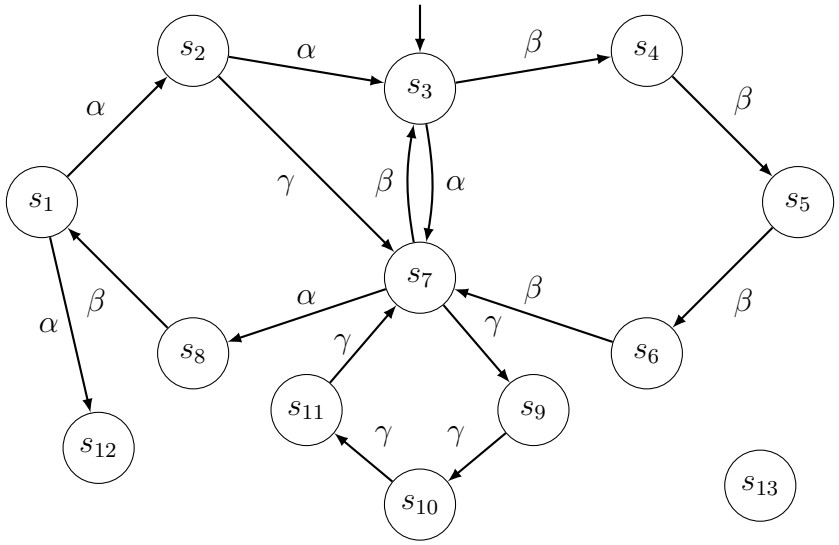

Figure 19: Simplified representation of $\mathcal{M}$

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:

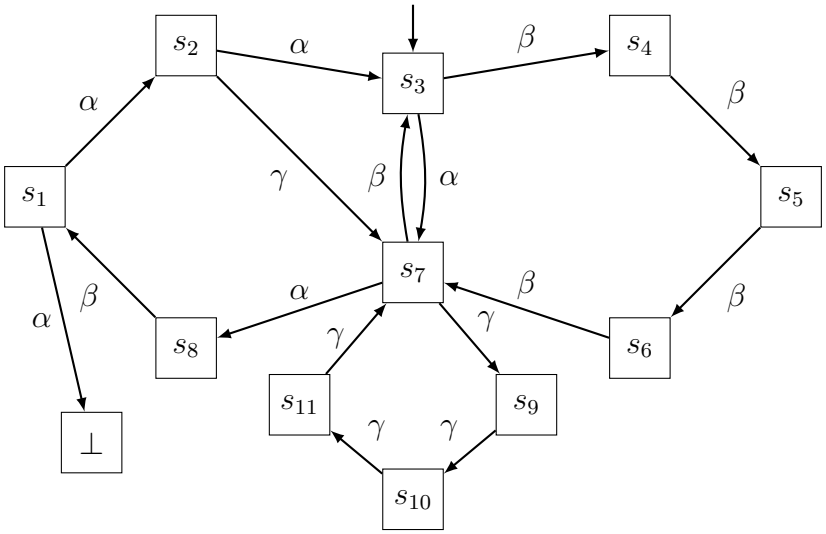

Figure 20: Simplified representations of the view $\mathcal{M}_{\exists C}$ on $\mathcal{M}$ from Figure 19. The states s_{12} and s_{13} are grouped since they are the only states not on a cycle.

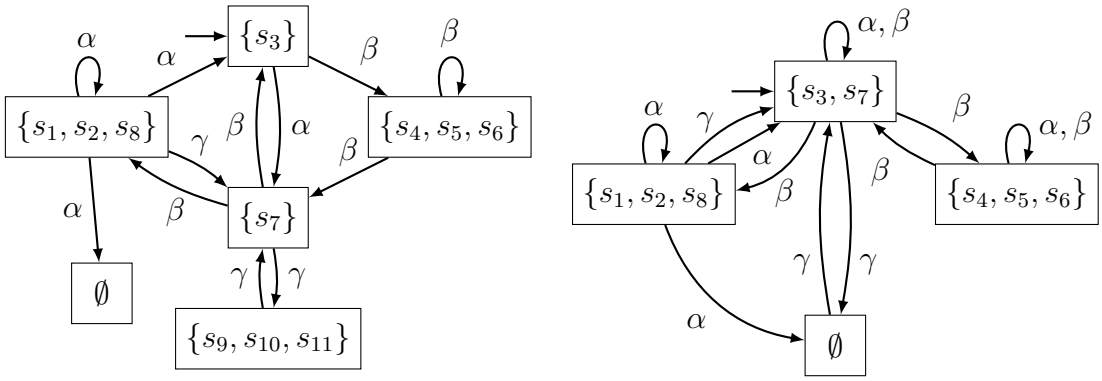

Figure 21: View $\mathcal{M}_{\{C_n\}}(2)$ (left) and view $\mathcal{M}_{\{C_n\}}(5)$ (right) on $\mathcal{M}$ from Figure 19

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

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.

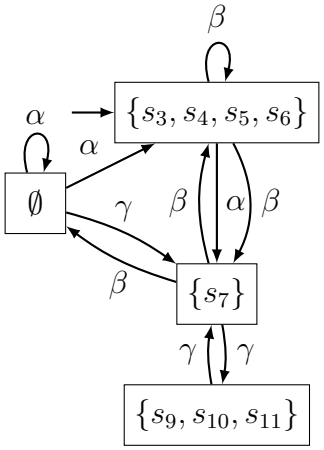

Figure 22: Simplified representation of the view $\mathcal{M}_{\cup\{s\}_C}$ on $\mathcal{M}$ from Figure 19

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

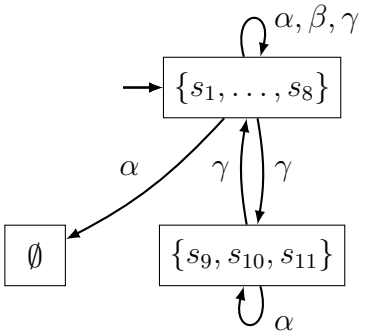

Figure 23: Simplified representation of the view $\mathcal{M}_{\cup\{s\}_C}(2)$ on $\mathcal{M}$ from Figure 19

In Figure 23 the simplified representation of $\mathcal{M}_{\{C_{\alpha,n}\}}(2)$ 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. It is $s_1, \ldots, s_8 \in C_1$. Hence, for $i \in \{1, \ldots, 8\}$ it is $F_{\cup\{s\}_C}(s_i) \subseteq \{s_1, \ldots, s_8\}$. 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 $F_{\cup\{s\}_C}(s_7) = \{s_1, \ldots, s_{11}\}$ because of $s_7 \in C_4$ 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.

Each state is mapped to the set of states resulting from joining the state sets of all the cycles the state is on.

4.2.3 Strongly Connected Components

Strongly connect components (SCC) are are interesting when exploring an MDP because they group states where may happen some complex operation. Morover 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 [22]. In the implementation an improved variant from Gabow [23] 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:

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

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

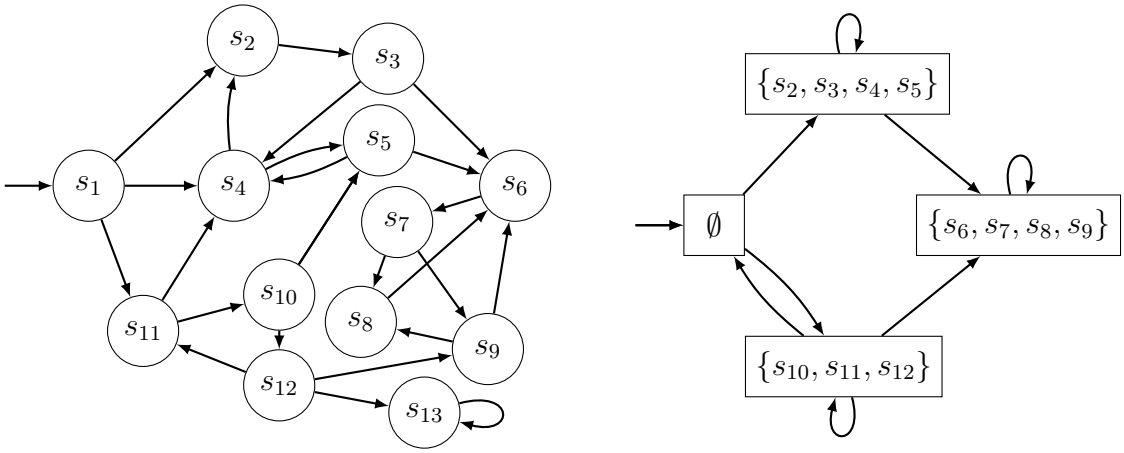

Figure 24: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{scc}(2)$ on it (right)

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.

A special kind of strongly connected components is the the bottom strongly connected component.

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. That is, 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.

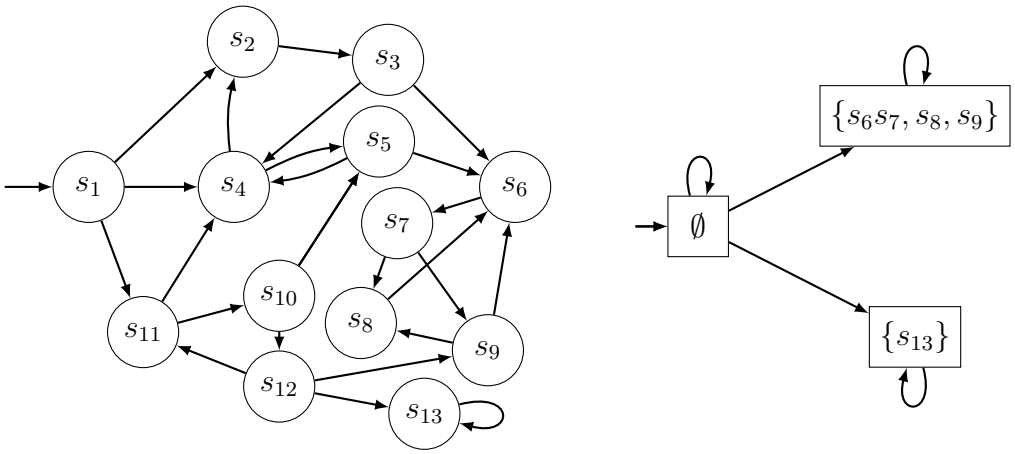

Figure 25: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{bscc}$ (0) on it (right)

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 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. When bottom strongly connected components are to be determined the found set of strongly connected components is then filtered, retaining those, that meet the requirement of being a BSCC.

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 shows an overview of the structure of the project.

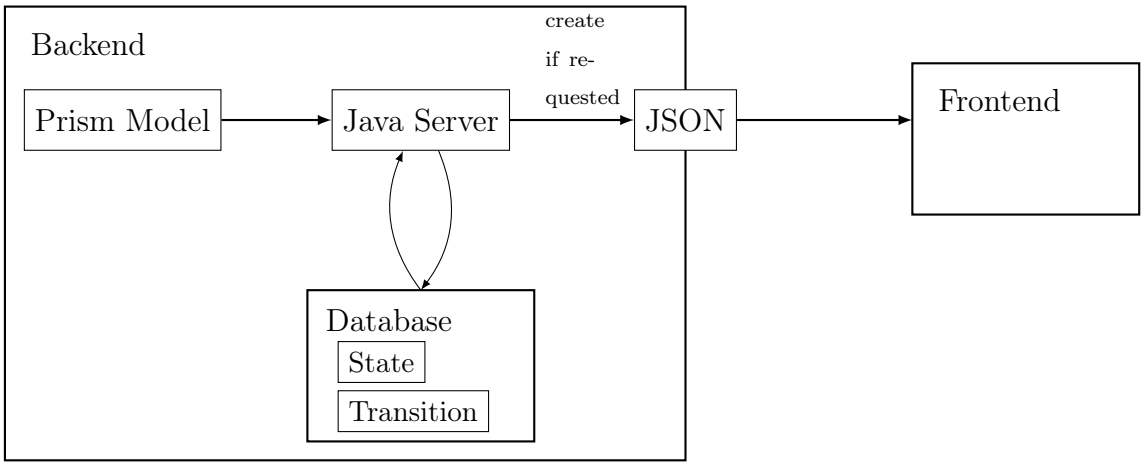

Figure 26: Project Structure, showing interaction and internal structure of frontend and backend

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 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 that corresponds to Z from Definition **??**. 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.

The concept of disregarding views is implemented a little less general as in the

```
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

formalization. In this context, the variable "semigrouping" plays a crucial role, indicating whether something should be grouped or not. For categorizing views disregarding 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:3030` 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 [21], which not only provides graph structures but also offers many common graph algorithms. 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.

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

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 take consider performance aspects. In the usecases we will see screenshots made in the to this time current version PMC-Vis. The blue printed labels of the states 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__` = •. If parallel composition is used in some form the values of the grouping functions are separated with `|` and are in the same order as they have been composed with the operator $||$.

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 add Screenshot appendix.

Already for only three philosophers the MDP has 956 states. When looking at the graphical representation from PMC-Vis 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. After applying further views could be $\mathcal{M}_{Var:a}$ (`p1`) applied to understand or explore the module.

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 where $(F_{Var:a}(s_1 \,|\, \texttt{p1}), F_{Var:a}(s_1 \,|\, \texttt{p2})) = (F_{Var:a}(s_2 \,|\, \texttt{p1}), F_{Var:a}(s_2 \,|\, \texttt{p2}))$. 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 arbitrary operations on parameters.

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

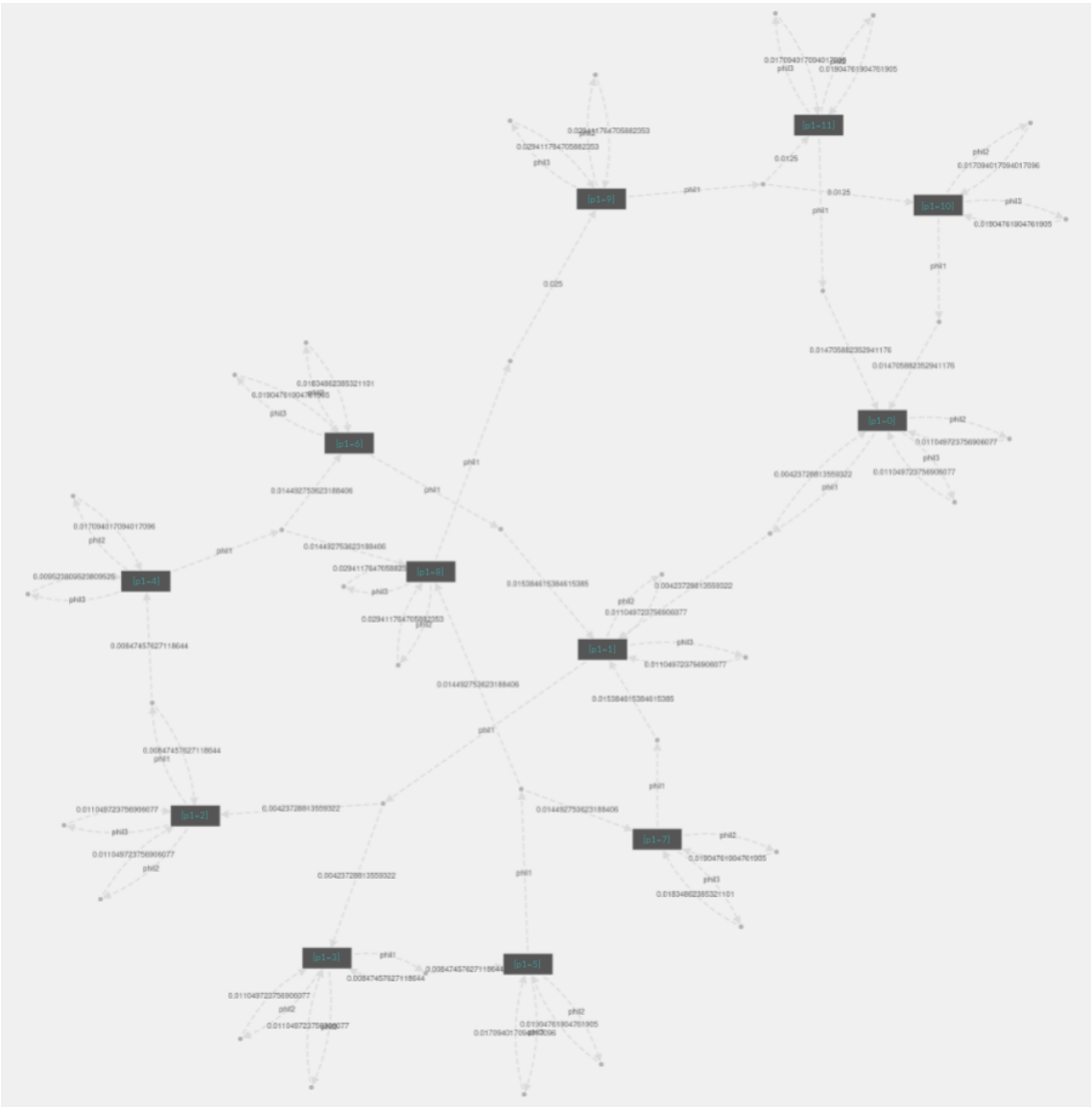

Figure 28: Screenshot in PMC-Vis of view $\mathcal{M}_{Var:a}(\texttt{p1})$

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 the following small MDP the 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 any disregarding view.

We observe, that this state is reachable, because it is shown and only reachable states are shown at all in PMC-Vis. The questions occurs how and why. In order to obtain 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)

An even better approach is to look in the model file (Figure 4)

Persons with experience in working with prism models, might quickly spot the issue, especially because this is a rather small MDP. With less experienced people and especially with larger and more complicated models, finding an issue becomes much more difficult. Hence, let us see how views can help us.

Firstly we will use the view $\mathcal{M}_{\overrightarrow{dist}}$ from that state on (Figure 31 (left)). Because in the current version of the project expansion of grouped states to the ones they contain on a visual level, has not been implemented yet, we will use a custom view that emulates this feature.

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 that state. The MDP-Graph then looks as in 32. It is to see that the state only contains a single state namely with the id seven. In the current version of implementation it is not possible, to obtain the parameter values. With the database file we obtain that this is the state where $x = 2$ and $y = 1$.

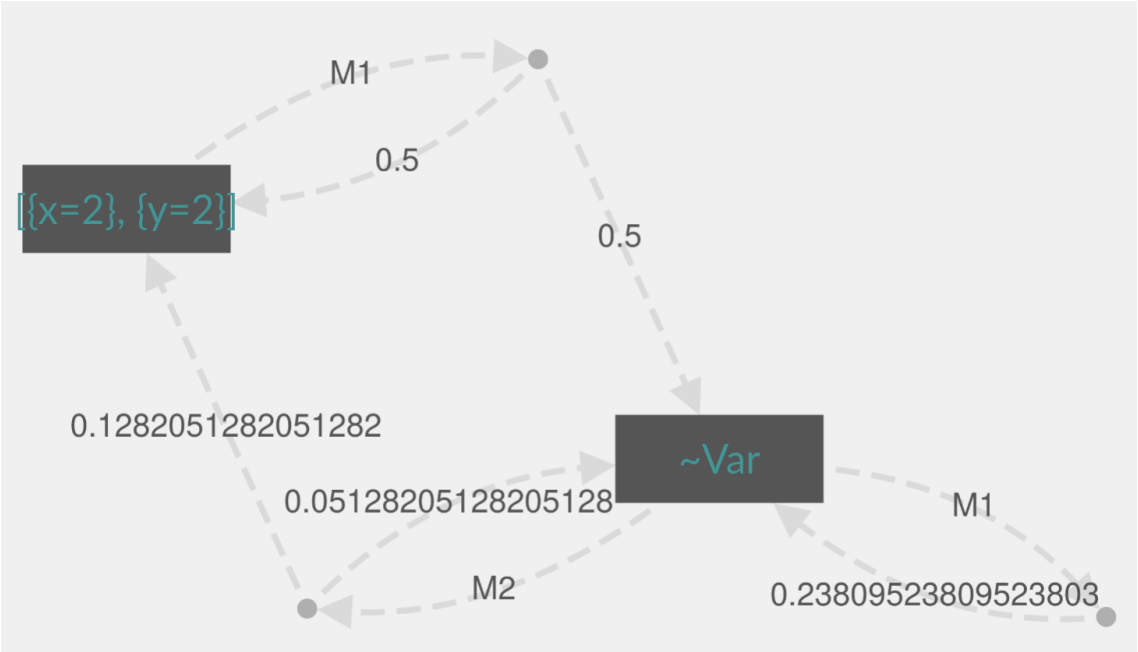

Figure 30: Screenshot in PMC-Vis of view $\mathcal{M}_{VarCNF}(c(s))$ where `~Var` refers to $\bot$.

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

```
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

Because $x = 2$ represents the system 1 one sending and system 2 waiting, we see that is possible for the second system to begin to send on the medium although it is already occupied by the first system!

When now looking at the prism file (Listing 4) we can see why this is the case. In last line of the second module when $y = 1$ there is a 50 percent chance to enter $y = 2$. This line originally was intended for termination of the transmission. From $y = 1$ it should not be possible to enter $y = 2$ if $x = 2$. After fixing this, the state no longer appears after the application of ${\mathcal{M}_{VarDNF}}^{\top}(c(s))$.

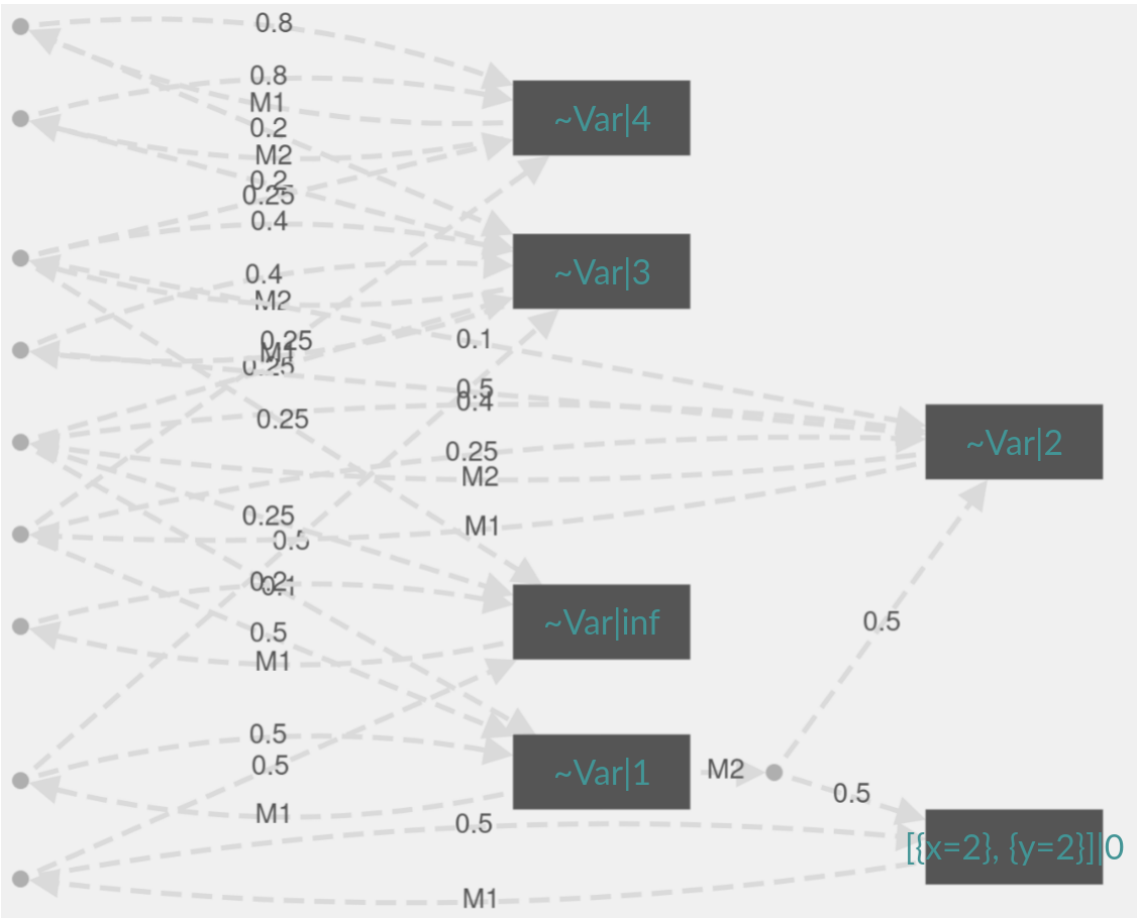

Figure 31: Screenshot in PMC-Vis of view $\mathcal{M}_{VarCNF}(c(s)) \,||\, \mathcal{M}_{\overrightarrow{dist}}$ where `~Var` refers to $\perp$.

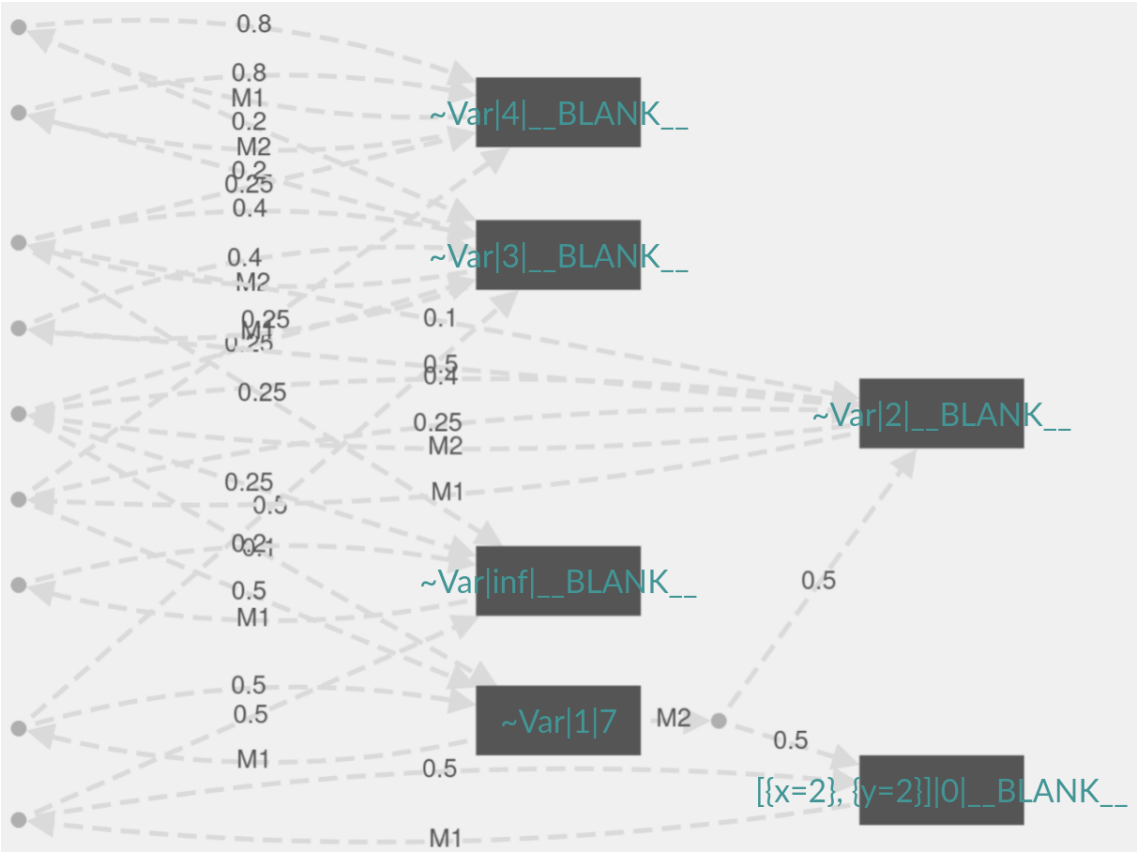

Figure 32: Screenshot in PMC-Vis of view $\mathcal{M}_{VarCNF}(c(s)) \,||\, \mathcal{M}_{\overrightarrow{dist}} \,||_Z \mathcal{M}_{id}$ where $Z = \{(F_{\overrightarrow{dist}}, 1)\}$ and `~Var` refers to $\perp$. Ugly Linebreak

6.3 Understand and Debug MDP

In this subsection we want to take a look at a more complex usecase how views might help us to understand and fix a given MDP. We refer to the MDP with as described by the prism file in Listing 5.

Firstly, we will gather some understanding of the model. We already saw in section 6.1 that variables can help a lot with understanding an MDP. When applying $\mathcal{M}_{Var:a}(\texttt{time})$ we see that the MDP has a limited timed behavior (Figure 33 left). When applying $\mathcal{M}_{Var:a}(\texttt{phases})$ we observe that the system is operating in phases (Figure 33 right). If we consider $\mathcal{M}_{Var:a}(\texttt{time}) || \mathcal{M}_{Var:a}(\texttt{phases})$ we see that the phases are repeated for each iteration of time (Figure 34 right). However, we still don not have any 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 obtain that we have sets of states which 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 name of the actions we see that this system seems to have a configuration phase a working phase a reconfiguring phase and an end phase. The grouping appears to be very similar to $\mathcal{M}_{Var:a}$ (`phases`). Hence, we consider $\mathcal{M}_{\overrightarrow{Act(s)}_{\approx}} || \mathcal{M}_{Var:a}(\texttt{phases})$ (Figure 36). Indeed the enumeration coincides with the grouping of outgoing actions, with the exception of `[end]` and `[reconfigure]` timed behavior?

Hence we know that the system is working for $\texttt{phase} = 0$, configuring for $\texttt{phase} = 1$ and termination configuration in $\texttt{phase} = 2$. This process is repeated six times until $\texttt{time} = 5$.

In general we learned that this system runs several times with choosing certain configurations each time before it runs. Its reward function rewards states that work with a better configuration. A classic model checking value is to determine "MaxUtility":R"Utility"max=?[F "end"]and "MinUtility":R"Utility"min=?[F "end"] . For "MaxUtility":R"Utility"max=?[F "end"]we obtain x, for "MinUtility":R"Utility"min=?[F "end"] inf. Since the system is modeled for a finite time and each time chooses from a finite set of configurations, it is unwanted behavior, that "MinUtility":R"Utility"min=?[F "end"] is infinite. We will now show how views can help to find the cause.

Such behavior of infinite "MinUtility":R"Utility"min=?[F "end"] is often caused by cycles. A feasible idea would be to use a view with cycles. As it will be discussed in Chapter Performance these views very ressource intensive when there are larger strongly connected components. Moreover when finding strongly connected components the cycles are equally found since each cycle is a strongly connected component. The view $\mathcal{M}_{scc}$ yields that there are quite large strongly connected components (Figure 37). To find out where these are locate we consider the parallely 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 configuring phase. When taking a look at the prism file, we see that we can arbitrarily often switch the configuration. Hence, it is not assured that a configuration can only be selected once. This causes infinite paths in the MDP, which in consequence cause infinite maximal expecta-

```
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

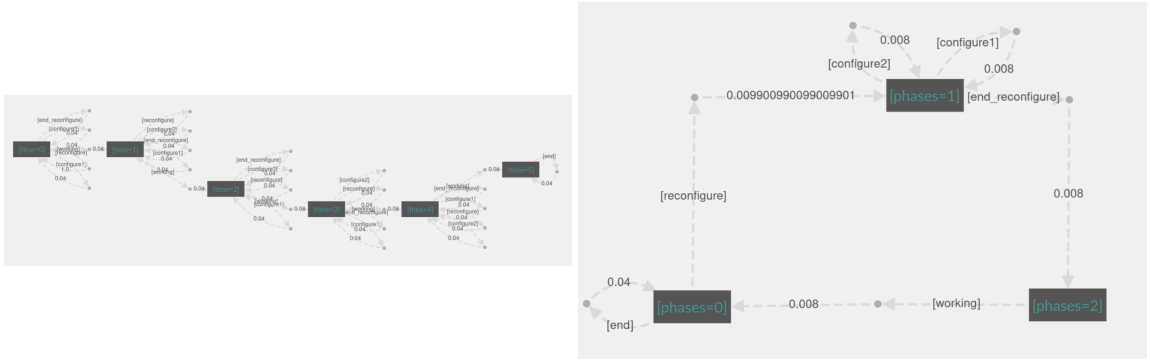

Figure 33: Screenshot in PMC-Vis of view $\mathcal{M}_{Var:a}(\texttt{time})$ (left) and view $\mathcal{M}_{Var:a}(\texttt{phases})$ (right)

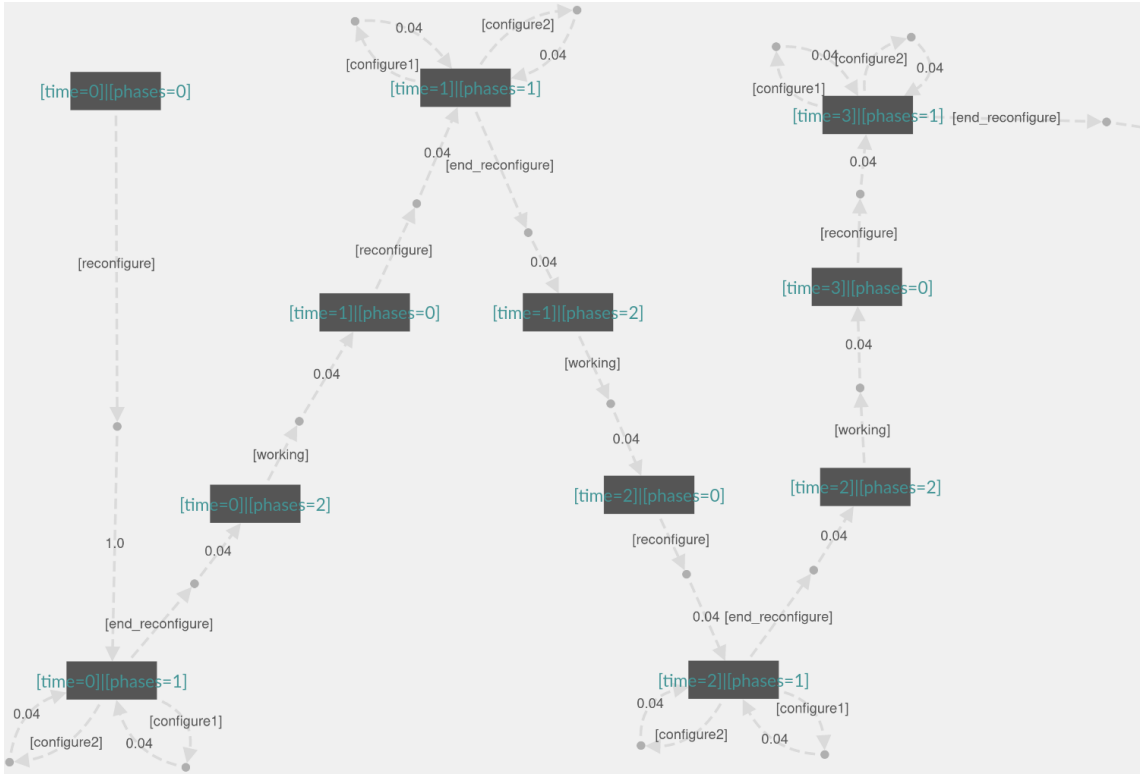

Figure 34: Screenshot in PMC-Vis of view $\mathcal{M}_{Var:a}(\texttt{time}) \,||\, \mathcal{M}_{Var:a}(\texttt{phases})$

tions. This can be fixed by assuring that a configuration can only be selected once. An easy way of accomplishing this is to sequentialize the selection of the two configurations: Firstly `[configuration1]` is selected, afterwards `[configuration2]` and finally the configuration phase ends (`[end_configuration]` is the only available action left to take) (Figure 6).

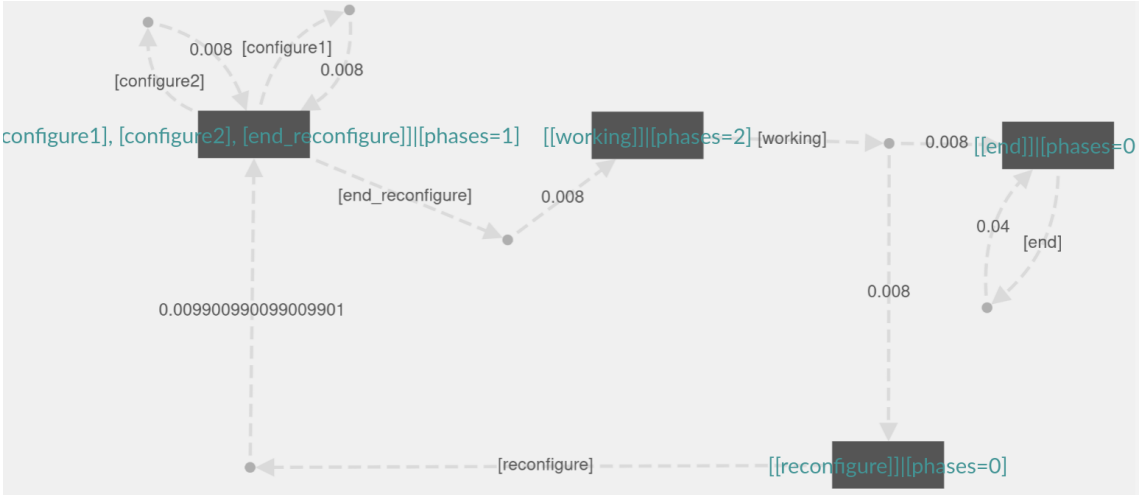

Figure 35: Screenshot in PMC-Vis of view $\mathcal{M}_{\overrightarrow{Act}(s)_{\approx}}$

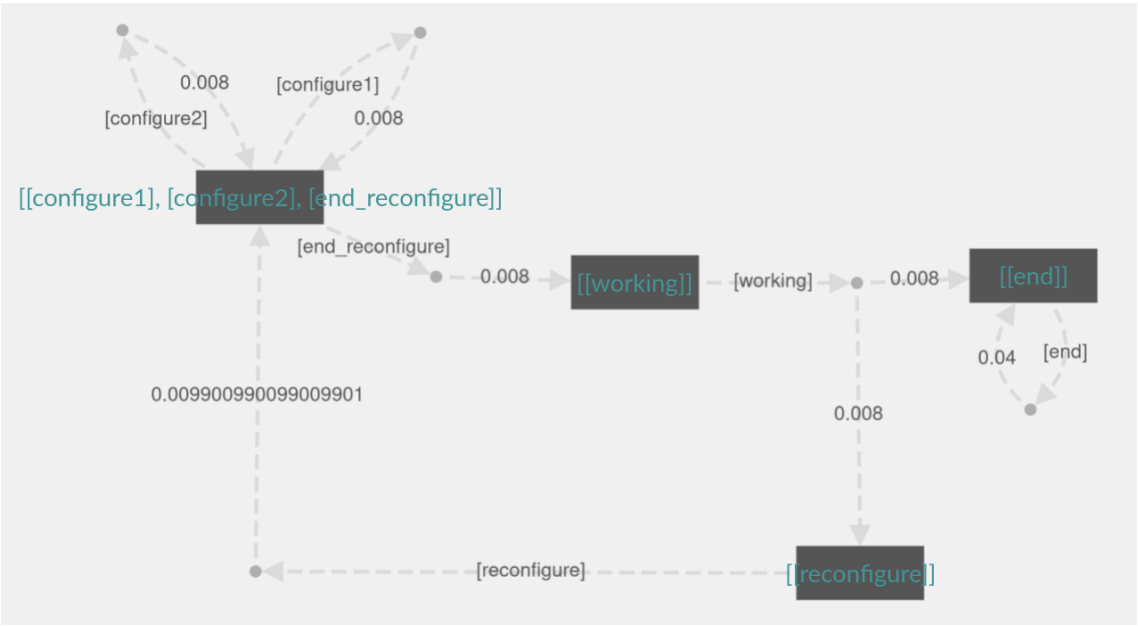

Figure 36: Screenshot in PMC-Vis of view $\mathcal{M}_{Var:a}(\texttt{time}) \,||\, \mathcal{M}_{Var:a}(\texttt{phases})$

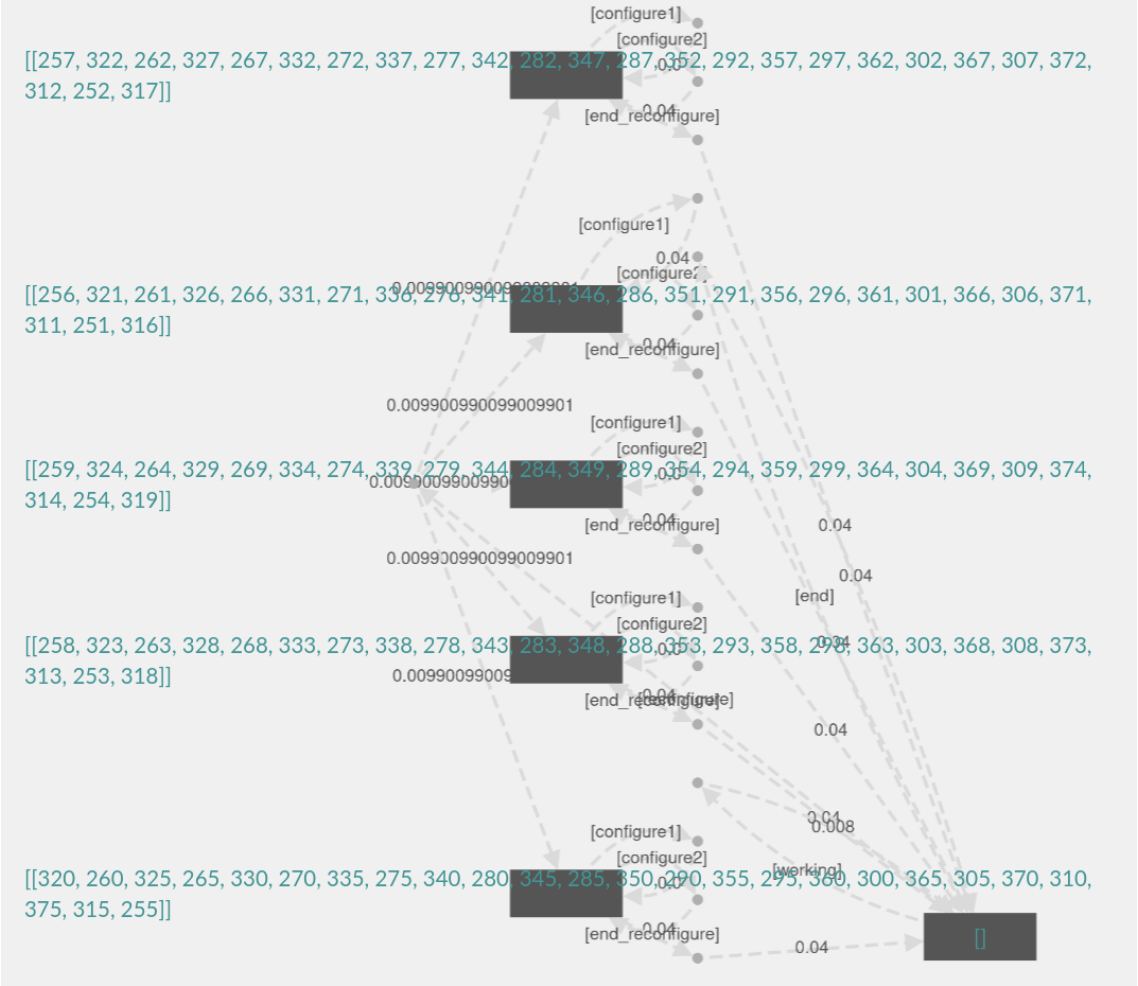

Figure 37: Screenshot in PMC-Vis of view $\mathcal{M}_{scc}$

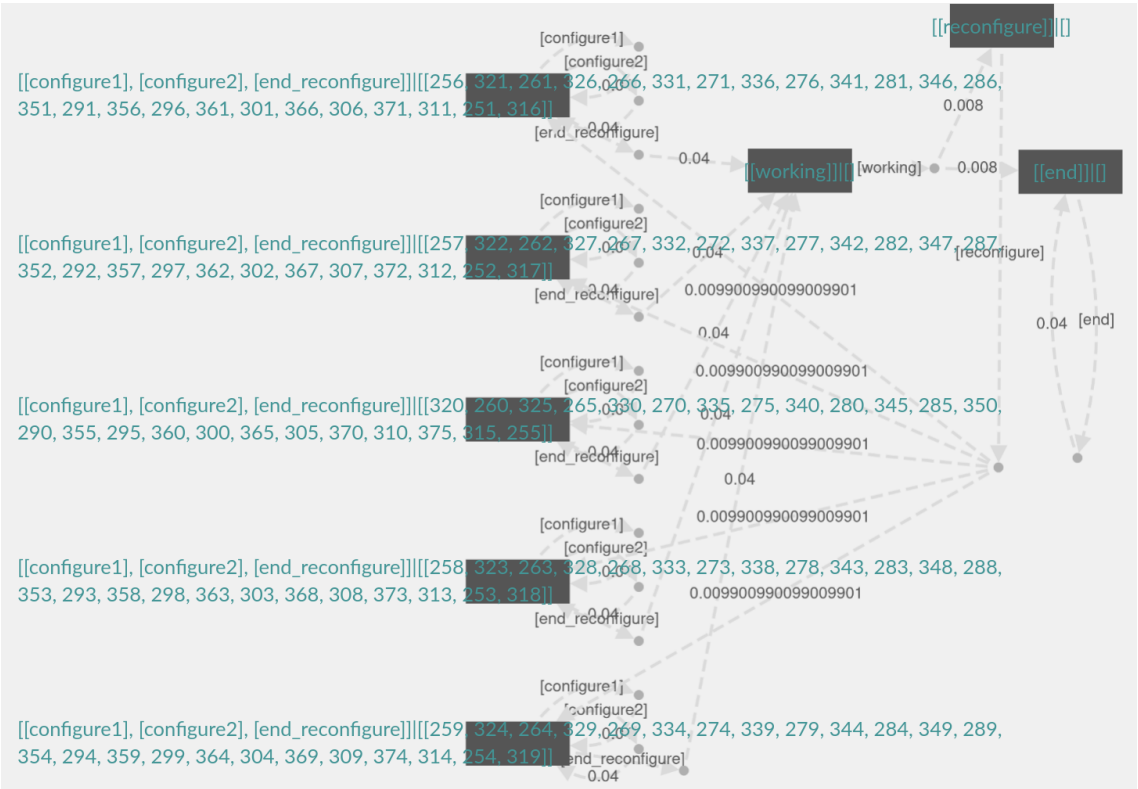

Figure 38: Screenshot in PMC-Vis of view $\mathcal{M}_{scc} \,||\, \mathcal{M}_{\overrightarrow{Act(s)}_{\approx}}$

```
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 shall 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 internal graph structure (`mdpGraph`) that is based on the jGraphtT library, with respect to its built time and memory occupation. The tests have been performed on a Dell XPS 9370 (16 GB RAM, Intel Core i7-8550U) with Manjaro KDE. Only neccesary tools where opened when the test were running: Firefox and IntelliJ.

Time was measured with self written class `Timer` which utilizes the package `java.time.LocalTime`. Class implementation and sample usecase are provided in the appendix to be added in the appendix. Memory occupation has been measured with with the tool Java Object Layout (JOL) provided by openJDK. The method used was `totalSize()` in `org.openjdk.jol.info.GraphLayout`. For the benchmark one single scalable MDP is used (prism model file in appendix). Sizes considered are 50, 100, 500,1 000, 5 000, 10 000, 50 000, 100 000, 500 000 and 1 000 000. Using one single scalable MDP means that execution times and memory occupied might vary with differing graph structure of the MDP. This is especially the case for views based on the graph structure. Hence in this section we will give more of an overview of the expected values for creation times and memory occupation rather than a detailed analysis.

Firstly we will consider the time required to create views. All views are created 100 times for each considered size of the MDP. Their execution times for each sizes then are averaged. As we know from Chapter 5 creating a view involves to instantiate the object and execute the build function. The instantiation only involves setting a hand full of private attributes. Since the amount of attributes of a view is fix and their size does not cohere with the size of the MDP, the is instantiation time is negligible. Building `mdpGraph` is also part of instantiating a view in case it has not yet been built on the model, but will be considered separately. Hence, we will focus on the time occupied by the `buildView()` method. As we know from section 5 the build `buildView()` method splits into four parts: 1) checking prerequisites ("prechecks"), 2) creating a new column, 3) executing grouping function 4) write results to database. Firstly we will consider the computation time for the grouping function as the time actions performed in the steps 1) 2) and 4) are almost identical or identical.

In Figure 39 we see the averaged execution times (100 executions) of `groupingFunction` for the different views. It is to see that the execution times overall are very similar and behave linear to the amount 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. Variation in the magnitude of microseconds with very small MDPs are to be expected. In general the performance results as shown indicates very good scalability with respect of views being usable on large models. This very good performance is to large portion attributed to the graphstructure 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 apart from prechecks, the execution time

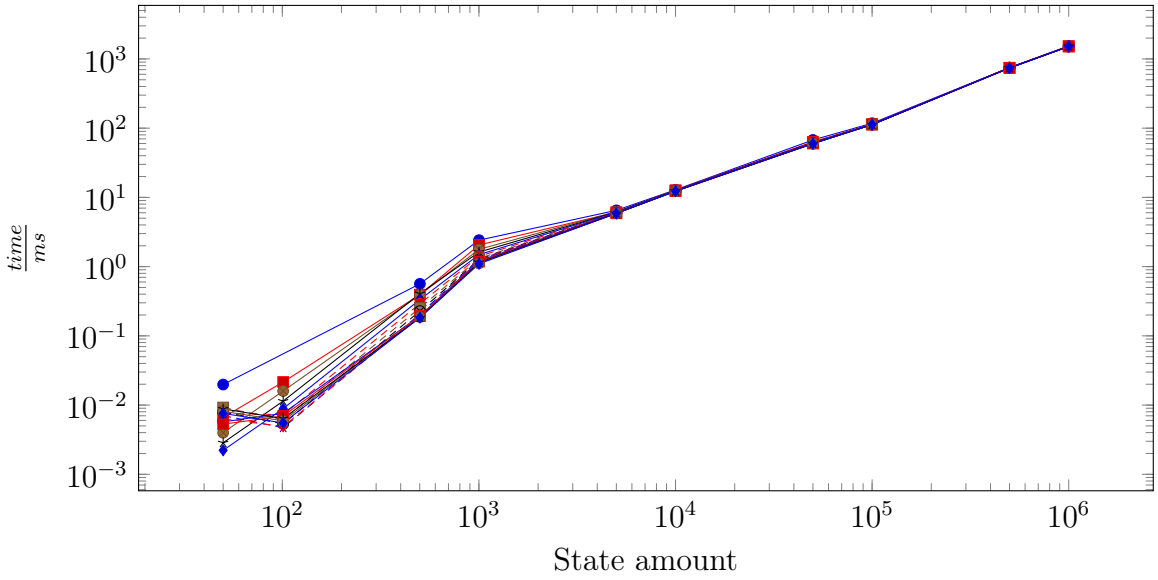

Figure 39: Average grouping function computation times

of the grouping function is the least time consuming operation. Most time is taken by writing the results to the database, where even building the `mdpGraph` is faster. Times for writing to the database, refer to the at this time present implementation of the database of PMC-Vis which might be subject to changes in the future.

As stated before the performance of computing grouping functions that quickly heavily rely on the implemented graph structure (`mdpGraph`). This graph structure is held in memory and become quite large for MDPs with about one million states. Figure 41 displays the measured deep size (including referenced objects) of the `mdpGraph`. As one would expect 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 are occupied for the graph alone if, 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 utilize the `mdpGraph` even for large MDPs. Still it is to be kept in mind that this only is the size of the built graph object. When reading from the database in order to build the `mdpGraph` more memory is needed as objects containing the information from the database also temporarily stay in memory. PMC-Vis itself will also require memory as other possibly unrelated process that run 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, while at the same time also close to the memory limitation of the system.

In general we conclude that computation of grouping functions is rather quick and scalable. Building times of the `mdpGraph` are reasonable, but entails heavy memory utilization. Database accesses are the most time consuming fraction of the time needed to create a view.

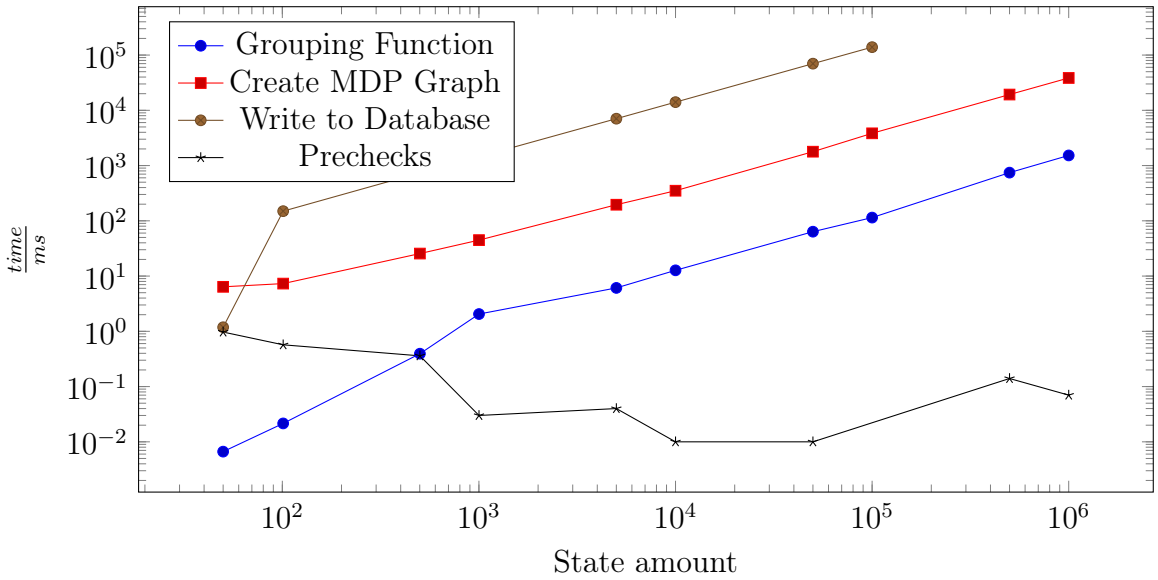

Figure 40: Average times for grouping function computation (1 representative from Figure 39), writing to the database and building the `mdpGraph`

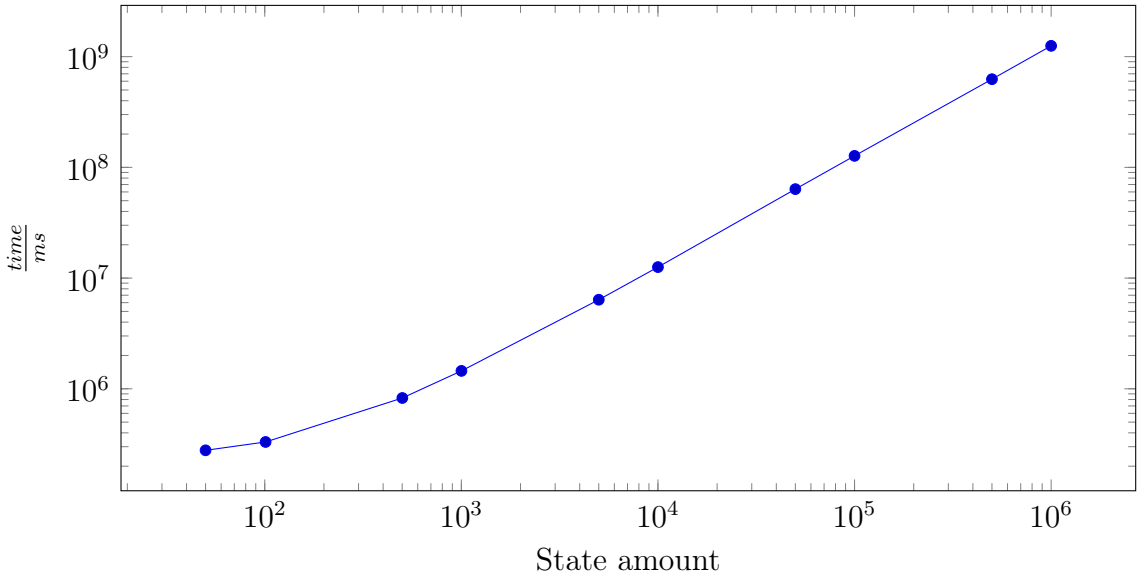

Figure 41: Size of `mdpGraph` for different model sizes

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 have been presented, that show 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 project PMC-Vis. The implementation relies on a fast but memory hungry graph structure provided by the jGraphtT library.

In conclusion it can be said that the formalization concisely describes the notion of the concept view, which assists in understanding the implementation and might be of use to develop other simplification approaches on model checking related systems. For proposed example views it is to note that the author had little experience in working with MDPs. Hence, most views are not motivated by the way in which a view might serve when working with an MDP but rather what simplifications on MDPs are possible taking to account its components and structure. The use cases shown only pose small selection of what might be possible. The true practicability will be proven when used in practice. Also MDP-experienced users actually working with views in practice will most likely yield further ideas of views that may be helpful.

Apart from practice driven advancements for views and their use cases, also from a theoretical perspective there is still room for exploration of concepts. For one it has not been taken advantage of probabilities present in MDPs, as considering views with probabilities would have exceeded the scope of a bachelor thesis. Furthermore, there might be advanced concepts from other disciplines that can be exploited 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 testing machine due to lack of memory. Thereby for a given amount of available memory, the data structure poses a hard limitation to the applicability of views in practice if the MDP exceeds a certain size. This issue can be addressed by not utilizing the datastrucuture (`mdpGraph`) but directly accessing the database. This enables views to be used on almost arbitrary large MDPs. While this is a great advantage its downside is an immense drop in performance (rise in built times), compared to the used `mdpGraph`, as the database accesses with the current version of PMC-Vis are magnitudes slower than accessing the information in the `mdpGraph`. This is due to the fact that the `mdpGraph` consists of very efficient datastructures such as hash maps and hash sets that lie in memory whereas accessing the database, entails accessing the storage of the machine, which in principle is much slower than accessing memory. Especially if statewise database accesses are to be performed, this causes immense execution times in creating a view.

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. Copyright - © 2016. This work is licensed under http://creativecommons.org/licenses/by/4.0/ (the "License"). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License; Zuletzt aktualisiert - 2020-01-29.

[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] 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.

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

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