Technische Universität Dresden • Fakultät Informatik

Data Preperation for PMC-Visualization

Bachelorarbeit zur Erlangung des ersten Hochschulgrades

Bachelor of Science (B.Sc.)

vorgelegt von

Franz Martin SCHMIDT

(geboren am 7. April 1999 in Halle (Saale))

Tag der Einreichung: August 6, 2023

Dipl. Inf Max Korn (Theoretische Informatik)

Contents

Abstract

Lorem ipsum

1 Introduction

PMC is short for Probabilistic Model Checking.

Model Checking is... Probabilistic means...

When analyzing PMCs a result table is generated which consists of result vectors. These result vectors for example contain probability values for temporal events, expected values of random variables and other qualitative and quantitative results. For central measure so called schedulers can be used to resolve non-determinism in the MDPs which in addition to the result provide one ore more optimal actions per state.

Already simple system models can lead to complex system behavior and an immense amount of states. An interactive visualization of an MDP can be used in support of understanding, analyzing and reviewing these complex systems. It can even be made use of to achieve further goals such as debugging or model repair. However, the pure data volume remains unaffected. The complexity is shifted to the visualization which has some general techniques to represent large amounts of data in a suitable way.

WHAT EXACTLY CAN BE DONE/HAS BEEN DONE —¿ INSERT HERE

Apart from the methods available in visualization there exist domain specific abstraction methods for transition systems. These are based on relations such as simulation order, simulation equivalence, bisimulation equivalence as well as trace equivalence, stutter linear time relations and stutter bisimulation equivalences. These relations can be based on the set of all possible transition systems or the set of states of one single transition system. Whereas the first declares some similarity of transition systems with regard to model check them the latter one can be used to obtain quotient systems. The set of states of such quotient system coincides with the the respective equivalence relation.

- all relations declare that at least one of the two systems being in relation can mimic the behavior of the other one in some way. In case of equivalence the systems can mimic each other.
- except for trace inclusion and trace equivalence whether such mimicking is possible depends on the existence of a specific relation $\mathcal{R} \subseteq S_1 \times S_2$ where S_1 and S_2 are the state sets of two corresponding transition systems TS_1 and TS_2. The symbol $\mathcal{R}$ is included in each definition of a relation on the set of all possible transition systems. In each definition it receives a different name and represents a different relation between the sets S_1 and S_2.
- Depending on the type of mimicking (simulation order, bisimulation equivalence, ...) different requirements are made on $\mathcal{R}$. It is dependent on these requirements whether $\mathcal{R}$ exists for a given pair of transition systems TS_1 and TS_2 and hence also whether or not these two transition systems are in relation.
- all relations have three requirements made on $\mathcal{R}$ in common.
- firstly, there has to be some relationship between the initial states. That is, at least $\exists(s_1, s_2) \in \mathcal{R}$ with $s_1 \in I_1, s_2 \in I_2$ where I_1 and I_2 are the respective sets of initial states of the two transition systems TS_1 and TS_2 in question

- Secondly for each $(s_1, s_2) \in \mathcal{R}$ the two states must have the same set of atomic propositions. That is $L(s_1) = L(s_2)$.
- lastly for all s_1, s_2 with $(s_1, s_2) \in \mathcal{R}$ some requirement is made about their (not necessarily direct) successors
- the mimicking types differ in the requirements they make on the relation $\mathcal{R}$ with respect to initial states (1) of the two transition systems and the requirements made on the successors (2).
- when one transition system can mimic another it is only required that that each initial state $s_1 \in I_1$ there is an initial state $s_2 \in I_2$ with $(s_1, s_2) \in \mathcal{R}$
- when two systems mimic each other it is as well required that for each initial state $s_1 \in I_1$ it is $(s_1, s_2) \in \mathcal{R}$ with $s_2 \in I_2$ but additionally that for each $s_2 \in I_2$ it is $(s_2, s_1) \in \mathcal{R}$ with $s_1 \in I_1$.
- where all of them differ is the requirements made on the successors of the states being in relation
- bisimulation equivalence ($\sim$) declares that two transition systems can mimic each other. Every two states with $(s_1, s_2) \in \mathcal{R}$ one must have an successor s' for each successor $\tilde{s}'$of the other state so that the two successors stand in relation with respect to $\mathcal{R}$. That is the successors standing in relation again must have the same set of atomic propositions and meet the requirement regarding their successors. If a state s is equal in its set of atomic propositions to another state $\tilde{s}$ and meets a respective requirement on a selection of the successors of $\tilde{s}$ we say it state-mimics it. Note that here state-mimicking is mutual. s_1 state-mimics s_2 and vice versa. definition? bisimulation equivalence is, as the name suggest, an equivalence relation on the set of all possible transition systems.
- simulation order ($\preceq$) is weaker than bismulation. It only declares that one system can state-mimic the other. For every two states with $(s_1, s_2) \in \mathcal{R}$ the second one must state-mimic the first one. That is, for each successor s_1' of s_1 the state s_2 must have an successor s_2' so that s_2; $state - mimics\ s_1'$. In this case of the simulation order relation s_2' state-mimicking s_1' coincides with $(s_1', s_2') \in \mathcal{R}$. For the simulation order relation it is not required that s_1 can state-mimic s_2. Simulation order is a preorder on the set of all possible transition systems.
- Simulation equivalence ($\simeq$) declares that for two transition systems TS_1, TS_2 it holds that $TS_1 \preceq TS_2$ and $TS_2 \preceq TS_1$. It might seem like it conincides with bisimulation but it does not. For simulation order it has to hold that for a given state s_1 in the mimicked system that there has to exist another state s_2 in the mimicking system, that state-mimics s_1. Although due to simulation equivalence s_2 now also must have a state s' that mimics s_2, it does not need to hold that $s' = s_1$. The state s_2 could be mimicked by another state than s_1. The mimicking between two states does not need to be mutual. Simulation equivalence is an equivalence relation on the set of all possible transition systems.

- in general it holds that for TS_1, TS_2 being transition systems $TS_1 \sim TS_2$ implies $TS_1 \simeq TS_2$ implies $TS_1 \preceq TS_2$.
- All of the mimicking types presented, mimicking is happening in a stepwise manner and atomic propositions are concidered as the property of concern. Choosing atomic propositions has the advantage of preserving the logical formulae used in model checking.
- bisimulation equivalence preserves CTL and CTL* formulae for finite TS without terminal states 579
- simulation equivalence/order maintain LTL by trace inclusion and universally and existencially quantified fragments of CTL* correct? 579
- The condition of stepwise mimicking is softened with stutter simulation and bisimulation equivalence expands the concept of simulation equivalence and stutter bisimulation equivalence. Instead of requiring for each successor s_1' of a state s_1 to be state-mimicked by a direct successor of s_2 it suffices if there is path to such a state s_2', that state-mimics s_1'. All states on the path must then also state-mimic s_1. Moreover this requirement is only made for successors of s_1 if the successor s_1' is not in relation to s_2.
- There also is another variant of stutter bisimulation equivalence which is called stutter bisimulation divergent equivalence which we wont consider here.
- So far relations on the set of all transition systems have been considered.
- All the above mentioned equivalence relations are also defined on the set of states of one single transition system.
- They are defined very similar to the relations on the set of all possible transition systems. Let $\bullet_{TS} \subseteq S \times S$ be some relation on the set of states S of an transition system. Two states $s_1, s_2 \in S$ are in relation $((s_1, s_2) \in \bullet_{TS})$ if there exists a relation $\mathcal{R} \subseteq S \times S$ that meets some requirements. These requirements are the same as the ones used on $\mathcal{R}$ for transition systems except for the condition on the initial states. That is, only the set of atomic propositions of s_1, s_2 has to be equal $(L(s_1) = L(s_2))$ and some requirement on their successors has to be met.
- They are used for abstractions by defining quotient systems of a given transition system. In a quotient systems the set of states consists of the equivalence classes of the equivalence relation.

  Definition 1.1. Let $TS = (S, Act, \longrightarrow, I, AP, L)$ be a transition system and $\bullet_{TS}$ and equivalence relation on S. The quotient transition system is defined by $TS/\bullet_{TS} = (S/\bullet_{TS}, \{\tau\}, \longrightarrow', I', AP, L')$ where:

  - $I' := \{[s]_{\bullet_{TS}} \mid s \in I\}$
  - $\longrightarrow' := \{([s]_{\bullet_{TS}}, \tau, [s']_{\bullet_{TS}}) \mid (s, \alpha, s') \in \longrightarrow\}$

  - $L'([s]_{\bullet_{TS}}) := L(s)$

- holds $TS \bullet TS/\bullet_{TS}$
- Similarily an abstract transition system using the simulation relation can be obtained... using an abstraction function
- def abstraction function, def relation
- obtain smaller system that has properties with respect to formulas discuss how much smaller? → would rather not
- expensive: to calculate with complexities
  - bisimulation:
  - simulation
  - stutter simulation:
  - stutter bisimulation:
- usecase: browsability for humans
  → understand and looking at the system from a human perspective of highest importance
  → less expensive calculation
- views of an MDP aim for high comprehensibility, low performance and will conceptually be very similar to the the notion of an abstract transition system.
- "Partial order reduction attempts to analyze only a fragment ˆTS of the full transition system TS by ignoring several interleavings of independent actions."

In alternative to these general approaches domain specific preprocessing can be used to gain certain views on a given MDP that display areas or sets of states in a summarized more compact form. For this purpose structural properties as well as criteria obtained from the result vectors and their containing optimal actions can be utilized.

This thesis is about the development, formalization and implementation of a collection of different *views*. Views are supposed to be applicable to only a subset of a given MDP and be composable with other views The input for a view is the result table of the MDP, whereas the output is a new collumn in the MDP result table that determines which states are to be grouped to a new one. The evaluation is performed within the existing web-based prototype of a PMC visualization platform (PMC-Viz) and on the basis of different MDP models. They are to be analyzed regarding the suitability to facilitate the understanding of complex operational models incl. analysis results and to support processes of debugging, model repair or strategy synthesis.

2 Preliminaries

Views will be defined on MDPs. Instead of directly providing the definition of 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. Considering a traffic light displaying green could be considered as one state whereas displaying red could be another one. Transitions model the progression of the system from one state to another one. Sticking to the example of the traffic light a transition could model the switch from state green light being displayed to the state of red light being displayed. There are several variants of transitions systems. We will use transition systems with *action names* and *atomic propositions* for states as in BAIER. 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. They are assigned to a state by a Labeling function L.

The following definition is directly taken form Principles of Modelchecking, Baier p. 20

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. 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 of being selected. It is important to note that Markov Chains are memoryless in the sense that the system evolution is not

dependent on the history of only on the current state. That is the evolution of the system does not depend on the sequence of so far traversed states with the transitions. That the nondeterministic behavior is replaced with a probabilistic one also means that there are no actions in Markov chains.

Definition 2.2. 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 $\mathcal{M}$ 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. A initial state s_0 with $\iota_{init}(s_0) > 0$ is yielded. Afterwards in each state a transition is yielded at random according to the probability distribution $\mathbf{P}$ in that state. The evolution of a Markov chain ends in a state s if and only if $\mathbf{P}(s, s) = 1$

- has no actions
  "As compositional approaches for Markov models are outside the scope of this monograph, actions are irrelevant in this chapter and are therefore omitted."

NOTES BEGIN

A Markov chain permits both probabilistic and nondeterministic choices thereby is a model that somewhat merges the concept of transition systems with the concept of Markov chains. They add probabilistic choices to transition systems or add nondeterminism to Markov chains. Probabilistic choices may be used to model the (probabilistic) behavior of the environment. But to do so statistical experiments are required to obtain adequate distributions that model the average behavior of the environment. If this information is not available too hard, too obtain, or a guarantee about system properties is required nondeterminism is the natural option for

modeling in these cases. Further usecases of MDPs are randomized distributed algorithms and abstraction in Markov chains. Randomized distributed algorithms are nondeterministic because there is a nondeterministic choice which process performs the next step and randomized because they have rather restricted set of actions that have a random nature. Abstraction in Markov chains can be feasible if states have been grouped by AP and and have a wide range of transition probabilities. If that is the case selection of a transition is rather non deterministic and thereby can be abstracted with an MDP by replacing the probabilities with nondeterminism.

Definition 2.3. A *Markov decision process* 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 $Act(s) = 1$ 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 ($\iota_{init}(s) > 0$).

not sure if I should: $I := \iota_{init}$ thereby meaning the underlying set
CONVENTION END TEST Handshaking and Prism

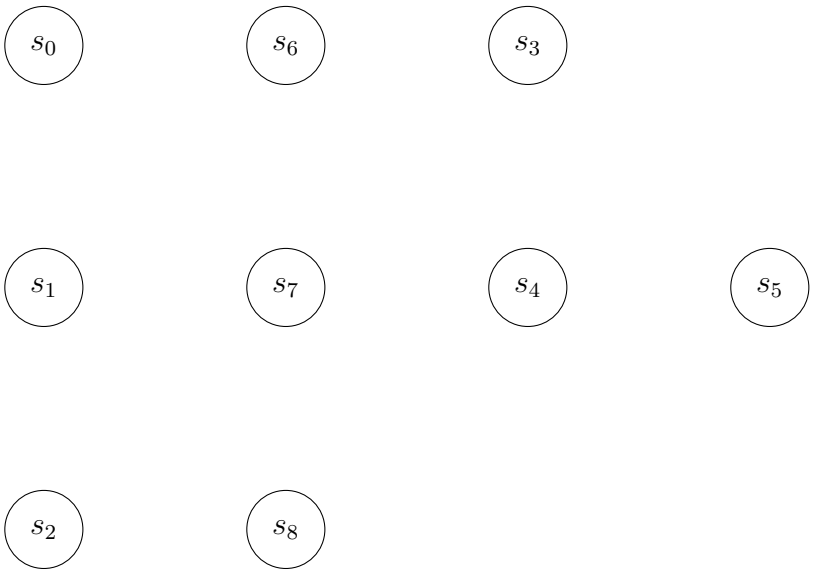

Figure 1: Simplified representation of $\mathcal{M}$ (left) and the view $\mathcal{M}_I$ on it(left)

3 View

Views are the central objective of this thesis. The purpose 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. topic init states

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 an 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 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. It is also very close to the actual implementation later on. 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 $\xi := \{\bullet\} \cup \bigcup_{n \in \mathbb{N}} \{t \in \{\bullet\}^{n+1} \mid \exists t' \in \xi : t' \in \{\bullet\}^n\}$.

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 $\bullet$. This generalization to arbitrary large tuples with $\bullet$ is needed for composition in section 3.4.

Definition 3.4. Let F_θ be 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 and F_θ a grouping function. A *view* is 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

$$\mathbf{P}'([s_1]_R, \alpha, [s_2]_R) = \frac{1}{|[s_1]_R|} \sum_{\substack{s_a \in [s_1]_R, \\ s_b \in [s_2]_R}} \mathbf{P}(s_a, \alpha, s_b)$$

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

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

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

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 a view should take parameter and we want to talk about an instance of a view with specific parameters $p_1, \ldots, p_n$ we will write it as $\mathcal{M}_\theta(p_1, \ldots, p_n)$. 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.

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

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

Definition 3.6. Let $\mathcal{M}_\theta$ be a view on an MDP $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$. $\mathcal{M}_\theta$ is called *binary* if for its grouping function F_θ it is $M(F_\theta) \in \{\{\top, \bot\}, \{\bullet, \bot\}, \{\top\, \bullet\}\}$. $\mathcal{M}_\theta$ is called categorizing if for its grouping function F_θ it is $\top, \bot \notin M(F_\theta)$.

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 $M(F_\theta) = \{\top, \bot\}$ 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 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 with 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 ${\mathcal{M}_\theta}^\perp$ from $\mathcal{M}_\theta$ with

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

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 different notions of compositions. All variants will ensure that the effect caused by each partaking views 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. There may be restrictions in the order of removal.

3.4.1 Parallel Composition

In this section we will introduce the concept of parallel composition. The most apparent idea 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 $F_{\theta_1||\theta_2||\ldots||\theta_n} : S \to M$ 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 defying 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 define the operator $||$ in order to build and modify parallel composed views.

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

1. $(F_{\theta_1} || F_{\theta_2})(s) := (F_{\theta_1}(s), F_{\theta_2}(s)) = F_{\theta_1||\theta_2}(s)$
   if it is not $F_{\theta_1} = F_x || F_y$ or $F_{\theta_2} = F_x || F_y$ for some grouping functions F_x, F_y
2. $((F_{\theta_1}, \ldots, F_{\theta_n}) || F_\theta)(s) := (F_{\theta_1}(s), \ldots, F_{\theta_n}(s), F_\theta(s)) = F_{\theta_1||\theta_2||\ldots||\theta_n||\theta}(s)$
   if for all $i \in \{1, \ldots, n\}$ it is not $F_{\theta_i} = F_x || F_y$ for some grouping functions F_x, F_y

We write $\mathcal{M}_{\theta_1} || \mathcal{M}_{\theta_2}$ for $\mathcal{M}_{\theta_1||\theta_2}$ defined by the grouping function grouping function $F_{\theta_1} || F_{\theta_2} = F_{\theta_1||\theta_2}$.

Note that the operator $||$ is not associative, since it is left evaluated. Moreover, for expressions of the form $F_\theta || (F_{\theta_1}, \ldots, F_{\theta_n})$ it is not defined. This is no issue since parallel composed grouping function defy order. For the same reason $||$ is communicative, with respect to the induced relation R.

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 a 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 $F_{\theta_1||\ldots||\theta_{i-1}||\theta_{i+1}||\ldots||\theta_n}$.

3.4.2 Selective Composition

Selective composition is a variant of parallel 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 $F_{\theta_1||\theta_2||\ldots||\theta_n}\ ||_Z\ F_\theta := F_{\theta_1||\theta_2||\ldots||\theta_n||\theta} : S \to M$ 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 $\mathcal{M}_{\theta_1||\theta_2||\ldots||\theta_n||\theta}$ 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 Components

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. The notion is to group states that were assigned the same set of atomic propositions. Why useful?, currently no "has AP" → maybe should be added

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.

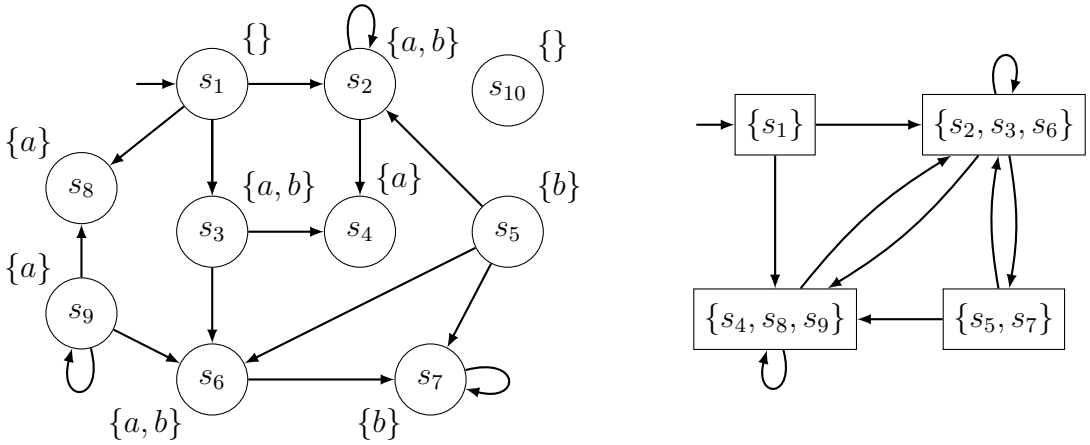

Figure 2: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{AP}$ on it (right)

In Figure 2 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 $\mathcal{M}_I$ is defined by its grouping function F_I where $\tilde{F_I} : \tilde{S} \to M$ with

$$F_I(s) = \begin{cases} \bullet, & \text{if } s \in I \\ \bot, & \text{otherwise} \end{cases}$$

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

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

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

By this we obtain the view $\mathcal{M}_I$ for a given an MDP $\mathcal{M}$ where: $S' = \bigcup_{s \in S} \{[s]_R\} = \{s \in S \mid s \in I\} \cup \bigcup_{s \in S \setminus I} \{\{s\}\}$.

All other components are constructed as in Definition 3.5.

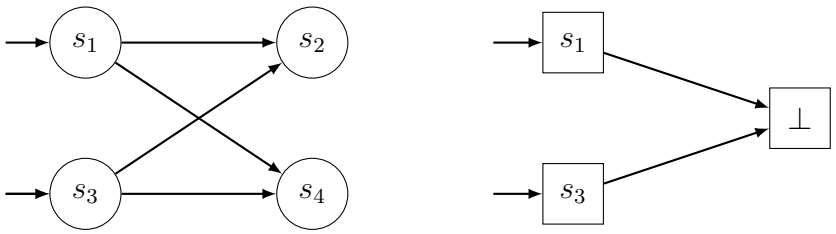

Figure 3: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_I$ on it (right)

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 dicuss some views utilizing actions on transitions that are outgonig from a state. We will write a state s has an outgoing action α if there exists a state s' with $(s, \alpha, s') \in \mathbf{P}$. For a state s, ingoing action α we say it has an ingoing action α if there exist a state s' with $(s', \alpha, s) \in \mathbf{P}$.

The most obvious variant 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 $\mathcal{M}_{\exists\overrightarrow{\alpha}}$ is defined by its grouping function $F_{\exists\overrightarrow{\alpha}}$ where $\tilde{F}_{\exists\overrightarrow{\alpha}} : S \to M$ with

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

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

For $s_1, s_2 \in S$ it is $F_{\exists\overrightarrow{\alpha}}(s_1) = F_{\exists\overrightarrow{\alpha}}(s_2)$ 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 the same outgoing action α) or $s_1 = s_2$. In accordance with Definition 3.4 it is

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

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

$$\begin{aligned} S_1 &:= \{s \in S \mid \exists s' \in S : (s, \alpha, s') \in \mathbf{P}\} = \{s \in S \mid s \text{ has outgoing action } \alpha\} \\ S_2 &:= \bigcup_{s \in S \setminus S_1} \{\{s\}\}. \end{aligned}$$

Since actions are a very important part of MDPs as well as of its more powerful siblings MDPs and MCs 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.

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 $\mathcal{M}_{n \leq \overrightarrow{\alpha}}$ is defined by its grouping function $F_{n \leq \overrightarrow{\alpha}}$ where $\tilde{F}_{n \leq \overrightarrow{\alpha}} : \tilde{S} \to M$ with

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

where $M := \{\bullet, \bot\}$, $n \in \mathbb{N}$ 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.

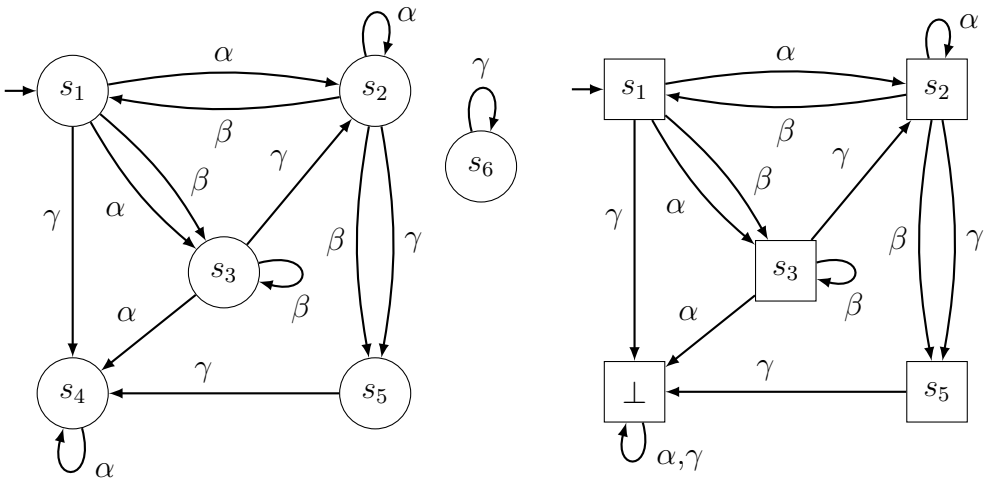

Figure 4: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\exists\overrightarrow{\alpha}}$ on it (right)

The number and 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 $F_{n\leq\overrightarrow{\alpha}}(s_1) = F_{n\leq\overrightarrow{\alpha}}(s_2)$ 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

$$\begin{aligned} [s]_R &= \{s \in S \mid F_{n\leq\overrightarrow{\alpha}}(s) = \alpha\} && \text{if } Q_{n\leq\overrightarrow{\alpha}}(s, s_1, \ldots, s_n) \\ [s]_R &= \{s \in S \mid F_{n\leq\overrightarrow{\alpha}}(s) = s\} = \{s\} && \text{otherwise} \end{aligned}$$

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

$$\begin{aligned} S_1 &:= \{s \in S \mid \exists s_1, \ldots, s_n \in S : Q_{n\leq\overrightarrow{\alpha}}(s, s_1, \ldots, s_n)\} \\ &= \{s \in S \mid \text{the action } \alpha \text{ is outgoing at least } n \text{ times}\} \text{ and} \\ S_2 &:= \bigcup_{s\in S\setminus S_1} \{\{s\}\}. \end{aligned}$$

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 $\mathcal{M}_{\overrightarrow{\alpha}\leq n}$ is defined by its grouping function $F_{\overrightarrow{\alpha}\leq n}$ where $\tilde{F}_{\overrightarrow{\alpha}\leq n} : \tilde{S} \to M$ with

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

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 α 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 $\mathcal{M}_{n\leq\vec{\alpha}}$.

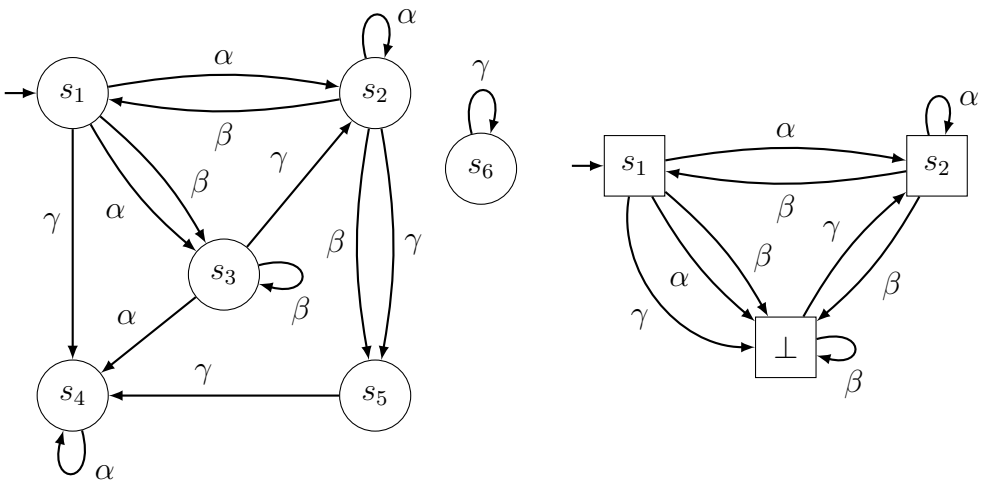

Figure 5: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{2\leq\vec{\alpha}}$ on it (right)

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 $\mathcal{M}_{m\leq\vec{\alpha}\leq n}$ is defined by its grouping function where $\tilde{F}_{m\leq\vec{\alpha}\leq n} : \tilde{S} \to M$ with

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

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 $Q_{n\leq\vec{\alpha}}(s, s_1, \ldots, s_n)$ and $Q_{\vec{\alpha}\leq n}(s, s_1, \ldots, s_{n+1})$ are the predicates from Definition 4.4 and Definition 4.5 respectively.

We already know that for a given $s \in S$ the expressions $\exists s_1, \ldots, s_n \in S : Q_{n\leq\vec{\alpha}}(s, s_1, \ldots, s_n)$ and $\forall s_1, \ldots, s_{m+1} \in S : Q_{\vec{\alpha}\leq m}(s, s_1, \ldots, s_{m+1})$ 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

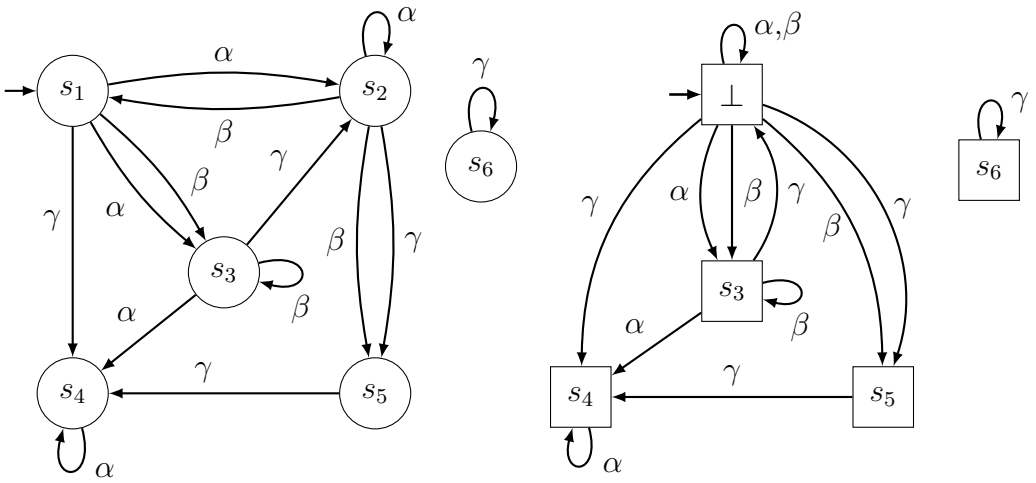

Figure 6: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\overrightarrow{\alpha}\leq 1}$ on it (right)

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

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

The new set of states S' of the view $\mathcal{M}_{m\leq\overrightarrow{\alpha}\leq n}$ 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

$$\begin{aligned} S_1 &:= \{s \in S \mid F_{m\leq\overrightarrow{\alpha}\leq n}(s) = \alpha\} \\ &= \{s \in S \mid C_{s,\overrightarrow{\alpha}} \text{ true}\} \\ &= \{s \in S \mid \text{the action } \alpha \text{ is outgoing } m \text{ to } n \text{ times}\} \text{ and} \\ S_2 &:= \bigcup_{s\in S\setminus S_1} \{\{s\}\}. \end{aligned}$$

The views above can be combined with parallel composition. The thereby obtained view requires that the respective conditions of all the combined views are met. In this sense it is a conjunctive combination.

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

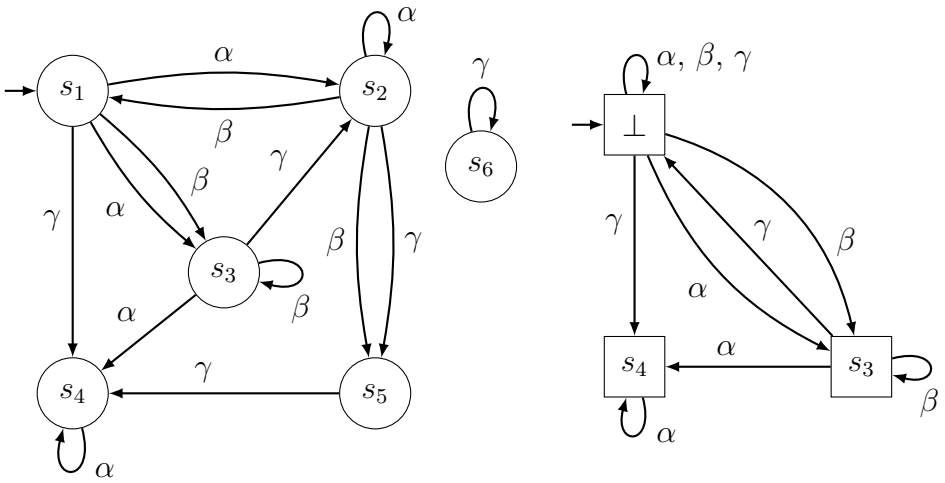

Figure 7: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{1\leq\vec{\alpha}\leq 1}$ on it (right)

or even identical in regard to their outaction. We consider this idea with the *OutActionsIdent View* in two variants: strong and weak (idententiy). Firstly we will consider the variant of strong identity.

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 $\tilde{F}_{\overrightarrow{Act(s)_=}} : \tilde{S} \to M$ with

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

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

The grouping function asserts to each state a set of pairs. Note that a pair is contained int 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

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

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. As mentioned earlier a weak variant of the OutActionsIdent view is also conceivable.

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

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

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

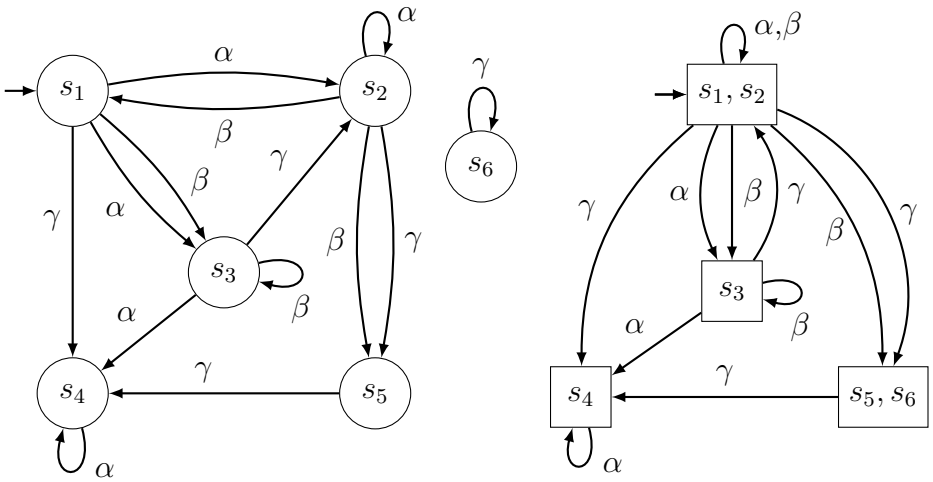

Figure 8: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\overrightarrow{Act}(s)_=}$ on it (right)

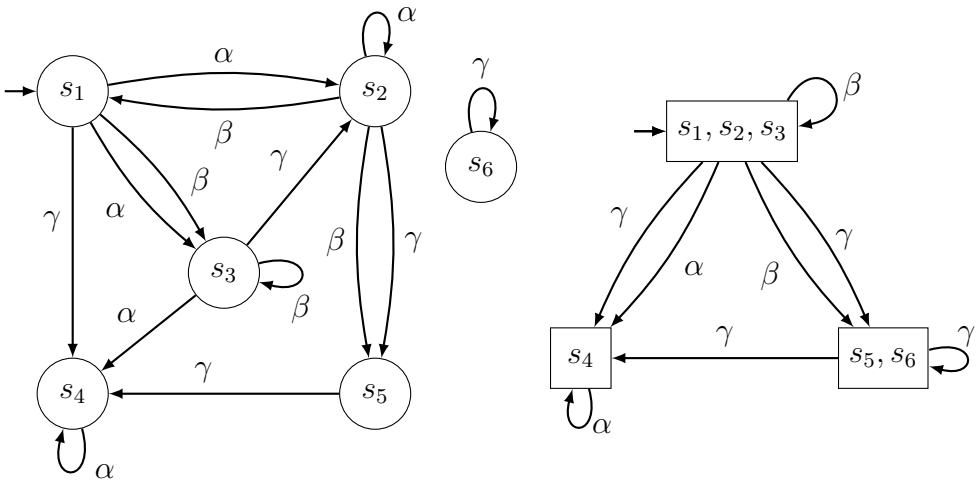

Figure 9: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\overrightarrow{Act}(s)_\approx}$ on it (right)

condition of image-set written in inconsistent style to strong identity. Swap strong and weak (order)?

The grouping function maps to the set of outgoing actions of a state and thereby discards information about the number of times an actions is outgoing. If an action is not outgoing from a state it is not contained in the set.

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

$$[\tilde{s}]_R = \{s \in S \mid F_{\overrightarrow{Act}(s)_\approx}(s) = F_{\overrightarrow{Act}(s)_\approx}(\tilde{s}) =: \{\alpha_1, \ldots, \alpha_l\}, l \in \mathbb{N}\}.$$

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

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 the 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

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

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

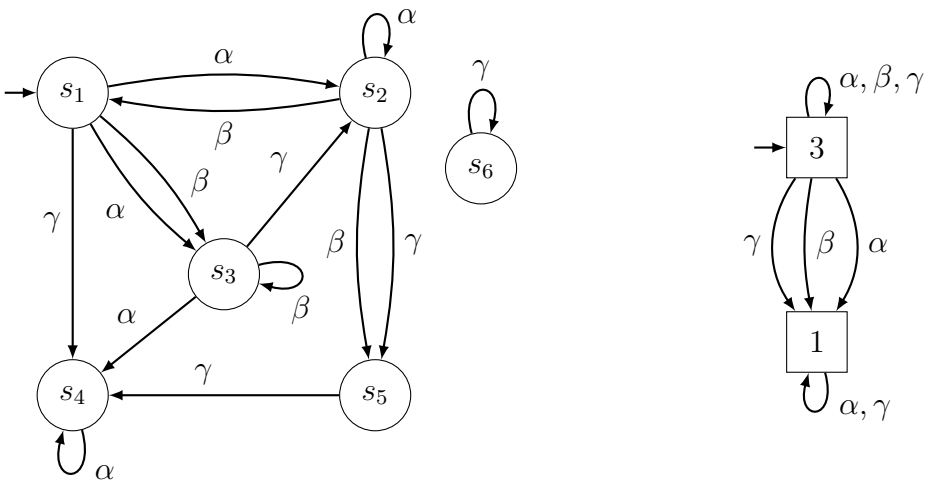

Figure 10: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{|\overrightarrow{Act}(s)|}$ on it (right)

The notion of the view 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 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 $\mathcal{M}_{|\overrightarrow{Act}(s)|_1}$ is defined by its grouping function $F_{|\overrightarrow{Act}(s)|_1}$ where $\tilde{F}_{|\overrightarrow{Act}(s)|_1} : \tilde{S} \to M$ with

$$s \mapsto \begin{cases} \bullet, & \text{if } |\{\alpha \in Act \mid \exists s' \in S : (s, \alpha, s') \in \mathbf{P}\}| = 1 \\ \bot, & \text{otherwise} \end{cases}$$

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

4.1.4 Ingoing Actions

Analogously to Outgoing Actions views of utilizing ingoing 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 $\mathcal{M}_{\exists \overleftarrow{\alpha}}$ is defined by its grouping function $F_{\exists \overleftarrow{\alpha}}$ 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 $M := \{\bullet, \bot\}$.

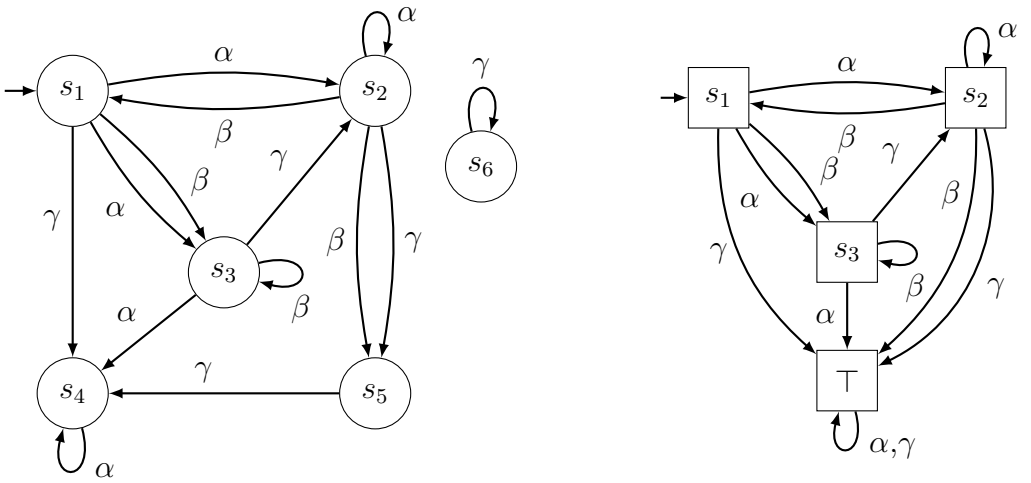

Figure 11: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{|\overrightarrow{Act(s)}|}$ on it (right)

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 $\mathcal{M}_{n \leq \overleftarrow{\alpha}}$ is defined by its grouping function $F_{n \leq \overleftarrow{\alpha}}$ 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 ingoing 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 $\mathcal{M}_{\overleftarrow{\alpha} \leq n}$ is defined by its grouping function $F_{\overleftarrow{\alpha} \leq n}$ 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 ingoing 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 $\mathcal{M}_{m \leq \overleftarrow{\alpha} \leq n}$ 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

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

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

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 $M := Act \cup \{\bot\}$.

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 $\tilde{F}_{|\overleftarrow{\alpha}(s)|} : \tilde{S} \to M$ with

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

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

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 com 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 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. Both of these options are rather tedious but possible. 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.*

Most of the time we will use $VarEval_{\mathcal{M}}$ to refer to the variable evaluation function. 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.

state has variable not here uptil now because probably not used

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 $F_{x=a}$ where $\tilde{F}_{x=a} : \tilde{S} \to M$ with

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

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 $F_{x=a}(s_1) = F_{x=a}(s_2)$ 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

$$[s]_R = \{s \in S \mid VarEval_{\mathcal{M}}(s, x) = a\}$$
$$[s]_R = \{s \in S \mid F_{x=a}(s) = s\} = \{s\}$$

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

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

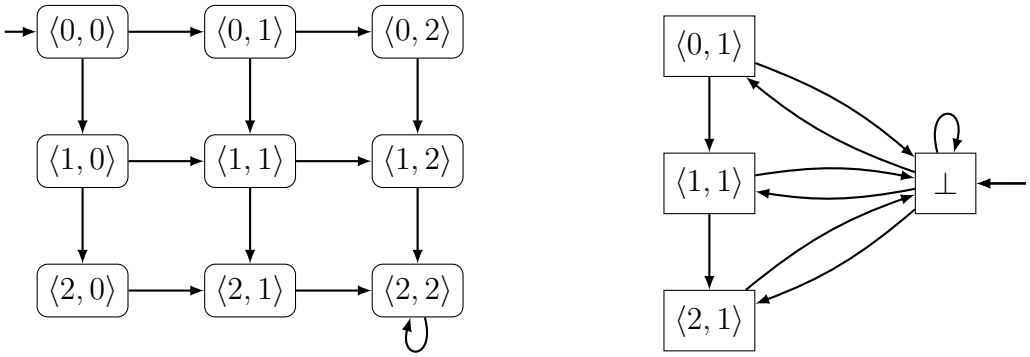

Figure 12: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\overrightarrow{dist}}$ on it (right)

Analogously a view that requires inequality instead of equality is feasible.

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 $\doteqdot$ to be an 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 $(x_{s,i} \doteqdot a)$ 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

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

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

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

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.

DISCUSSION OF EQUALITY, EQ CLASSES AND RESULTING STATES MISSING

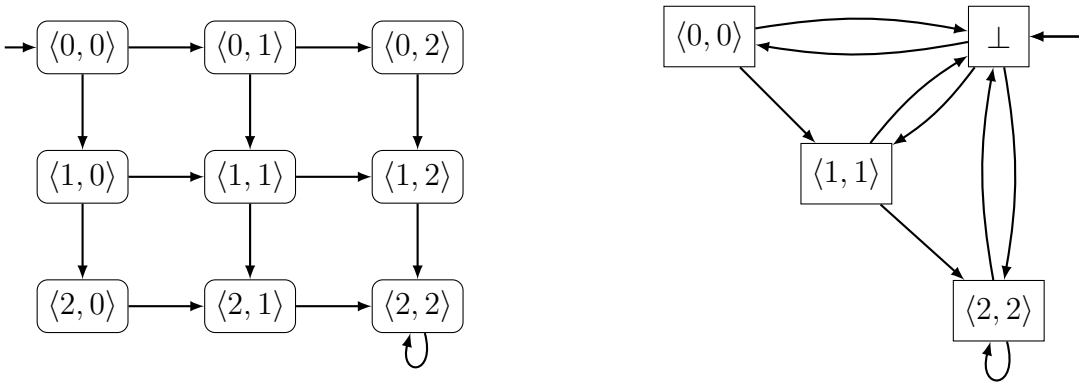

Figure 13: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{VarDNF}$ on it (right)

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

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

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

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

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

The only difference from the view $\mathcal{M}_{VarCNF}$ to the view $\mathcal{M}_{VarDNF}$ is whether the respective formulae is in conjunctive or disjunctive normal form.CITATION Since each formulae in conjunctive normal form can tan be transformed to a formulae in disjunctive normal form and vice verse 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.

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. This idea could be achieved with a view based on the grouping function F_{a_1}

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 $M := VarEval_{\mathcal{M}}(S, x)$.

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 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 $\mathcal{M}_{Var:a}$.

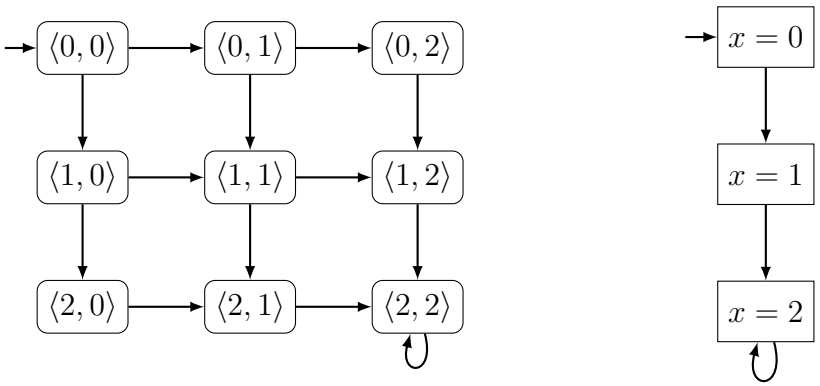

Figure 14: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{Var:a}$ on it (right)

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.

definition is very simlar to execution fragments! Citation?

Definition 4.24. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP. A (simple and finite) *path* P is a sequence $(s_1, \alpha_1, s_2, \alpha_2, \ldots, \alpha_{n-1}, s_n)$ 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 4.25. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP. A (simple and finite) *undirected path* $\bar{P}$ is a sequence $(s_1, \alpha_1, s_2, \alpha_2, \ldots, \alpha_{n-1}, s_n)$ 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 4.26. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP and $P = (s_1, \alpha_1, s_2, \alpha_2, \ldots, \alpha_{n-1}, s_n)$ be a path in $\mathcal{M}$. The number $n =: len(P)$ is called *length* of the path P.

Definition 4.27. 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 $s_2 \in S_2$. That is, if $S_1 \cap S_2 = \emptyset$ it is

$$\overrightarrow{dist}(\mathcal{M}, S_1, S_2) := \min\{len(P) \mid P \in P_{\mathcal{M}},\ first(P) \in S_1,\ last(P) \in S_2\},$$

If $S_1 \cap S_2 \neq \emptyset$, it is $\overrightarrow{dist}(\mathcal{M}, S_1, S_2) := 0$.

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 $\overrightarrow{dist}(\mathcal{M}, \{s\}, \tilde{S})$. 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. The following algorithm calculates the distance of each node from a given set $\tilde{S} \subseteq S$ considering granularity n:

Implementation Algorithm

The set $distance(\mathcal{M}, \tilde{S}, n)$ declares the returned set of ALGORITHM. 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 4.28. Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be an MDP, $\tilde{S} \subseteq S$, $n \in \mathbb{N}$ and $\tilde{S} \subseteq S$ arbitrary. $distance(\mathcal{M}, \tilde{S}, n)$ IMPLEMENTATION IN PSEUDOCODE? DEFINITION BEFORE? 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

$$s \mapsto d - (d \bmod n) \qquad \text{where } d = \overrightarrow{dist}(\mathcal{M}, \tilde{S}, s)$$

and $M = \mathbb{N} \cup \{\inf\} \cup \{\bot\}$.

Definition 4.29. 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 to a state $s_1 \in S_1$ from a $s_2 \in S_2$. That is, if $S_1 \cap S_2 = \emptyset$ it is

$$\overleftarrow{dist}(\mathcal{M}, S_1, S_2) := \min\{len(P) \mid P \in P_{\mathcal{M}},\ last(P) \in S_1,\ first(P) \in S_2\},$$

If $S_1 \cap S_2 \neq \emptyset$, it is $\overleftarrow{dist}(\mathcal{M}, S_1, S_2) := 0$.

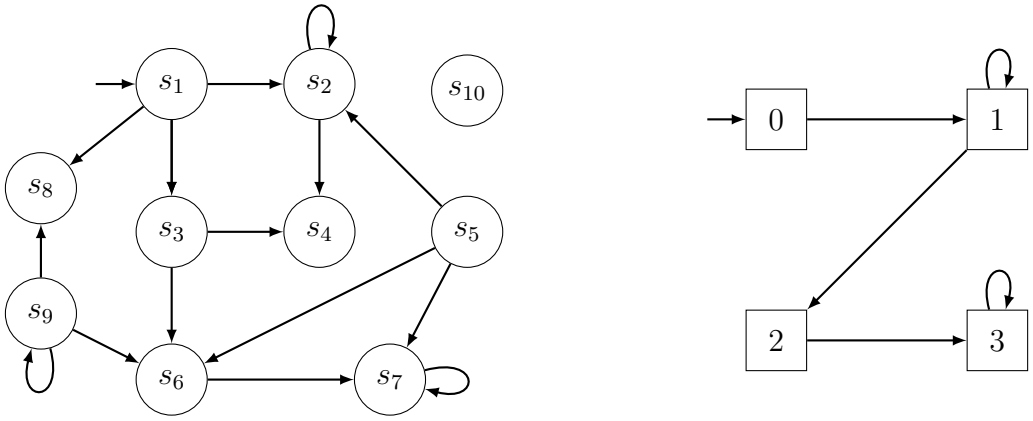

Figure 15: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{\overrightarrow{dist}}$ on it (right)

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 $s_1 \in S_1$ to a $s_2 \in S_2$ ignoring the direction of edges. That is, if $S_1 \cap S_2 = \emptyset$ it is

$$\overline{dist}(\mathcal{M}, S_1, S_2) := \min\{len(\bar{P}) \mid \bar{P} \in \bar{P}_{\mathcal{M}},\ first(\bar{P}) \in S_1,\ last(\bar{P}) \in S_2\},$$

If $S_1 \cap S_2 \neq \emptyset$, it is $\overline{dist}(\mathcal{M}, S_1, S_2) := 0$.

Variants of ALGORITHM which allow directinoless or only reverse traversal of the MDP are provided in the appendix. The respective views would utilize their returned set in the exact same way. That is, $distance(\mathcal{M}, \tilde{S}, n)$ is replaced with the set of the respective algorithm and nothing else changes.

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 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 4.31. 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. only if?? 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 4.32. 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 4.33. 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 jave library jGraphtT 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 they pose the risk of getting stuck in endless loops, when performing actions on the MDP. Model checking is one of the most relevant actions on MDPs that are vulnerable to loops. They are performed commonly on them. Therefore a view that simply finds existing cycles is feasible.

Definition 4.34. 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 $\tilde{F}_{\exists C} : \tilde{S} \to M$ with

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

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.
DISCUSSION OF EQUALITY, EQ CLASSES AND RESULTING STATES MISSING

Definition 4.35. 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 $M = \mathcal{P}(C_{\mathcal{M},n}) \cup \{\bot\}$.

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.
DISCUSSION OF EQUALITY, EQ CLASSES AND RESULTING STATES MISSING
GREEDY APPROACHES (STATE ON SINGLE CYCLE), AND SET APPROACH $s \to C_1 \cup C_2$ omitted because not sure if needed
The view above might be useful when having found cycles to see what cycles specifically exist. Often it is interesting to find cycles that consist only of transitions of the same action. The following view accomplishes that.

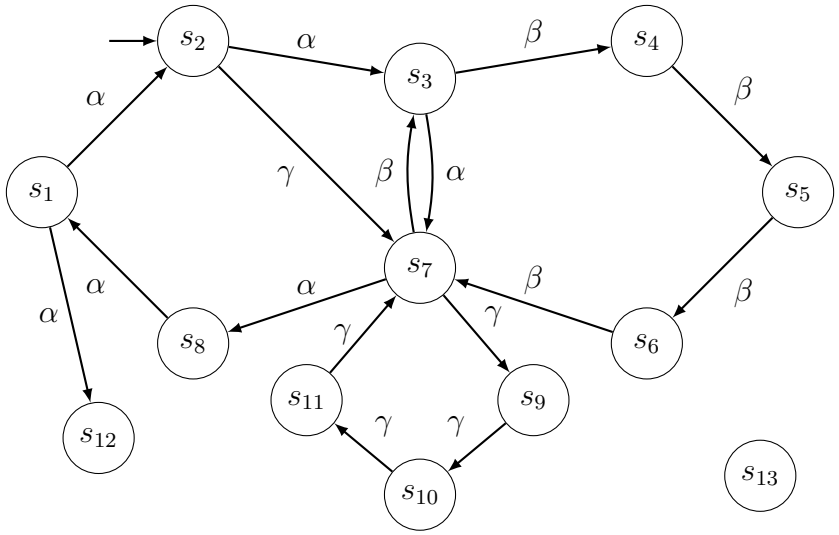

Figure 16: Simplified representation of $\mathcal{M}$

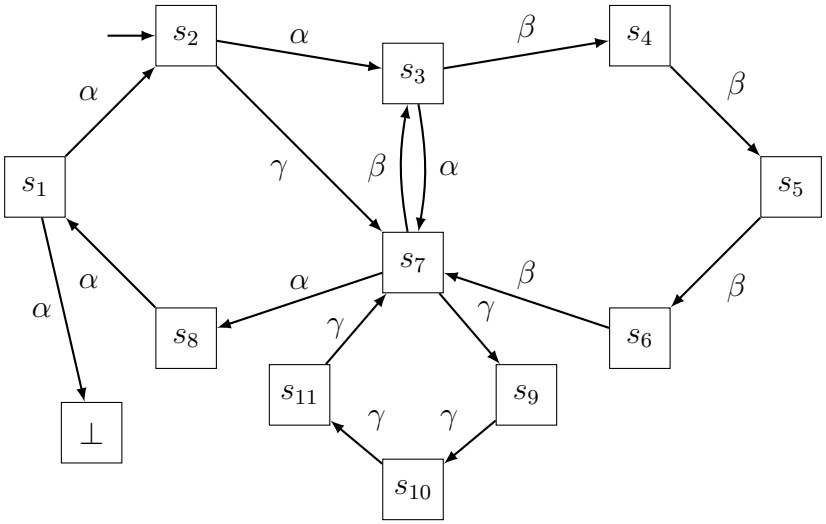

Figure 17: Simplified representation of the view $\mathcal{M}_{\exists C}$ on $\mathcal{M}$ from Figure 16

Definition 4.36. 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 $M = \mathcal{P}(C_{\mathcal{M},n}) \cup \{\perp\}$.

The view specializes the view from Definition 4.35 in the sense that it additionally requires for all $C \in F_{\{C_n\}}$ that all actions occurring in the cycle are the same. doubling to above definition?

NO DISCUSSION IF PREVIOUSLY

In definition 4.35 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 implemenation 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.

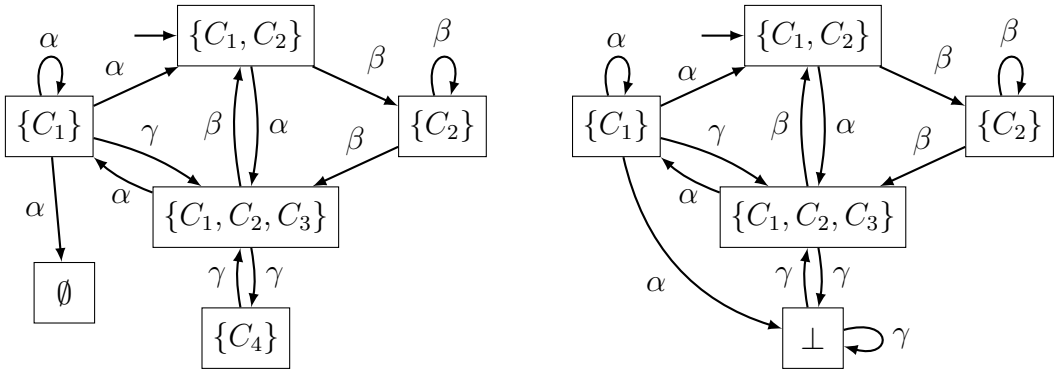

Figure 18: View $\mathcal{M}_{\{C_{\alpha,n}\}}$ [0] (left) and view $\mathcal{M}_{\{C_{\alpha,n}\}}$ [5] (right)

Definition 4.37. 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 $M = S \cup \{\perp\}$.

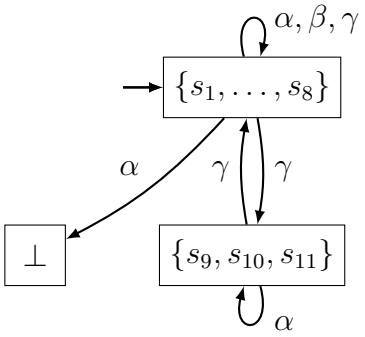

Figure 19: Simplified representation of the view $\mathcal{M}_{\cup\{s\}_C}$ on $\mathcal{M}$ from Figure 16

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

4.2.3 Strongly Connected Components

Strongly connect components (SCC) are of major importance in model checking I think so because I found the terms many times in the book. Remaining question: really? why? 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 4.38. 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 $\mathcal{M}$ is denoted with $SCC(\mathcal{M})$.

Since the strong connection is an equivalence relation the SCCs are equivalence classes and hence disjoint. To find SCCs Tarjans algorithm is the classic. In the implementation an improved variant from Gabow is used supplied by the jGraphtT library. CITATION

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 $\mathcal{M}_{scc}$ is defined by its grouping function F_{scc} where $\tilde{F}_{scc} : \tilde{S} \to M$ with

$$s \mapsto \{T \in SCC(\mathcal{M}) \mid s \in T, n \leq |T|\}$$

and $M = SCC(\mathcal{M}) \cup \{\bot\}$.

This view groups all states together that are in the same SCC. Because SCCs are disjoint each s will be mapped to its one and only SCC. This is because... DISCUSSION OF EQUALITY, EQ CLASSES AND RESULTING STATES MISSING

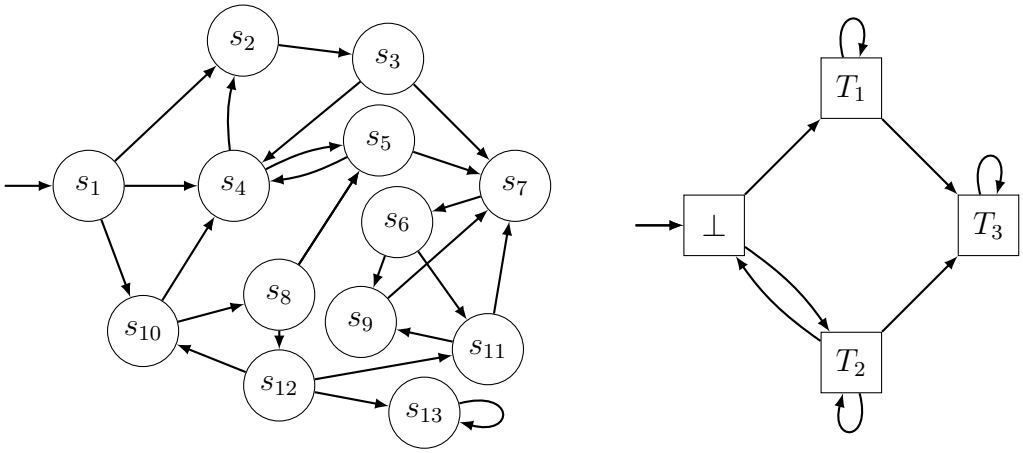

Figure 20: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{scc}$ on it (right)

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

Definition 4.40. Let T be a SCC. A *bottom strongly connected component* (BSCC) is a SCC where it holds that: concurrent versions

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

$$\forall s \in T : \sum_{t \in T} \mathbf{P}(s, t) = 1$$

That is from T there is no state reachable outside of T. The set of bottom strongly connected components of $\mathcal{M}$ is denoted with $BSCC(\mathcal{M})$.

These are of special relevance because..

Definition 4.41. 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

$$s \mapsto \{T \in BSCC(\mathcal{M}) \mid s \in T, n \leq |T|\}$$

and $M = BSCC(\mathcal{M}) \cup \{\bot\}$.

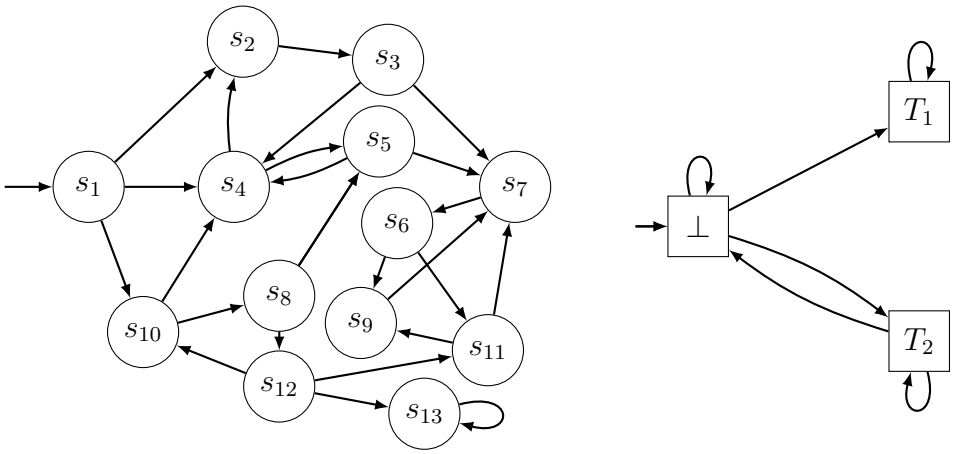

Figure 21: Simplified representations of $\mathcal{M}$ (left) and the view $\mathcal{M}_{bscc}$ on it (right)

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. Equality, equivalence classes and the new state set are constructed analogously to the view $\mathcal{M}_{scc}$.

4.3 Utilizing the MDP Result Table

In this section we will discuss views utilizing the result table. The actual implementation is relies on one powerful view that can set to perform arbitrary actions using model checking results.

needhelp Understanding Cluster

5 View Implementation

supposed to give a short overview how view are implemented from a conceptional perspective. No more than 1 to 2 pages.
$\hat{F}$

- implementation in web application pmc-vis that aims for ...
- the general project structure is ... prism model, database, java, json + roughly were clusters are lokaed in the project
- implementation of views rely on an internal graph structure that was necessary for the views that implement grouping based on structural properties of the MDP graph. Views based on MDP components firstly where implemented with direct accesses on the database but later on also adopted to the internal graph structure for performance reasons
- internal graph structure the jGraphtT library was used. It supplies graph structures as well as common algorithms performed on them. It was chosen because it is the most common, most up to date java library for graphs with the best documentation and broadest functionality. It is developed by ... has a java style documentation and is open source. The broad functionality assured that when implementing a view it is most certainly assured that if necessary a respective algorithm is available.
- The MDP was realized as an directed weighted pseudograph. The transition probabilities are stored as weights. A pseudograph has been chosen because it allows double edges as well as loops.
- in order of keeping the graph lean nodes and vertices are long values. The refer to the state id and the transition id respectively. The state id matches the one in the db whereas the transition id is newly generated because in the database a transition is a action with its probability distribution. This difference to the MDP definition is reversed with the internal graph structure for graph algorithms of the jGraphtT library to work properly on the MDP graph.
- For access to atomic propositions, actions and alike the graph maintains two hashmaps that map the id to the actual transition or state object respectively
- a view is created by a createView() call.
- In general each view is a dedicated java class, derived from an abstract class View, which contains attributes that shall be available in all views, as well as methods that are needed by all views. Many of these are used for testing and I/O. In Figure an overview of relevant attributes and methods is shown.
- most important buildView() which accomplishes build of view.
- In some cases views are merged into one class that realizes combining behavior that otherwise would have been realized by composition (why? $\rightarrow$ was easy, allows more flexibility in functionality)

- when implementing a view only the grouping function is implemented. This makes sense because the process of actually generating the view is always the same as it was already notable in the definitions of a view, where every view is defined by its grouping function.
- the grouping function is calculated and its function values to the result table in a new column.
- visualization in general and also visualization after grouping was not part of the objective of the thesis and already present. As within the thesis there is one general approach how the resulting new graph is generated.
- every view class is registered in a public enum used to select which view is to be generated
- every view class is derived from general view class.
- in the implementation view still have the deprecated name cluster
- every view class has some private attributes to store information like
- every view class has the methods
- the most relevant one is build cluster. It consist of 4 parts. 1. Checking if it is already build aborting if so. 2.create a new column in the database where the results of the grouping function are to be saved. 3. The actual computing part of the grouping function, where for every state an SQL Query adding the respective grouping function value is added to a list of SQL Query strings. 4. all the SQL queries that contain the insertion operation of the grouping function value being executed.
- We will take a look at this in the example..

6 Comparison and Evaluation

Performance Cycles -¿ Selection of states and induced subgraph easily possible perform cycle search only on subset of graph -¿ in general generating views on subgraphs easily possible (only when needed for performance) Clustering exact Cycles when clustering exact cycles

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-VisFigure it appears to be little helpful for understanding and exploring the graph due to its sheer size. Although large this $\mathcal{M}$ 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 $\mathcal{M}_{Var:a}$ (p1) groups all states that have the same value of p1 ignoring the values of the remaining variables. That is, the values of p2 p3 are hidden, which results in only showing the module of philosopher p1 Figure. 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. For example if in Dining Philosophers we were interested in the part of the module where p1 picks their first fork we could use the view $\mathcal{M}_{VarDNF}$ with c(s) = (Figure).

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 $\mathcal{M}_{Var:a} || \mathcal{M}_{Var:a}$. This results in states being grouped where $(F_{Var:a}, F_{Var:a}) = (F_{Var:a}, F_{Var:a})$. Hence only the value of s3 is hidden which results exactly in the desired interleaved model (Figure).

In general the views $\mathcal{M}_{VarDNF}$ and $\mathcal{M}_{VarCNF}$ are very powerful since they allow arbitrary operations on parameters.

- USE: dinging Philosopher
- consider dining Philosophers with
- current graphical Representation looks as follows

- only has about 1000 states in case studies prism mdp with about ... states
- in general seems simple
- parameters cluster: allows looking at just one module
- also interleaving of arbitrary modules
-
- hasAction or OutAction-Ident for specific MDP-that run through configuration phases

6.2 Find out why illegal states is reached

In this chapter 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 state ¡2,2¿ 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.

We observe, that this state is reachable. The questions occurs how and why. In order to obtain this information we should investigate by which state this critical state has been reached. One way of accomplishing that is to look into the database that stores states and transitions. They look as follows.

An even better approach is to look in the model file, which looks as follows:

Persons with experience in working with prism models, might quickly spot the issue, especially because this is a rather small $\mathcal{M}$. With less experienced people or a lot larger 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. Because in the current version of the project not expansion has not been implemented yet we will use a custom view that emulates this feature

Definition 6.1. content...

We will use partial application with $F_{\overrightarrow{dist}}\ (\mathrm{s}) = 1$ to expand that state. The MDP-Graph then looks as in Figure. 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 $y = 1$. Thus, we see that is possible for the second system to send on the medium although it is already occupied by the first system!

When now looking at the prism file we can see why this is the case. In line 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 $\mathcal{M}_{VarDNF}$ (....).

- USE: two process switch
- illegal state has been reached?
- given state 2,2 should not be reachable
- is reachable
- is only!
- apply distance cluster
- emulate not yet feature of expansion
- use identityView (with Parameters)
- see in Database
- fix in File

6.3 Understand and Exploring an MDP + SCC - Cycles?!

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 $\mathcal{M}$reference.

Firstly, we will gather some understanding of the model. We have already seen in Chapter **??** that the variables view can show modules. Firstly lets apply $\mathcal{M}_{Var:a}$ (phases) to understand the module phases Figure. We see that system operates in three different phases. Since we utilize the view $\mathcal{M}_{\overrightarrow{Act}(s)_=}$ Hence we know that the system is working for phase=0, configuring for phase=1 and termination configuration in phase=2. If instead we only apply the view $\mathcal{M}_{Var:a}$ (phases) we see that the system has a timed behavior (Figure). If we view the model with $\mathcal{M}_{Var:a}$ (phases) || $\mathcal{M}_{Var:a}$ (time), we can see how these phases are repeated over time Figure. The view $\mathcal{M}_{scc}$ is intended to give an overview. If we combine it with $\mathcal{M}_{Var:a}$ (time) it yields an excellent overview over the system (Figure), especially if we combine it with $\mathcal{M}_I$ (Figure).

In general we learned that this system runs several times with choosing certain configurations each time before it runs. Its rewards function rewards states that work with a better configuration. A classic model checking value is to determine MIN EXP REWARD and MAX EXP REWARD. For MIN EXP REWARD we obtain x, for MAX EXP REWARD 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

MAX EXP REWARD is infinite. We will now show how views can help to find the cause.

Such behavior of infinite MAX EXP REWARD 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 yields that there are quite large strongly connected components (Figure). To find out where these are locate we compose view the MDP with $\mathcal{M}_{scc} \mid\mid \mathcal{M}_{\overrightarrow{Act(s)=}}$. We see that the strongly connected components are in the configuring phase. Now we can take a look at the prism file, we see that we can arbitrarily often switch the configuration. Hence, it is to be assured that a configuration can only be selected once. An easy way of accomplishing this is to sequentialize the selection of the two configuration: Firstly configuration1 is selected, afterwards configuration2 and finally the configuration phase ends.

- USE: scc with loops
- begin with variables: reason: give overview of certain modules
- time -¿ shows we have timed behavior. After a certain amount of time the evolvement terminates
- phases we can see that reapeated behavior
- interleave phases we can see that we iterate through all phases for number of times
- Still no idea what the model actually does
- since this view has actions it may help to use the outactions view
- see phases
- when interleaving this again with phases see that in phase=0 ... and will eventually terminate in phase=0
- config2 also configuration but seems suspicious
- time and phases ¡- both beautifully show structure
- time apply strongly connected components -¿ structure of mdp very clear
- apply init to see where it start
- maxReward is infinite -¿ not wanted
- search for cycles
- are cycles → exact cycles maybe exact cycles
- fix mdp → no more cycles → show no more cycles

6.4 Performance

6.5 Critical remarks

init cluster obsolete no found usage for quantity on of actions or exact identity

7 Outlook

Many other views accomplishable with the properties view. ie Initcluster AP Cluster has Cycle All implemented Clusters in Appendix (only definition) Own implementation of cycles with same actions $\mathbb{Z}_n$ $[s]_R$s $\mathbb{N}$ in symbol list Implementation of own algorithms Distance Cluster Forward backward both $\circlearrowleft$ 870 End Components possible probabilities = 1