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: July 14, 2023

Dipl. Inf Max Korn (Theoretische Informatik)

Contents

Abstract **1**

1 Introduction **2**

2 Preliminaries **6**

3 View **9**
3.1 Grouping Function 9
3.2 Formal Definition 10
3.3 Composition of Views 10
3.3.1 Parallel Composition 11

4 View Implementation **13**

5 Comparison and Evaluation **15**

6 Outlook **16**

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 MDP. 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

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 $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$ be MDP an M be and arbitrary set with $\bot \in M$. We call any function $F_\theta : S \to M$ a *grouping function*. The symbol θ is a unique identifier.

The identifier θ normally hints the objective of the grouping function. Two states are grouped (should be Definition?) 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 $\bot$. The mapping to the symbol $\bot$ will be used whenever a state is supposed to be excluded from the grouping. The definition offers an easy way of defining 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 3.2. Let F_θ be a grouping function. We define the equivalence relation $R := \{(s_1, s_2) \in S \times S \mid F_\theta(s_1) = F_\theta(s_2) \neq \bot\} \cup \{(s, s) \mid s \in S\}$

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 $s_1, s_2 \in S$. If $F_\theta(\tilde{s}) = \bot$ with $\tilde{s}$ being one of states s_1 or s_2 it is $(s_1, s_2) \notin R$. Hence when considering transitivity and symmetry we assume $F_\theta(s_1), F_\theta(s_2) \neq \bot$. 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 3.2 and use its equivalence classes $[s]_R$ ($s \in S$) as states for the view. The rest of the MDP is structured accordingly.

Definition 3.3. 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 3.2.

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 $\mathcal{M}_\theta$. 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.

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

other versions feasible??

3.3 Composition of Views

In essence views are simplification generated from MDP. It seems rather obvious that the composition of views is a very practical feature. 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.

3.3.1 Parallel Composition

One of the most uncomplicated ideas 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 3.4. 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 its grouping functions. A *parallel composed grouping function* is a grouping function $F_{\theta_1||\theta_2||\ldots||\theta_n} : S \to M$ with

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

The respective parallel composed view is denoted with $\mathcal{M}_{\theta_1||\theta_2||\ldots||\theta_n}$.

The operator $||$ used in the definition is inspired from the operator used in electric circuits, when the respective elements are connected in parallel.

If we want to speak about this grouping function in a general way, where it is only of importance that we refer to this type of composition and the given grouping functions are of no importance, we will denote a *parallel composition grouping function* with $F_{||}$.

The operator $||$ in $F_{||}$ is associative with respect to the grouped states. That is exactly the same states are grouped no matter where parenthesis are put.

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

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

Proof. Let F_u, F_v, F_w be grouping functions and S a set of states with $s \in S$. It is

$$F_{u||v}(s) = (F_u(s), F_v(s)) \qquad F_{v||w}(s) = (F_v(s), F_w(s))$$
$$F_{(u||v)||w}(s) = ((F_u(s), F_v(s)), F_w(s)) \qquad F_{u||(v||w)}(s) = (F_u(s), (F_v(s), F_w(s))$$

Let $s_1, s_2 \in S$ be arbitrary. Moreover let

$$F_{(u||v)||w}(s_1) =: ((a,b),c) \text{ and } F_{(u||v)||w}(s_2) =: ((x,y),z). \quad \text{Then it is} \quad F_{u||(v||w)}(s_1) = (a,(b,c)) \text{ and } F_{u||(v||w)}(s_2) = (x,(y,z)).$$

Then it is

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

This is because

$$\begin{aligned} & ((a,b),c) = ((x,y),z) \\ \iff \quad & (a,b) = (x,y) \wedge c = z \\ \iff \quad & a = x \wedge b = y \wedge c = z \\ \iff \quad & a = x \wedge (b,c) = (y,z) \\ \iff \quad & (a,(b,c)) = (x,(y,z)) \end{aligned}$$

□

The operator $||$ in $F_{||}$ is also commutative with respect of the states being grouped.

Proposition 3.2. *Let F_u and F_v be grouping functions and S a set of states. For all $s_1, s_2 \in S$ it holds that*

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

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

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

Then it is

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

It follows that

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

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

□

Other variants missing if existent Subsetapplication missing

4 View Implementation

supposed to give a short overview how view are implemented from a conceptional perspective. No more than 1 to 2 pages.

- 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 predefined algorithms 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
- depending on enum value according cluster is created
- normally each view is a dedicated java class, sometimes 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..

5 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 Outlook

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