

# An Equivalence and Minimization Algorithm for CTMCs

von  
Stefan Herting

## Diplomarbeit

in Informatik

vorgelegt der  
Fakultät für Mathematik, Informatik und Naturwissenschaften der  
Rheinisch-Westfälischen Technischen Hochschule Aachen  
im Januar 2010

Angefertigt am  
LEHRSTUHL FÜR INFORMATIK II  
– SOFTWAREMODELLIERUNG UND VERIFIKATION –  
bei Prof. Dr. Ir. Joost-Pieter Katoen  
Zweitgutachter Prof. Dr. Dr. h.c. Wolfgang Thomas

Betreut durch Martin Neuhäuser



Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Aachen, im Januar 2010



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Preliminaries</b>	<b>5</b>
2.1	Continuous-time Markov chains . . . . .	5
2.2	Probability measures on CTMCs . . . . .	7
2.3	Properties of CTMCs . . . . .	9
2.4	Minimization of CTMCs . . . . .	11
<b>3</b>	<b>Markovian testing minimization equivalence</b>	<b>15</b>
3.1	Markovian testing equivalence . . . . .	16
3.2	Markovian testing minimization equivalence . . . . .	20
3.2.1	Quotient under MTME . . . . .	24
3.2.2	Equivalence of CTMCs under MTME . . . . .	26
3.3	Comparison of equivalence relations . . . . .	29
<b>4</b>	<b>Preserved properties</b>	<b>35</b>
4.1	Properties which are not preserved . . . . .	35
4.2	Preservation of time-bounded reachability . . . . .	35
<b>5</b>	<b>Minimization algorithm</b>	<b>45</b>
5.1	Algorithm . . . . .	45
5.1.1	Example run of the algorithm . . . . .	48
5.1.2	Correctness proof . . . . .	50
5.1.3	Further properties . . . . .	54
5.2	Heuristics for the non-deterministic choice . . . . .	55
5.2.1	Combine a modified Knuth's Algorithm X with branch & bound . . . . .	59
5.2.2	Fast probabilistic approach . . . . .	64
5.3	Minimization Workflow . . . . .	64
<b>6</b>	<b>Case studies</b>	<b>67</b>
6.1	Case study 1: Simple peer-to-peer protocol . . . . .	67
6.2	Case study 2: Last-in first-out supply process . . . . .	69
<b>7</b>	<b>Conclusion</b>	<b>73</b>
7.1	Further work and open questions . . . . .	74



# Chapter 1

## Introduction

Software and hardware systems have become more and more complex, while at the same time the cost of failure of some of these systems has grown tremendously. These costs can go beyond monetary losses, since the failure of some systems can pose a threat to life and limb. So we are in need of assurance that, simply put, systems do what they are supposed to do.

The techniques in the field of *model checking* are among those that can give us such assurance. Model checking can establish if the model of a system adheres to its specification. It does this by comparing a model of the system with a property, expressed as a logical formula, from the specification. Given a system model and a formula the process to determine if the model satisfies the formula, and thus the system has a certain property, is fully automatic. In the language of mathematical logic, we check if the system is a model of the logical formula, i.e. satisfies it; hence the name model checking.

The system models used in model checking are based on directed graphs: Each state of the system corresponds to one node in the underlying graph. Such a node is simply called a *state* and the set of all nodes the *state space*.

Unfortunately, the number of states in the model often grows exponentially with the number of components in the system. This phenomenon is called *state space explosion*. The performance of model checking depends largely on the number of states in the system model, so state space explosion becomes a huge obstacle to model checking for systems of a realistic size. One countermeasure to state space explosion is *state space minimization*; or short just minimization. The idea is to represent a system by a new model which has the same properties, as expressed by logical formulae, but fewer states. We can then run the model checking algorithms on the smaller system model and still get the same results concerning properties as on the full model.

Every minimization technique has to balance the properties it preserves with the reduction in state space it can achieve. So if a technique achieves a larger reduction than another technique it might only be able to do this by preserving fewer properties.

In this thesis we will develop a new minimization technique for a system model called *continuous-time Markov chains* (CTMCs). The standard technique for minimiza-

tion of CTMCs is *bisimulation*. Bisimulation is well-established and in its different variants also the standard technique for minimization of a number of other system models. In this thesis we develop a minimization technique that achieves greater state space reduction than bisimulation. This can only be done at the cost of preserving fewer properties compared to bisimulation.

This new technique is based on *Markovian testing equivalence* (MTE), which was developed by Marco Bernardo in [Ber07a]. Bernardo develops MTE on a process calculus called *sequential Markovian process calculus* (SMPC). Although SMPC has CTMCs as its underlying semantics, MTE on SMPCs is not directly usable for model checking on CTMCs and thus we need to transfer the idea behind MTE to CTMCs. Bernardo shows that MTE includes bisimulation for SMPC and that a number of important properties are preserved. His work serves as a starting point for this thesis. Our goal is to go the whole way from the theoretical underpinnings of the minimization technique to an algorithm which does the actual minimization.

Towards this goal, we first translate MTE to CTMCs and extend it to an equivalence on the state space. It turns out that MTE cannot yield a minimization technique as it does not put all states into relation which can be implicitly merged under MTE. The problem is that MTE implicitly compares sets of states with each other, which cannot be reflected in MTE itself as it is an equivalence between states. To solve this problem we propose Markovian testing minimization equivalence (MTME) based on the idea behind MTE. In the construction of MTME, we manage to avoid an equivalence between sets of states and map the idea to an equivalence between states.

We show that MTME includes bisimulation, which means that MTME minimization always performs at least as well regarding state space reduction as bisimulation. This leaves the question: The preservation of which properties had to be sacrificed for this increase in state space reduction and which remain? Some branching properties are not preserved. We however show that with time-bounded reachability and time-bounded until MTME preserves two important properties.

In the next step towards the goal of a complete minimization technique, we develop an algorithm to compute an MTME for a given CTMC. This algorithm is a partition refinement algorithm but it also has to accommodate for the fact that MTME internally compares sets of states. Because there does not exist a unique maximal MTME the algorithm includes a non-deterministic choice. This choice can be expressed as an exact cover problem with side conditions. So based on Knuth's Algorithm X, which computes solutions to exact cover problems, we develop an inner algorithm.

We show correctness of the algorithm and that it is able to compute, given an MTME, this MTME or a coarser one. The latter is a sort of completeness result as it means that the full minimization potential of MTME can be achieved using this algorithm.

In two case studies we show that MTME performs at least as well as bisimulation and sometimes can achieve 20 to 40 percent additional state space reduction in an acceptable amount of time.

## Outline of the thesis

Chapter 2 introduces and defines the basic concepts needed in this thesis. In the first part of Chapter 3, we present Markovian testing equivalence (MTE) as defined by Bernardo and then translate it to CTMCs. However, MTE is not suitable for minimization of CTMCs, so in the second part of Chapter 3 we develop Markovian testing minimization equivalence (MTME) on the basis of MTE. In Chapter 4 we investigate which properties are preserved by MTME. Chapter 5 presents an algorithm for the computation of an MTME. Two case studies for minimization with MTME are analyzed in Chapter 6. Finally, Chapter 7 summarizes the results of the thesis and highlights where open questions remain and further work seems promising.



# Chapter 2

## Preliminaries

This chapter introduces the basic concepts and definitions for this work. It already places them into the context of model checking and especially the minimization of CTMCs. Most explanations are geared towards giving a working intuition to those who are new to continuous-time or probabilistic model checking and not familiar with its prevailing concepts. For a more in-depth treatment, the reader is referred to additional literature (e.g. [BK08] for model checking in general, [BHHK03] for continuous-time model checking and [ADD00] for probability and measure theory).

### 2.1 Continuous-time Markov chains

Continuous-time Markov chains (CTMCs) describe the systems whose properties we want to investigate by model checking. CTMCs are timed and probabilistic, i.e. they model the passage of time in the system and changes of the state of the system are described probabilistically.

We present CTMCs as an extension of transition systems, which are also known as Kripke structures. Transition systems are the standard formalization in model checking of systems which are neither timed nor probabilistic.

#### Transition system

A transition system (without actions)  $TS$  is a tuple  $(S, \rightarrow, L, I)$ . It has *states*, which represent the states the system can take and the set of states  $S$  is called the *state space* of the transition system. In this thesis the state space is assumed to be finite, although in general it just has to be countable. If a system can go from one state to another state, there is a so called *transition* between them. The transition relation is given by the binary relation  $\rightarrow \subseteq S \times S$ . The set of initial states  $I \subseteq S$  contains all states in which the system can start. Each state has a *labeling* which is determined by the *labeling function*  $L : S \rightarrow 2^{AP}$ . Such a labeling is a subset of a set of atomic propositions  $AP$ . This set of atomic propositions can be an arbitrary fixed set and describes the states of the system.

**Definition 2.1** (Transition System). A transition system is a tuple  $TS = (S, \rightarrow, L, I)$  where,

- $S$  is the set of states,
- $\rightarrow \subseteq S \times S$  the transition relation,
- $L : S \rightarrow AP$  the labeling function, where  $AP$  is the set of atomic proposition, and
- $I \subseteq S$  the set of initial states.

### Continuous-time Markov chain

CTMCs have a similar structure to transition systems but they add information about timing and the probability of a transition. Both aspects are handled at the same time using an exponential distribution on every transition that models the waiting time until the transition is taken.

An exponential distribution has probability density function  $\lambda \cdot e^{-\lambda t}$ , where  $\lambda$  is the so called *rate* of the exponential distribution and  $t$  the argument of the function. The expected value of an exponential distribution with rate  $\lambda$  is  $\frac{1}{\lambda}$ .

Two properties of exponential distributions are important for CTMCs. First of all, it is the only continuous memoryless function. When interpreting the exponential distribution as waiting time, memorylessness means that the probability to wait another  $t$  time units before taking a transition is independent of the time we have already waited. This is important as Markov chains have to fulfill the Markov property which says that the next state of the system and the waiting time until this state is reached only depend on the current state. So in particular, it must not depend on the amount of time we have already waited. The second important property is that the minimum of  $n$  exponentially distributed random variables is again exponentially distributed. If the rates of these random variables are  $\lambda_1, \dots, \lambda_n$ , then the minimum is given by an exponential distribution with rate  $\sum_{i=1}^n \lambda_i$ .

A CTMC is described by a tuple  $\mathcal{M} = (S, R, L, \nu)$ . The set of states  $S$  and the labeling function  $L$  are exactly the same as in a transition system. The transition relation  $\rightarrow$  is replaced by a rate function  $R$  and the set of initial states  $I$  by an initial distribution  $\nu$ . The *rate function*  $R : S \times S \rightarrow \mathbb{R}_{\geq 0}$  assigns a rate  $\lambda$  to each tuple of states  $(s, s')$ . If  $\lambda > 0$ , it is the rate of an exponential distribution that describes the time until this transition fires and is taken. In the case that a state has multiple outgoing transitions, the first transition that fires is taken, which implicitly determines the probability of taking a specific transition.

The *initial distribution*  $\nu$  assigns to each state the probability that the system starts in it.

**Definition 2.2** (Continuous-time Markov chain). A continuous-time Markov chain (CTMC) is a tuple  $\mathcal{M} = (S, R, L, \nu)$ , where

- $S$  is a countable, nonempty set of states,
- $R : S \times S \rightarrow \mathbb{R}_{\geq 0}$  is a rate function,
- $L : S \rightarrow 2^{AP}$  is a labeling function for some finite set of atomic propositions  $AP$  and

- $\nu : S \rightarrow [0, 1]$  is the initial distribution, such that  $\sum_{s \in S} \nu(s) = 1$ .

We define the rate from a state  $s \in S$  into a set of states  $S' \subseteq S$  as  $R(s, S') = \sum_{s' \in S'} R(s, s')$ . The exit rate  $E(s)$  of a state  $s \in S$  is the rate of this state into the whole state space  $S$ :  $E(s) = R(s, S)$ . The exit rate determines the average sojourn time, which is the average time that passes between entering a state and taking an outgoing transition, as  $\frac{1}{E(s)}$ .

We are now interested in the probability to take a specific transition out of a state. So let  $s \in S$  be the current state,  $S = \{s_1, \dots, s_n\}$  and  $\lambda_i = R(s, s_i)$ . Then the probability to take a specific transition is the ratio of the rate of the transition and the exit rate of the state, i.e.  $\frac{R(s, s_i)}{E(s)} = \frac{\lambda_i}{\sum_{j=1}^n \lambda_j}$ .

We sometimes need to consider the union of two CTMCs without initial distribution. The union is simply constructed by putting the two CTMCs side by side. Let  $\mathcal{M}_1 = (S_1, R_1, L_1)$  and  $\mathcal{M}_2 = (S_2, R_2, L_2)$  be CTMCs over the same set of atomic propositions  $AP$ . Furthermore, let them have disjoint state spaces, i.e.  $S \cap S' = \emptyset$ . Then the union CTMC is defined as  $\mathcal{M}_1 \cup \mathcal{M}_2 = (S_1 \cup S_2, R', L')$ , where  $R'$  and  $L'$  mimic the original function for their respective domains and  $R'$  equals 0 for pairs of states from two different state spaces. We will often use this union construction implicitly when we apply definitions to two CTMCs at once.

### Predecessors and successors in CTMCs

A CTMC is sometimes seen as a directed graph by interpreting non-zero rates between states as an edge. Successor and predecessor relationships between (sets of) states can then be defined.

**Definition 2.3** ( $pred(s), post(s)$ ). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $s \in S$ . Then  $pred(s) = \{s' \mid R(s', s) > 0\}$  and  $post(s) = \{s' \mid R(s, s') > 0\}$ .

A set of states has two types of predecessor sets:  $pred(S')$  can be described as the set of all states which have a transition into *any* state of  $S'$ . In contrast to this  $cpred(S')$  describes the common predecessors of  $S'$ , that is, the set of all states which have a transition into *all* states of  $S'$ .

**Definition 2.4** ( $pred(S'), cpred(S')$ ). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $S' \subseteq S$ . Then  $pred(S') = \bigcup_{s \in S'} pred(s)$  and  $cpred(S') = \bigcap_{s \in S'} pred(s)$ .

## 2.2 Probability measures on CTMCs

For this thesis, we need to base probability theory on measure theory since measure theory can provide us with a solid foundation for probabilities on the set of paths of a CTMC. This is necessary since the set of paths of a CTMC, which we still have to formally define, has a discrete and a continuous component. The discrete component being the sequence of states and the continuous component the sequence of sojourn times.

We assume a general understanding of probability theory without measures and try to connect this to measure theory.

In probability theory without measures, we have a set of outcomes, called the *sample space*. We call a subset of the sample space an event and assign a probability to it. Using measure theory, we do not want to consider arbitrary subsets of the sample space but only certain ones. More precisely, we want the set of these subsets of the sample space to form a  $\sigma$ -algebra.

**Definition 2.5** ( $\sigma$ -algebra). Let  $X$  be a non-empty set, called sample space. Then  $F \subseteq 2^X$  is a  $\sigma$ -algebra if

- $X \in F$ ,
- if  $A \in F$ , then also the complement  $X \setminus A \in F$  and
- if  $A_1, A_2, \dots \in F$ , then also the countable union  $\bigcup_{i \in \mathbb{N}} A_i \in F$ .

So a  $\sigma$ -algebra is closed under complement and countable union. Using De Morgan's law, which expresses intersection by union and complement, we see that a  $\sigma$ -algebra is also closed under countable intersection.

The next step is to assign a probability to each element of a  $\sigma$ -algebra. This function is called a *probability measure*.

**Definition 2.6** (Probability measure). Let  $X$  be a non-empty set, and let  $F \subseteq 2^S$  a  $\sigma$ -algebra. Then  $\mu : F \rightarrow [0, 1]$  is a probability measure if

- $\mu(X) = 1$  and
- if  $A_1, A_2, \dots \in F$  and the  $A_i$  are pairwise disjoint sets, then  $\mu(\bigcup_{i \in \mathbb{N}} A_i) = \sum_{i \in \mathbb{N}} \mu(A_i)$ .

So  $\mu$  assigns probability 1 to the whole sample space. Also if we look at pairwise disjoint subsets of the sample space from the  $\sigma$ -algebra and take their union, the probability of this union is the sum of the probabilities of the subsets. Both conditions are expected behavior for probabilities.

We can now introduce a probability measure on sets of paths in a CTMC. The definitions of paths and a probability measure on the set of all paths follow those in [BHHK03].

*Paths* represent, as the name implies, one possible path through a CTMC. They consist of the sequence of states visited and the time that has passed in each state. A path can either be finite and ending in an absorbing state, e.g. a state that has no successor, or it can be infinite.

**Definition 2.7** (Paths). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC. Let  $t_i \in \mathbb{R}_{>0}$  and  $s_i \in S$  for all  $i \in \mathbb{N}$  and  $R(s_i, s_{i+1}) > 0$ . Then a path is either an infinite sequence  $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} \dots$  or it is a finite sequence  $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \dots \xrightarrow{t_{k-1}} s_k$  where  $s_k$  is absorbing, e.g.  $\text{succ}(s) = \emptyset$ .

We call the set of all paths in  $\mathcal{M}$   $\text{Paths}^{\mathcal{M}}$  and its subset of all paths starting in  $s$   $\text{Paths}_s^{\mathcal{M}}$ . We omit the reference to  $\mathcal{M}$  wherever possible. If  $\pi = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} \dots$ ,

the  $i$ -th state in a path is given by  $\pi[i] = s_i$ . The state occupied at time  $t$  is given by  $\pi@t = \pi[k]$  where  $k$  is the smallest index with  $t < \sum_{i=0}^k t_i$ . In order to assign a non-zero probability to paths, we have to consider sets of paths. We do this in the form of cylinder sets. A *cylinder set* is described by a finite alternating sequence of states and time-intervals  $s_0, I_0, s_1, \dots, I_{k-1}, s_k$  which gives the beginning of paths. Formally, it contains all paths  $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \dots \xrightarrow{t_{k-1}} s_k \xrightarrow{t_k} \dots$  where  $t_i \in I_i$  for  $1 \leq i < k$ .

**Definition 2.8** (Cylinder sets). Let  $s_i \in S$  and  $I_i$  non-empty intervals in  $\mathbb{R}_{\geq 0}$  for all  $i \in \mathbb{N}$ . Then

$$Cyl(s_0, I_0, s_1, \dots, I_{k-1}, s_k) = \left\{ \pi \in Paths \mid \pi = s_0 \xrightarrow{t_0} s_1 \dots \xrightarrow{t_{k-1}} s_k \xrightarrow{t_k} \dots, t_i \in I_i, 0 \leq i < k \right\}.$$

The set of all cylinder sets generates a  $\sigma$ -algebra over  $Paths^{\mathcal{M}}$ . In order to assign a probability to cylinder sets, we first have a look at the probability, being in  $s$ , to take the transition  $s \rightarrow s'$  within a time-interval of  $I$ . The probability to take the transition from  $s$  to  $s'$  is  $\frac{R(s,s')}{E(s)}$ . The probability to do so within the time-interval  $I$  is independent of this and can be given as  $e^{-E(s) \cdot \inf I} - e^{-E(s) \cdot \sup I}$ . We can now inductively define the probability  $Pr_\nu$  to take a path in a cylinder set given an initial distribution  $\nu$ .

**Definition 2.9.** The probability measure on cylinder sets is given by  $Pr_\nu(Cyl(s_0, I_0, \dots, s_k, I', s')) = Pr_\nu(Cyl(s_0, I_0, \dots, s_k)) \cdot \frac{R(s_k, s')}{E(s_k)} \cdot (e^{-E(s_k) \cdot a} - e^{-E(s_k) \cdot b})$  where  $a = \inf I', b = \sup I', Pr_\nu(Cyl(s_0)) = \nu(s_0)$  and  $\nu$  an initial distribution.

In the case of a single initial state  $s$ , i.e.  $\nu(s') = 1$  for  $s' = s$  and 0 otherwise, we write  $Pr_s$  instead of  $Pr_\nu$ . This effectively restricts the paths to those starting in  $s$ .

## 2.3 Properties of CTMCs

CTMCs model systems, but what we are ultimately interested in is how these systems behave. We describe this behavior using logical formulae and call the fact if such a formula is satisfied in a state a property. There are a number of logics on CTMCs. We will however restrict ourselves to *continuous stochastic logic* (CSL), as it is presented in [BHHK03]. CSL is interesting insofar as it characterizes bisimulation (cf. [DP03]). We first define the syntax of CSL and then explain the semantics. CSL can be divided into state and path formulae, which are as follows: As the names imply, a state formula is interpreted over a state and a path formula is interpreted over a path of a CTMC. We call a state which satisfies a state formula  $\Phi$  a  $\Phi$ -state.

**Definition 2.10** (Syntax of CSL). Let  $p \in [0, 1]$  be a real number,  $\triangleq \in \{\leq, <, >, \geq\}$  a comparison operator, and  $I \subseteq \mathbb{R}_0^+$  a nonempty interval. Then the syntax of CSL is inductively defined by:

- *true* is a state formula.

- Each atomic proposition  $a \in AP$  is a state formula.
- If  $\Phi$  and  $\Psi$  are state formulae, then so are  $\neg\Phi$  and  $\Phi \wedge \Psi$ .
- If  $\Phi$  and  $\Psi$  are state formulae, then  $X^I\Phi$  and  $\Phi \mathcal{U}^I \Psi$  are path formulae.
- If  $\varphi$  is a path formula, then  $\mathcal{P}_{\leq p}(\varphi)$  is a state formula.
- If  $\Phi$  is a state formula, then so is  $\mathcal{S}_{\leq p}(\Phi)$ .

The most basic type of state formulae are propositional logic formulae over the atomic propositions, which are defined by the first three points in the definition. Each atomic proposition serves as a propositional logic variable. Negation  $\neg$ , conjunction  $\wedge$  and *true* have their usual meaning.

$X_I\Phi$  and  $\Phi \mathcal{U}^I \Psi$  are path formulae and thus a path can either satisfy or not satisfy them.  $X_I$  is the timed next operator, where  $I$  gives a time interval. A path starting with  $s_0 \xrightarrow{t_0} s_1$  satisfies  $X^I\Phi$ , if  $s_1$ , which is the second or next state on the path, satisfies  $\Phi$  and is reached within the time interval  $I$ , i.e. if  $t_0 \in I$ .

$\mathcal{U}^I$  is called the timed until operator. A path satisfies  $\Phi \mathcal{U}^I \Psi$  if there exists a time  $t \in I$  at which we visit a state in which  $\Psi$  holds and all preceding states satisfy  $\Phi$ .

For both operators we omit the time-interval  $I$  if it is no restriction, i.e. if  $I = \mathbb{R}_0^+$ . We now define the satisfaction relation for path formulae.

**Definition 2.11** (Semantics of CSL path formulae). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $\pi \in Paths^{\mathcal{M}}$  a path. Let  $\pi$  start with  $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \dots$ . Then  $\pi$  satisfies a path formula  $\varphi$ , written  $\pi \models \varphi$ , as follows:

$$\begin{aligned} \pi \models X^I\Phi & \quad \text{iff } \pi[1] \text{ is defined, } \pi[1] \models \Phi \text{ and } t_0 \in I. \\ \pi \models \Phi \mathcal{U}^I \Psi & \quad \text{iff there exists a } t \in I \text{ such that } \pi@t \models \Psi \text{ and for all } t' \in [0, t) \\ & \quad \text{holds } \pi@t' \models \Phi. \end{aligned}$$

$\mathcal{P}_{\leq p}(\varphi)$  is satisfied if the probability to encounter a path which satisfies  $\varphi$  is within the range given by  $\leq p$ . For each path formula  $\varphi$ , the set of paths which satisfy  $\varphi$  is expressible by a union of cylinder sets and we can use the probability measure on cylinder sets to determine this probability.

$\mathcal{S}_{\leq p}(\Phi)$  describes that the so called stationary probability of a state formula is within the interval given by  $\leq p$ . Conceptually, the stationary probability gives the probability to be in a  $\Phi$ -state after an infinite amount of time. So it describes the behavior of a CTMC in the long run when a sort of equilibrium has been reached. We omit a formal definition of stationary probability, as it is not needed in the remainder of the thesis and needs a number of supporting definitions.

We can now formally define the semantics of state formulae in CSL.

**Definition 2.12** (Semantics of CSL state formulae). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $s \in S$ .  $s$  satisfies  $\Psi$ , written  $s \models \Psi$ , as follows:

$$\begin{aligned} s \models \text{true} & \quad \text{for all } s \in S. \\ s \models a & \quad \text{iff } a \in L(s). \\ s \models \Psi \wedge \Phi & \quad \text{iff } s \models \Psi \text{ and } s \models \Phi. \\ s \models \neg\Psi & \quad \text{iff not } s \models \Psi. \\ s \models \mathcal{P}_{\leq p}(\varphi) & \quad \text{iff } Pr_s(\{\pi \in Paths^{\mathcal{M}} \mid \pi \models \varphi\}) \leq p. \end{aligned}$$

## Common properties

We now want to present common properties expressible by CSL.

If a state  $s$  satisfies  $\mathcal{P}_{>0}(\varphi)$  there exists a path starting in  $s$  which satisfies  $\varphi$ . However, the reverse is, in general, not true, since a single path does not have probability 0. So if  $\varphi$  is satisfied for a set of paths starting in  $s$  with measure 0, e.g. a single path,  $\mathcal{P}_{>0}(\varphi)$  is not satisfied in  $s$  although there exist paths which satisfy  $\varphi$ . A path which eventually reaches a  $\Phi$ -state can be expressed by the path formula  $true \mathcal{U} \Phi$ . Using  $true$  as the first operand does not put any restrictions on the path leading up to the  $\Phi$ -state.  $true \mathcal{U} \Phi$  is often abbreviated as  $\diamond\Phi$ , pronounced "eventually  $\Phi$ ".

We now combine both and obtain reachability of a  $\Phi$ -state  $\mathcal{P}_{>0}(\diamond\Phi)$ . We can extend this formula in order to express *time-bounded reachability*, which is the property to reach an  $\Phi$ -state with probability at most  $p$  and within at most  $t$  time units. The corresponding formula is  $\mathcal{P}_{<p}(\diamond^{[0,t]}\Phi)$ .

## 2.4 Minimization of CTMCs

Minimization of CTMCs means that a CTMC is transformed into another CTMC with fewer states. This has to be done in such a way that the properties we are interested in are preserved. The benefit is that model checking can be done on the smaller CTMC, which makes it faster or, due to memory constraints, possible in the first place.

All minimization techniques in this thesis follow the same pattern: At first, states of a CTMC that "behave the same" are grouped together via an *equivalence relation*. Then those grouped states are replaced by a single state. The exact meaning of "behaving the same" differs between minimization techniques and is at the core of them.

As the name implies, equivalence relations put elements into relation which are equal in some aspect. An example is the relation that considers the parity of integers, e.g. if they are odd or even, and puts integers into relation if they have the same parity. Equivalence relations are defined as binary relations that are reflexive, symmetric and transitive. These properties do not come as a surprise when keeping the example in mind. Reflexivity means that each element is in relation to itself. Symmetry means that if  $a$  is in relation with  $b$  then also  $b$  with  $a$ . Finally transitivity means that if  $a$  and  $b$  are in relation, as are  $b$  and  $c$  then  $a$  has also to be in relation to  $c$ .

**Definition 2.13** (Equivalence Relation). Let  $M$  be a set.  $E \subseteq M \times M$  is an equivalence relation on  $M$  if it is

- reflexive:  $(a, a) \in E$  for all  $a \in M$ ,
- symmetric:  $(a, b) \in E$  iff  $(b, a) \in E$  and
- transitive: if  $(a, b) \in E$  and  $(b, c) \in E$  then  $(a, c) \in E$ .

We commonly write  $aEb$  as an infix notation for  $(a, b) \in E$ . The elements of the set  $M$  are divided into *equivalence classes* by an equivalence relation. Coming back to the example, one equivalence class contains the odd numbers

and one the even numbers. In the context of minimization, such an equivalence class contains states and is collapsed into a single state.

**Definition 2.14** (Equivalence Classes). Let  $M$  be a set and  $E \subseteq M \times M$  an equivalence relation on  $M$ . Then the equivalence class of  $a$  under  $E$  is defined as  $[a]_E = \{b \in M \mid (a, b) \in E\}$ . The set of all equivalence classes  $\{[a]_E \subseteq M \mid a \in M\}$  is denoted by  $M/E$ .

Note that for all  $(a, b) \in E$ , it holds that  $[a]_E = [b]_E$ . So the choice of representative for an equivalence does not matter.

*Partitions* are strongly related to equivalence relations. A partition divides the elements of a set  $M$  into pairwise disjoint subsets in such a way that the union of these subsets is again the set  $M$ . In other words, every element of  $M$  is contained in exactly one subset.

**Definition 2.15** (Partition). Let  $M$  be a set.  $P \subseteq 2^M$  is a partition of  $M$  if

- $M = \bigcup_{M' \in P} M'$  and
- $M_1 \cap M_2 = \emptyset$  for all  $M_1, M_2 \in P$ .

Given an equivalence relation  $E$  on  $M$ , the set of equivalence classes  $M/E$  forms a partition of  $M$ . Also, the other way around, given a partition  $P$ , it holds that  $\{(a, b) \in M \times M \mid M' \in P, a, b \in M'\}$  is an equivalence relation. So each partition corresponds to an equivalence relation and vice versa. Also this correspondence is symmetric, i.e. if partition  $P$  corresponds to equivalence relation  $E$  then also equivalence relation  $E$  corresponds to partition  $P$ . This means that both concepts can be used interchangeably.

We say a partition  $P$  is coarser than a partition  $P'$  if for every set  $T' \in P'$  there exists a set  $T \in P$  such that  $T' \subseteq T$ . In other words, for every  $T \in P$  there exist  $T'_1, \dots, T'_n \in P'$  which form a partition of  $T$ . The opposite to coarser is finer. So, if  $P$  coarser than  $P'$ , then  $P'$  finer than  $P$ .

## Bisimulation

A *bisimulation*  $\sim$  is an equivalence relation on the states of a CTMC with two additional conditions for every pair of states  $s, s'$  with  $s \sim s'$ : First of all,  $s$  and  $s'$  have to share the same labeling. This is not a surprising condition as the labeling represents the most basic properties of the system the CTMC models and minimization wants to preserve properties. Furthermore, the rate from  $s$  into any other equivalence class  $C$  has to be the same as the rate from  $s'$  to  $C$ . The idea behind this is that otherwise we could distinguish  $s$  from  $s'$  which would result in a difference in properties.

**Definition 2.16** (Bisimulation). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC. A bisimulation on  $\mathcal{M}$  is an equivalence relation  $\sim_{bis}$  such that for all  $s \sim_{bis} s'$  holds

- $L(s) = L(s')$  and
- $R(s, C) = R(s', C)$  for all  $C \in S/\sim_{bis}$ .

An important consequence of the second condition is that also the exit rates of bisimilar states are the same, e.g.  $s \sim_{bis} s'$  implies  $E(s) = E(s')$ .

We call two states bisimilar if there exists a bisimulation which puts them into relation.

For any CTMC there can be, and usually are, different bisimulations. However there is a unique maximal bisimulation which contains all other bisimulations.

**Theorem 2.1.** Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and let  $B \subseteq 2^{S \times S}$  be the set of all bisimulations on  $\mathcal{M}$ . Then  $\sim_{max} = \bigcup_{\sim \in B} \sim$  is a bisimulation and we call  $\sim_{max}$  the maximal bisimulation on  $\mathcal{M} = (S, R, L, \nu)$ .

Two states are put into relation by the maximal bisimulation if and only if they are bisimilar. So the maximal bisimulation can be characterized as  $\sim_{max} = \{(s, s') \in S \times S \mid s \text{ bisimilar } s'\}$ .

## Bisimulation Quotient

Given a bisimulation, we can minimize a CTMC. The resulting CTMC is called the *bisimulation quotient*. Its state space consists of the set of equivalence classes of the bisimulation. In other words, all states of an equivalence class are collapsed into a single state representing this class. All states in an equivalence class of a bisimulation share the same labeling. So we use this labeling as the labeling of the equivalence class. Similarly, all bisimilar states go with the same rate into another class. This rate becomes the rate between these classes in the quotient. Lastly, we define the value of the initial distribution for an equivalence class as the sum of the values of the initial distribution of its contained states.

**Definition 2.17** (Bisimulation quotient). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $\sim$  a bisimulation on  $\mathcal{M}$ . Then the bisimulation quotient is defined as  $\mathcal{M}_\sim = (S_\sim, R_\sim, L_\sim, \nu_\sim)$  with

- $S_\sim = S/\sim$ ,
- $R_\sim : S_\sim \times S_\sim \rightarrow \mathbb{R}_{\geq 0}$  where  $R_\sim : ([s]_\sim, C) \mapsto R(s, C)$ ,
- $L_\sim : S_\sim \rightarrow AP : [s]_\sim \mapsto L(s)$  and
- $\nu_\sim : S_\sim \rightarrow \mathbb{R}_{\geq 0} : C \mapsto \sum_{s \in C} \nu(s)$ .

## Partition refinement algorithm

The standard algorithm to compute a (maximal) bisimulation given a CTMC is the *partition refinement algorithm*. The idea of this algorithm is to first partition the set of states according to their labeling and exit rates. This initial partition is then refined by successively splitting individual sets  $C$  in the partition in order to make them comply to the condition that  $R(s, D) = R(s', D)$  for  $s, s' \in C$  and for all  $D$  in the partition. In this case  $D$  is called the splitter. This process is repeated until the partition comes to a fixpoint and the aforementioned condition is satisfied by all sets in the partition.

**Input:** CTMC  $\mathcal{M} = (S, R, L, \nu)$   
Let  $R \subseteq S \times S$  be the relation defined by:  $(s, s') \in R \Leftrightarrow L(s) = L(s') \wedge E(s) = E(s')$   
Let  $P = S/R$

do  
     $P' = P$   
    for all  $C \in P'$   
        Let  $R_{split} \subseteq C \times C$  with  $(s, s') \in R_{split} \Leftrightarrow R(s, D) = R(s', D)$  for all  $D \in P$   
         $P = (P \setminus C) \cup C/R_{split}$   
while  $P' \neq P$

**Output:**  $P$

# Chapter 3

## Markovian testing minimization equivalence

This chapter first shortly describes the context in which the original version of Markovian testing equivalence (MTE) was developed. It then presents a version of MTE which has been adapted for CTMCs. As MTE is not suitable for the minimization of arbitrary CTMCs, its underlying concepts are employed to develop Markovian testing minimization equivalence (MTME). Finally this chapter compares the presented equivalences with each other and with bisimulation.

From this point onwards, we will only consider CTMCs with the property that every state has at least one predecessor. This restriction allows for much clearer and compact definitions, as it avoids the special case of states without predecessors. It does, however, not restrict the applicability of the minimization technique built, as an arbitrary CTMC can easily be transformed into one with this property while preserving all interesting properties. The idea is to give each state with no predecessor an artificial one. These newly introduced states in turn have a self-loop and thus a predecessor. As the initial distribution is left unchanged the added states are unreachable. Since all considered properties speak exclusively of the reachable part of a CTMC this transformation preserves these properties.

Formally, we transform the CTMC  $\mathcal{M} = (S, R, L, \nu)$  into the CTMC  $\mathcal{M}_\top$ . Therefore, let  $S_{np} = \{\top_s \mid s \in S, \text{pred}(s) = \emptyset\}$  be the set of additional states and  $\mathcal{M}_\top = (S_\top, R_\top, L_\top, \nu_\top)$  defined as follows:

$$\begin{aligned} \bullet S_\top &= S \cup S_{np} & \bullet L_\top(s) &= \begin{cases} L(s) & s \in S \\ \emptyset & \text{otherwise} \end{cases} \\ \bullet R_\top(s, s') &= \begin{cases} R(s, s') & s, s' \in S \\ 1 & s = \top'_s \\ 1 & s = s', s \in S_{np} \\ 0 & \text{otherwise} \end{cases} & \bullet \nu_\top(s) &= \begin{cases} \nu(s) & s \in S \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

### 3.1 Markovian testing equivalence

Markovian testing equivalence (MTE) was developed by Bernardo in [Ber07a]. We will not present the complete original definition here as it requires a substantial apparatus, which would not be used afterwards anymore. Instead we describe the formalism in which MTE was developed and then directly provide a definition on CTMCs.

#### Sequential Markovian process calculus

The original version of MTE works on the sequential Markovian process calculus (SMPC). In this context, processes are described by process terms. The set of process terms of SMPC is generated by the following syntax:

$$\begin{array}{l}
 P ::= \underline{0} \\
 \quad | \langle a, \lambda \rangle . P \\
 \quad | P + P \\
 \quad | A
 \end{array}$$

The semantics of such a process term are multitransition systems, i.e. transition systems where there can be more than one transition between states. Process terms correspond to states and each transition is labeled with  $\langle a, \lambda \rangle$  where  $a$  is an action and  $\lambda$  the rate of an exponential distribution. The operators are defined as follows:

- $\underline{0}$ : The null-term cannot execute any actions and is thus a state without any outgoing transitions.
- $\langle a, \lambda \rangle . P$ : Sequential composition can execute an  $a$ -action with rate  $\lambda$  and then behaves like  $P$ .
- $P + P$ : Alternative composition behaves like either process term.
- $A$ : A definition of a process term can contain the definition of process constants.  $A$  behaves like the process term it is defined as.

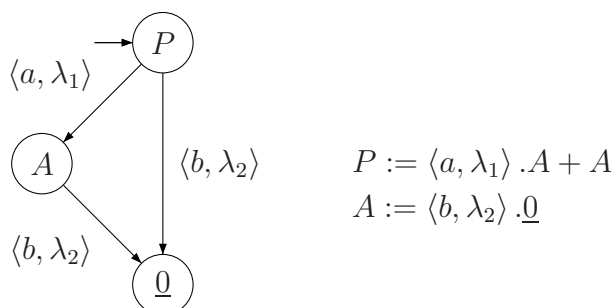


Figure 3.1: SMPC process term and resulting multitransition system.

Figure 3.1 shows a simple process term and its resulting multitransition system:  $P$  can do an  $a$ -action with rate  $\lambda_1$  and then behave like  $A$  or directly behave like  $A$ .  $A$  does a  $b$ -action and then stops to execute any actions since it has reached the null-term.

A multitransition system can be converted into a CTMC by replacing multitransitions with a single transition and removing the action-names. This new transition is labeled with the sum of the rates of the transitions it replaces.

The original version of MTE is an equivalence relation on the set of all process terms. In the underlying semantics on CTMCs this corresponds to an equivalence between CTMCs with a single initial state.

### Markovian testing equivalence on CTMCs

We now directly provide a translation of MTE to CTMCs. It is based on Bernardo's fourth characterization ([Ber07a] Chapter 4.3.4). This characterization is especially suited, since it fully abstracts from testing and replaces it by an approach which compares the probability of performing computations with certain timing characteristics. These computations are described by extended traces. Extended traces contain in addition to the labeling of states also their exit rates. An extended trace  $\sigma \in (\mathbb{R}_0^+ \times 2^{AP})^*$  is written as  $\sigma = (r_1, A_1) \circ (r_2, A_2) \circ \dots \circ (r_n, A_n)$ . We can project the labeling and rate components as  $trace(\sigma) = A_1, A_2, \dots, A_n$  and  $rates(\sigma) = r_1, r_2, \dots, r_n$ . The set of all extended traces is defined as  $\mathcal{ET} = (\mathbb{R}_0^+ \times 2^{AP})^*$ .

For a given extended trace  $\sigma$  we now define the set of cylinder sets which are compatible with  $\sigma$ . The union of these cylinder sets contains exactly those paths which start with a sequence of states that match the exit rates and labeling of the extended trace.

**Definition 3.1.** Let  $\sigma = (r_1, A_1) \circ (r_2, A_2) \circ \dots \circ (r_n, A_n)$  be an extended trace. The set of cylinder sets compatible with  $\sigma$  is defined as

$$C_\sigma = \{Cyl(s_0, \mathbb{R}_0^+, \dots, \mathbb{R}_0^+, s_n) \mid L(s_i) = A_i, E(s_i) = r_i, 1 \leq i \leq n\}.$$

A path is called compatible with  $\sigma$  if it is in the union of the cylinder sets in  $C_\sigma$ . We are now interested in the probability measure of this union of the cylinder sets in  $C_\sigma$ , that is  $Pr_s(C_\sigma) := Pr_s(\bigcup_{Cyl \in C_\sigma} Cyl)$ .

We first show that the union of these cylinder sets is actually measurable: For a finite state space, the cardinality of  $C_\sigma$  is finite. This means  $\bigcup_{Cyl \in C_\sigma} Cyl$  describes a countable union. So it is an element of the  $\sigma$ -algebra and thus also a measurable set.

Furthermore, the cylinder sets contained in  $C_\sigma$  are disjoint. This allows us to express the measure of the union as the sum of the measures of the contained cylinder sets. Consequently,  $Pr_s(\bigcup_{Cyl \in C_\sigma} Cyl) = \sum_{Cyl \in C_\sigma} Pr_s(Cyl)$  for all  $\sigma \in \mathcal{ET}$ .

$Pr_s(C_\sigma)$  gives the probability for paths starting in  $s$  to be compatible to an extended trace  $\sigma$ . This allows to compare states based on this probability. If there exists an extended trace  $\sigma$  such that  $Pr_s(C_\sigma) \neq Pr_t(C_\sigma)$ ,  $s$  and  $t$  can be distinguished using  $\sigma$ . If there exists no such  $\sigma$  these states cannot be distinguished by extended traces. Markovian testing equivalence is defined exactly by this indistinguishability by extended traces.

**Definition 3.2.** Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC. A relation  $\sim_{MTE} \subseteq S \times S$  is called a Markovian testing equivalence (MTE) if for all  $s, s' \in S$

$$s \sim_{MTE} s' \text{ iff}$$

$$\forall \sigma \in \mathcal{ET} . Pr_s(C_\sigma) = Pr_{s'}(C_\sigma)$$

We call states Markovian testing equivalent (mt-equivalent) if MTE puts them into relation. In other words,  $s$  mt-equivalent  $s'$  iff  $Pr_s(C_\sigma) = Pr_{s'}(C_\sigma)$  for all  $\sigma \in \mathcal{ET}$ . MTE is always an equivalence relation. It inherits this property from the equality on real numbers between  $Pr_s(C_\sigma)$  and  $Pr_{s'}(C_\sigma)$  which is also an equivalence relation. Additionally, for any CTMC there exists exactly one MTE. If two states are bisimilar they are also mt-equivalent. In the same fashion the maximal bisimulation is also an MTE. The proof for these results is lengthy and can be found in Subsection 3.3, which is dedicated to the comparison of the presented equivalences.

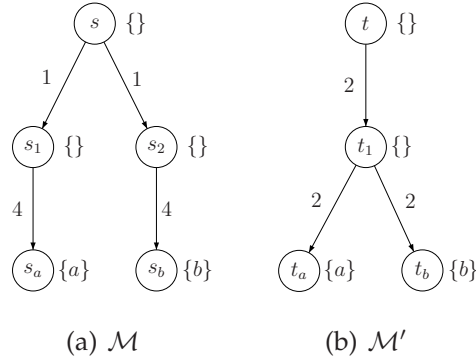


Figure 3.2: Example: Markovian testing equivalence, adapted from [Ber07a].

Figure 3.2 shows the smallest example where MTE is stronger than bisimulation. In this example, apart from reflexivity, only  $s_a$  is bisimilar to  $t_a$  and  $s_b$  to  $t_b$ . So these pairs of states are also mt-equivalent. However, in addition to bisimulation,  $s$  and  $t$  are mt-equivalent.

For  $s$  and  $t$  to be mt-equivalent  $Pr_s(C_\sigma)$  has to be equal to  $Pr_t(C_\sigma)$  for all extended traces  $\sigma$ .  $s$  and  $t$  both have a labeling of  $\{\}$  and an exit rate of 2. This means that all extended traces  $\sigma$  which do not start with  $(2, \{\})$  have  $Pr_s(C_\sigma) = Pr_t(C_\sigma) = 0$ . The same goes for all other extended traces which do not have a compatible path starting in  $s$  or  $t$ .

The remaining extended traces are  $\sigma_a = (2, \{\}) \circ (4, \{\}) \circ (0, \{a\})$ ,  $\sigma_b = (2, \{\}) \circ (4, \{\}) \circ (0, \{b\})$  and their prefixes.

We now have a closer look at  $Pr_s(C_{\sigma_b})$  and  $Pr_t(C_{\sigma_b})$ .  $C_{\sigma_b}$  contains two cylinder sets, which correspond to the time-abstract paths  $s \rightarrow s_2 \rightarrow s_b$  and  $t \rightarrow t_1 \rightarrow t_b$ :

$$C_{\sigma_b} = \{Cyl(s, \mathbb{R}_0^+, s_2, \mathbb{R}_0^+, s_b), Cyl(t, \mathbb{R}_0^+, t_1, \mathbb{R}_0^+, t_b)\}.$$

We can now compute  $Pr_s(C_{\sigma_b}) = Pr_s(Cyl(s, \mathbb{R}_0^+, s_2, \mathbb{R}_0^+, s_b)) + Pr_s(Cyl(t, \mathbb{R}_0^+, t_1, \mathbb{R}_0^+, t_b))$ . The second term is equal to 0 as the cylinder set starts with  $t$  and not  $s$ . This leaves us

with  $Pr_s(C_{\sigma_b}) = Pr_s(Cyl(s, \mathbb{R}_0^+, s_2, \mathbb{R}_0^+, s_b))$ , which is the probability to go, starting in  $s$ , via  $s_2$  to  $s_b$ . Thus  $Pr_s(C_{\sigma_b}) = \frac{R(s, s_2)}{E(s)} \cdot \frac{R(s_2, s_b)}{E(s_2)} = \frac{1}{2} \cdot 1 = \frac{1}{2}$ . In the same way we can compute  $Pr_t(C_{\sigma_b}) = \frac{R(t, s_1)}{E(t)} \cdot \frac{R(s_1, t_b)}{E(t_1)} = \frac{2}{2} \cdot \frac{2}{4} = \frac{1}{2}$ . This shows that indeed  $Pr_s(C_{\sigma_b}) = Pr_t(C_{\sigma_b})$ . The cases of  $C_{\sigma_a}$  and the prefixes of these two extended traces go analogously.

What happens in this example is that both CTMCs can go in two steps from their initial state to a  $b$ -state with probability  $\frac{1}{2}$  and the visited states have the same sequence of labelings and exit rates. This is exactly what the definition of MTE requires. The difference now is the point at which the decision to go to the final  $b$ -state has to be taken.

Seen from the perspective of minimization we can transform  $\mathcal{M}$  into  $\mathcal{M}'$  by merging  $s_1$  and  $s_2$  into  $t_1$  and both CTMCs still exhibit the same behavior with respect to extended traces.

This concept can be generalized, which is depicted in Fig. 3.3. As in the previous example, we have three tiers of states.  $s$  and  $s'$  in the upper tier are mt-equivalent for the right choice of rates. On the middle tier  $s_1, \dots, s_n$  are merged into  $s''$ . The lower tier, consisting of states  $t_1, \dots, t_m$ , is unchanged.

$s$  has  $n$  successors  $s_1, \dots, s_n$ . The rate from  $s$  to  $s_i$  is called  $r_i$ . Each  $s_i$  has successors in  $\{t_1, \dots, t_m\}$ . We call the rate from  $s_i$  to  $t_j$   $r_{i,j}$ . For the sake of simplicity, we consider  $r_{i,j}$  to be 0 if there is no such transition.

We now want to derive under which conditions  $s$  mt-equivalent  $s'$  in Fig. 3.3:

All  $s_i$  and  $s''$  are required to have the same labeling and exit rate, i.e.  $L(s_i) = L(s_j) = L(s'')$  and  $E(s_i) = E(s_j) = E(s'')$  for all  $1 \leq i, j \leq n$ . This is necessary since otherwise these states could be immediately distinguished. For the same reason we require the same for the upper tier, i.e.  $L(s) = L(s')$  and  $E(s) = E(s')$ . The latter means that  $r' = E(s) = \sum_{1 \leq i \leq n} r_i$ .

We want to derive the value of  $r'_j$ : If  $s$  mt-equivalent  $s'$ , the probability  $p_{s,t_j}$  to go from  $s$  to  $t_j$  and the probability  $p_{s',t_j}$  to go from  $s'$  to  $t_j$  have to be the same.  $p_{s,t_j} = p_{s',t_j}$  leads to  $\sum_{1 \leq i \leq n} \frac{r_i}{r'} \cdot \frac{r_{i,j}}{E(s_i)} = \frac{r'_j}{r'} \cdot \frac{r'_j}{E(s'')}$ .  $E(s_i)$  is independent of  $i$ . So we choose  $i$  to be 1 and we set  $E(s'') = E(s_1)$ . So  $E(s_1) \cdot \sum_{1 \leq i \leq n} \frac{r_i}{r'} \cdot r_{i,j} = \frac{r'_j}{E(s_1)}$ . We cancel out  $E(s_1)$  and get the result  $r'_j = \sum_{1 \leq i \leq n} \frac{r_i}{r'} \cdot r_{i,j}$ .

In summary,  $s$  mt-equivalent  $s'$  in Fig. 3.3 iff

- $L(s) = L(s')$  and  $E(s) = E(s')$ ,
- $L(s_i) = L(s_j)$  and  $E(s_i) = E(s_j)$  for all  $1 \leq i, j \leq n$ ,
- $L(s'') = L(s_1)$  and  $E(s'') = E(s_1)$ ,
- $r' = \sum_{1 \leq i \leq n} r_i$  and  $r'_j = \sum_{1 \leq i \leq n} \frac{r_i}{r'} \cdot r_{i,j}$ .

This effectively gives us a rewriting rule for MTE. In order to be used in such a way within a CTMC we also require that  $s_1, \dots, s_n$  do not have any predecessors beside  $s$ . Bernardo shows for non-recursive process terms that a version of this rewriting rule together with rules that axiomatize bisimulation axiomatizes MTE ([Ber07a] Chapter

4.6). Axiomatization in this context means that the rules can be used to transform a process term into any other mt-equivalent process term.

However, these rules cannot be used to build an effective minimization technique

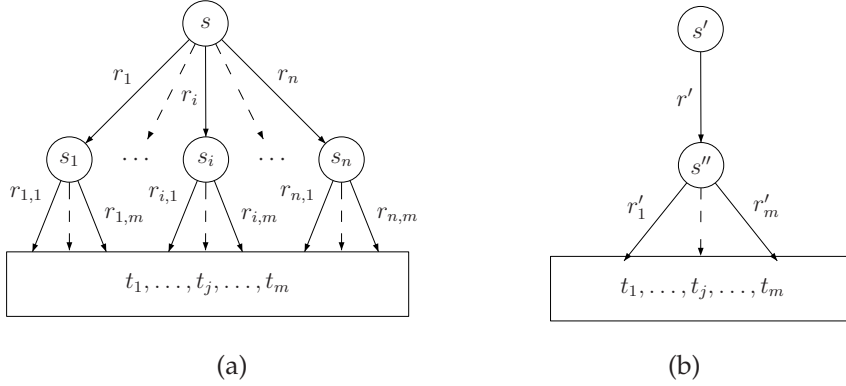


Figure 3.3: Rewriting rule for MTE, adapted from [Ber07a].

on CTMCs. The rules have to be used in both directions, so their application is not trivial. Also they only work for non-recursive process terms. Non-recursive process terms yield directed acyclic graphs as CTMCs. This would restrict the applicability of such a technique also to non-recursive CTMCs.

## 3.2 Markovian testing minimization equivalence

Our goal in this thesis is to build a minimization technique for CTMCs based on MTE. However, when we have a closer look, MTE does not put all the states into relation that can be merged. In Fig. 3.2,  $s$  is mt-equivalent to  $t$  and we can merge  $s_1$  with  $s_2$  in order to transform  $\mathcal{M}$  into  $\mathcal{M}'$ . Despite this,  $s_1$  is not mt-equivalent to  $s_2$  and thus they are not in the same equivalence class. In other words, the quotient of MTE on the state space does not yield a sensible grouping of states which can be extended to a quotient on the CTMC.

The reason is, that the original version of MTE is an equivalence between process terms. Process terms correspond to CTMCs with a single initial state. So the original MTE does not have to explicitly consider states deeper in the CTMC. In the translation to CTMCs we already extend this to an equivalence on the state space. Unfortunately, this did not suffice. As a solution, we propose a new equivalence relation called Markovian testing minimization equivalence (MTME).

### Subclasses

We will describe MTME starting with its building blocks. Figure 3.4 contains a larger example. Here, the union of  $\mathcal{M}$  and  $\mathcal{M}'$  can be minimized into  $\mathcal{M}_{min}$ . Again, the states in the middle tier, i.e.  $s_1, s_2, s'_1$  and  $s'_2$ , can be merged into a single state but are not pairwise mt-equivalent. In our new equivalence, they should be in the same equivalence class, so they can be merged. However, when we have a closer look, we see that,

for example  $s_1$  can reach an  $a$ -state while  $s_2$  and  $s'_2$  can not. Similar examples can be found for all pairs of these states. Bisimulation and MTE compare single states. The trick with MTME is now to consider sets of states instead of single states and compare these. So we group the states into two sets  $SC_1 = \{s_1, s_2\}$  and  $SC_2 = \{s'_1, s'_2\}$ . Now there exists a path from  $SC_1$  and a path from  $SC_2$  to an  $a$ -state and the same for  $b$ - and  $c$ -states. So, to reiterate, the idea is that we do not compare the states of each equivalence class directly with each other, but instead, we group them into subclasses and compare these.

Fortunately, we can avoid an equivalence relation between sets of states, i.e. a relation on  $2^S$ . It turns out that the sets we would like to compare each have a common predecessor. Given an equivalence class, the idea is now to group the states by their common predecessors. We call these subsets of equivalence classes *subclasses*. Formally, subclasses are maximal subsets of an equivalence class, such that all states in such a set have a common predecessor.

**Definition 3.3** (Subclasses). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC,  $\sim \subseteq S \times S$  an equivalence relation and  $C \in S/\sim$ . Then

$$Subc(C, s_0) = post(s_0) \cap C.$$

We call  $Subc(C, s_0)$  the subclass of  $C$  induced by  $s_0$ :

$$Subc(C) = \{Subc(C, s_0) \mid s_0 \in pred(C)\}.$$

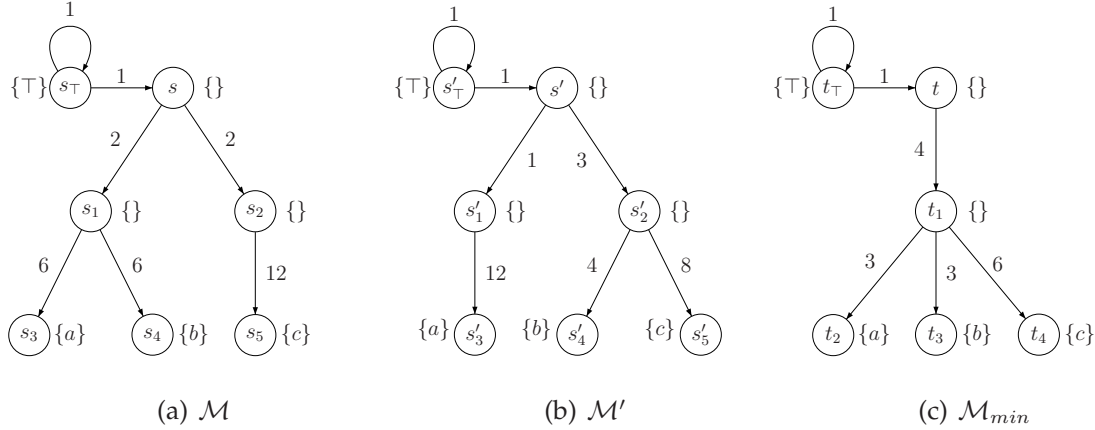


Figure 3.4:  $\mathcal{M}$ ,  $\mathcal{M}'$  and their minimization  $\mathcal{M}_{min}$ .

An example for the construction of subclasses can be seen in Fig. 3.5. The subclass  $C$  consists of  $s_1, s_2$  and  $s_3$ , and  $s_0$  has the three successors  $s_2, s_3, s_4$ . The subclass  $Subc(C, s_0)$  contains now those members of  $C$  which have  $s_0$  as a predecessor, i.e. states  $s_2$  and  $s_3$ .

The subclasses of a class do not in general partition the class. An example can be seen in Fig. 5.1 on page 49<sup>1</sup>.  $C = \{s_3, s_4, s_5, s_6\}$  forms an equivalence class with

<sup>1</sup>The figure is also used as an example for a run of the minimization algorithm for MTME and has thus been placed in the respective chapter for easier reference.

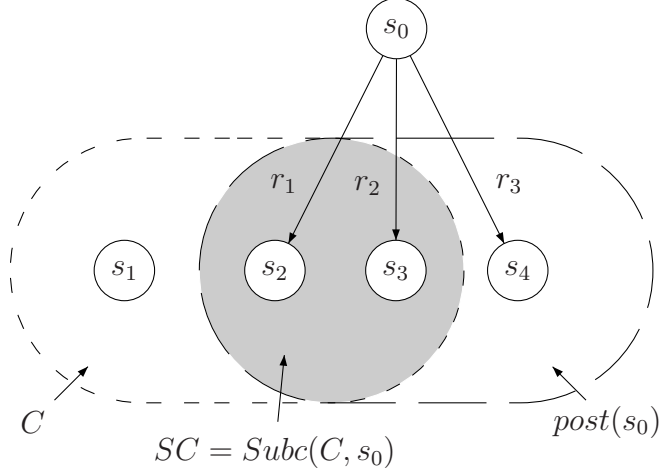


Figure 3.5: Construction of a subclass.

subclasses  $SC_1 = \{s_3, s_4, s_5\}$  induced by  $s_1$  and  $SC_2 = \{s_5, s_6\}$  induced by  $s_2$ .  $SC_1$  and  $SC_2$  overlap since  $s_5$  has two predecessors:  $s_1$  and  $s_2$ .

### Weight function on subclasses

In order to be able to compare subclasses with each other, we have to assign a weight to each of their states. Let  $SC$  be the subclass of  $C$  induced by  $s_0$ . Then the weight assigned to a state  $s \in SC$  will be the conditional probability to go from  $s_0$  to  $s$  given a transition from  $s_0$  to  $SC$ :  $P(s_0 \rightarrow s \mid s_0 \rightarrow SC)$ . Coming back to the example from Fig. 3.4, the subclass  $\{s'_1, s'_2\}$ , which is induced by  $s'$ , would have weight  $\frac{1}{4}$  for state  $s'_1$  and weight  $\frac{3}{4}$  for state  $s'_2$ .

The same subclass could have been induced by two different predecessors, so we have to take  $s_0$  into the signature of *weight*.

**Definition 3.4.** Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC. Then  $weight : S \times S \times 2^S \rightarrow \mathbb{R}_{>0}$  with

$$weight(s_0, s, SC) = \frac{R(s_0, s)}{R(s_0, SC)}$$

We use  $weight(s_0, s, SC)$  exclusively for  $s_0 \in pred(SC)$ , so the case where  $R(s_0, SC)$  becomes 0 does not occur. It is a weight function on the states of  $SC$  normalized to 1.

**Lemma 3.1.** Let  $C \subseteq S$ ,  $s_0 \in pred(C)$ ,  $SC = Subc(C, s_0)$  then  $\sum_{s \in SC} weight(s_0, s, SC) = 1$ .

For the example in Fig. 3.5 we get  $weight(s_0, s_1, SC) = \frac{R(s_0, s_1)}{R(s_0, SC)} = \frac{r_1}{r_1 + r_2}$ . Similarly we obtain  $weight(s_0, s_2, SC) = \frac{r_2}{r_1 + r_2}$ .

## Weighted rate

Given a subclass  $SC = Subc(C, s_0)$  of an equivalence class  $C$  with predecessor  $s_0 \in pred(C)$  and another equivalence class  $D$ , we are now interested in the rate from  $SC$  to  $D$ . We weight this rate by the above defined function. In this way, we get the conditional probability to perform transitions from  $s_0$  via  $C$  to  $D$  given a transition from  $s_0$  to  $C$ .

**Definition 3.5** (Weighted rate). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC. Then  $wrate : S \times 2^S \times 2^S \dashrightarrow \mathbb{R}_{>0}$  with

$$wrate(s_0, C, D) = \sum_{s \in Subc(C, s_0)} weight(s_0, s, Subc(C, s_0)) \cdot R(s, D)$$

We sometimes directly use the subclass  $SC = Subc(C, s_0)$  as argument and write  $wrate(s_0, SC, D)$  instead of  $wrate(s_0, C, D)$ . This is possible, since  $Subc(Subc(C, s_0), s_0) = Subc(C, s_0)$  and consequently  $wrate(s_0, SC, D) = wrate(s_0, C, D)$ .

## Markovian testing minimization equivalence

Now we can define Markovian testing minimization equivalence (MTME). First of all, for two states to be considered equivalent, we require that they have the same labeling and exit rate. This is a natural condition since we want to obtain an equivalence which is stronger than bisimulation.

Apart from that, we compare for each equivalence class its subclasses using  $wrate$ .

**Definition 3.6** (Markovian testing minimization equivalence). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC.

An equivalence relation  $\sim \subseteq S \times S$  is a Markovian testing minimization equivalence (MTME) iff

- for all  $s_1 \sim s_2$  :

$$L(s_1) = L(s_2) \tag{1}$$

$$\text{and } E(s_1) = E(s_2). \tag{2}$$

- for all  $C, D \in S/\sim$  and all  $s_0, s'_0 \in pred(C)$ :

$$wrate(s_0, C, D) = wrate(s'_0, C, D) \tag{3}$$

Consider the CTMC in Fig. 5.1 on page 49 and the equivalence relation  $\sim$  with partition  $\{\{s_0\}, \{s_1\}, \{s_2\}, \{s_3\}, \{s_4, s_5, s_6\}, \{s_a\}, \{s_b\}, \{s_c\}\}$ . This relation is an MTME. It is easy to see that the states in every equivalence class share the same labeling and exit rate. For condition (3), dealing with weighted rates, the only interesting equivalence class is  $C = \{s_4, s_5, s_6\}$ .  $C$  has predecessors  $s_1$  and  $s_2$ . These induce the subclasses  $\{s_4, s_5\}$  and  $\{s_5, s_6\}$ . We can now compute the weighted rates. We choose  $D = \{s_b\}$  as the destination class.  $wrate(s_1, C, D) = weight(s_1, s_4, \{s_4, s_5\}) \cdot R(s_4, s_b) +$

$weight(s_1, s_5, \{s_4, s_5\}) \cdot R(s_5, s_b) = \frac{1}{2} \cdot 2 + \frac{1}{2} \cdot 0 = 1$ . In the same way, we compute  $wrate(s_2, C, D) = 1$ . Continuing this for all  $D \in S/\sim$  we can establish that  $\sim$  is indeed an MTME.

Now let  $\sim_2 \subseteq S \times S$  be induced by  $\{\{s_0\}, \{s_1\}, \{s_2\}, \{s_3\}, \{s_4, s_6\}, \{s_5\}, \{s_a\}, \{s_b\}, \{s_c\}\}$ . This equivalence is an MTME as well. In fact it is also the maximal bisimulation. This shows that MTMEs are not unique, i.e. there can be more than one equivalence relation that is an MTME for any given CTMC. In this example, but not in general,  $\sim$  is strictly coarser than  $\sim_2$ . So in terms of minimization,  $\sim$  is preferable. This motivates the definition of a maximal MTME as one where there exists no coarser MTME.

**Definition 3.7** (Maximal MTME). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $\sim \subseteq S \times S$  a Markovian testing minimization equivalence (MTME) on  $\mathcal{M}$ . If there does not exist an MTME  $\sim'$  on  $\mathcal{M}$  such that  $\sim'$  is coarser than  $\sim$ , then  $\sim$  is called a maximal MTME on  $\mathcal{M}$ .

Given an equivalence relation, any coarser equivalence relation can be obtained by merging equivalence classes. So in a maximal MTME no two (or more) equivalence classes can be merged and still result in an MTME.

### 3.2.1 Quotient under MTME

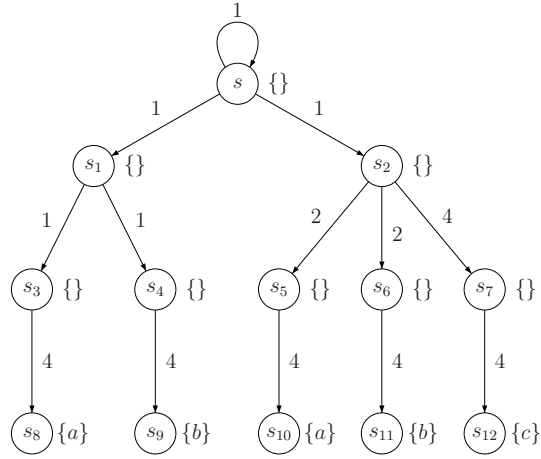
Our next step towards minimization is to define the quotient under MTME.

**Definition 3.8** (Quotient under MTME). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $\sim \subseteq S \times S$  a Markovian testing minimization equivalence (MTME) on  $\mathcal{M}$ . We define the quotient of  $\mathcal{M}$  under  $\sim$  as  $\mathcal{M}/\sim := (S', R', L', \nu')$ , where

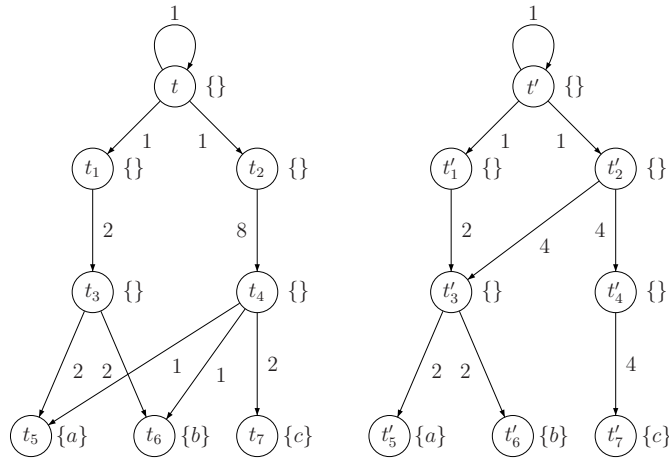
- $S' = S/\sim \subseteq 2^S$ ,
- $R' : S' \times S' \rightarrow \mathbb{R}_{\geq 0}$ ,
- $R' : ([s]_{\sim}, C) \mapsto wrate(s_0, [s]_{\sim}, C)$  where  $s_0 \in pred([s]_{\sim})$ ,
- $L' : S' \rightarrow AP : [s]_{\sim} \mapsto L(s)$  and
- $\nu' : S' \rightarrow \mathbb{R}_{\geq 0} : [s]_{\sim} \mapsto \sum_{s' \in [s]_{\sim}} \nu(s')$ .

The definition of the rate function of the quotient uses a state  $s_0 \in cpred(S_1)$ .  $s_0$  can be chosen arbitrarily, since the definition of MTME ensures that  $wrate(s_0, S_1, S_2)$  is independent of this choice. Similarly the labeling function is well-defined, as states belonging to the same equivalence class of an MTME share the same labeling.

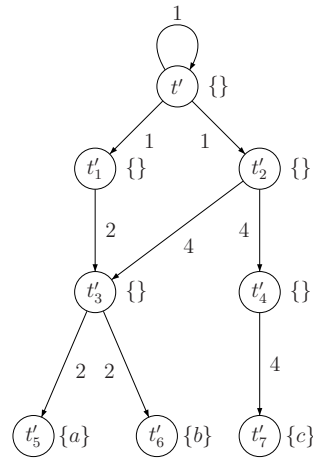
We have already seen that there can exist multiple MTMEs for a given CTMC. An example how this manifests in the quotient can be seen in Fig. 3.6. The original CTMC  $\mathcal{M}_0$  can be minimized in two different ways.  $\mathcal{M}$  is the quotient for the MTME induced by  $\{\{s\}, \{s_1\}, \{s_2\}, \{s_3, s_4\}, \{s_5, s_6, s_7\}, \{s_8, s_{10}\}, \{s_9, s_{11}\}, \{s_{12}\}\}$ .  $\mathcal{M}'$  is obtained as the quotient induced by  $\{\{s\}, \{s_1\}, \{s_2\}, \{s_3, s_4, s_5, s_6\}, \{s_7\}, \{s_8, s_{10}\}, \{s_9, s_{11}\}, \{s_{12}\}\}$ . As an example we will compute for  $\mathcal{M}'$  the rates of the equivalence class  $C = \{s_3, s_4, s_5, s_6\}$ .



(a)  $\mathcal{M}_0$



(b)  $\mathcal{M}$



(c)  $\mathcal{M}'$

Figure 3.6: Different minimizations by maximal MTME.

$C$  has the two predecessors  $s_1$  and  $s_2$ .  $s_1$  induces the subclass  $SC = \{s_3, s_4\}$  and  $s_2$  induces  $SC' = \{s_5, s_6\}$ . Since both subclasses have to have the same weighted rates we can choose one. In  $SC'$  both states are weighted with  $\frac{1}{2}$ . The rate into equivalence class  $\{s_8, s_{10}\}$  is thus computed as  $\frac{1}{2} \cdot R(s_5, \{s_8, s_{10}\}) + \frac{1}{2} \cdot R(s_6, \{s_8, s_{10}\}) = \frac{1}{2} \cdot 4 + \frac{1}{2} \cdot 0 = 2$ . Both quotients cannot be minimized anymore, since their only MTME is the identity relation. In other words, they are maximal MTMEs. This shows that unlike bisimulation there does not exist a unique maximal MTME.

### Repeated Minimization

Let  $\mathcal{M}_q$  be the quotient of a CTMC under a maximal MTME. Then it is still possible that the maximal MTME on  $\mathcal{M}_q$  is not the identity relation. In other words, it can be further minimized using MTME. The reason for this behavior is that MTME does only con-

sider which successors are in the same equivalence classes and does not do the same for predecessors. So if in the quotient the predecessors of two states have been merged this can allow for further minimization. An example can be seen in Fig. 3.7.  $\mathcal{M}'$  is the quotient of  $\mathcal{M}$  under the maximal MTME  $\{\{s_\top\}, \{s\}, \{s_1, s_2\}, \{s_3, s_4\}, \{s_5\}, \{s_6\}, \{s_7\}, \{s_8\}\}$ .  $\mathcal{M}''$  is the quotient under the MTME  $\{\{s'_\top\}, \{s'\}, \{s'_1\}, \{s'_2, s'_3\}, \{s'_4\}, \{s'_5\}, \{s'_6\}\}$  of  $\mathcal{M}'$ . We can see that  $s_3, s_4$  and  $s_5$  can only be merged in two steps since in the first step their predecessors  $s_1$  and  $s_2$  have to be merged.

One of the goals in the construction of MTME was to lend itself to a minimization algorithm by only using conditions which can be verified locally by looking at the neighbors of a state. For the predecessors of an equivalence class it is however important which weight they have. So we would need to know the rates of the predecessor's predecessor. This can be seen in the example:  $s_1$  and  $s_2$  are in the same equivalence class and form a subclass with weights  $\frac{4}{6}$  and  $\frac{2}{6}$ . These weights stem from the rate of their predecessor  $s$  and are incorporated by the quotienting construction into rates one step down. Namely they are incorporated into the rates of the outgoing transitions of  $s'_1$ . Only now we can decide locally for  $s'_2$ , which represents  $\{s_3, s_4\}$ , and  $s'_3$ , which represents  $s_5$ , that these states can be merged and more importantly which are the correct weights.

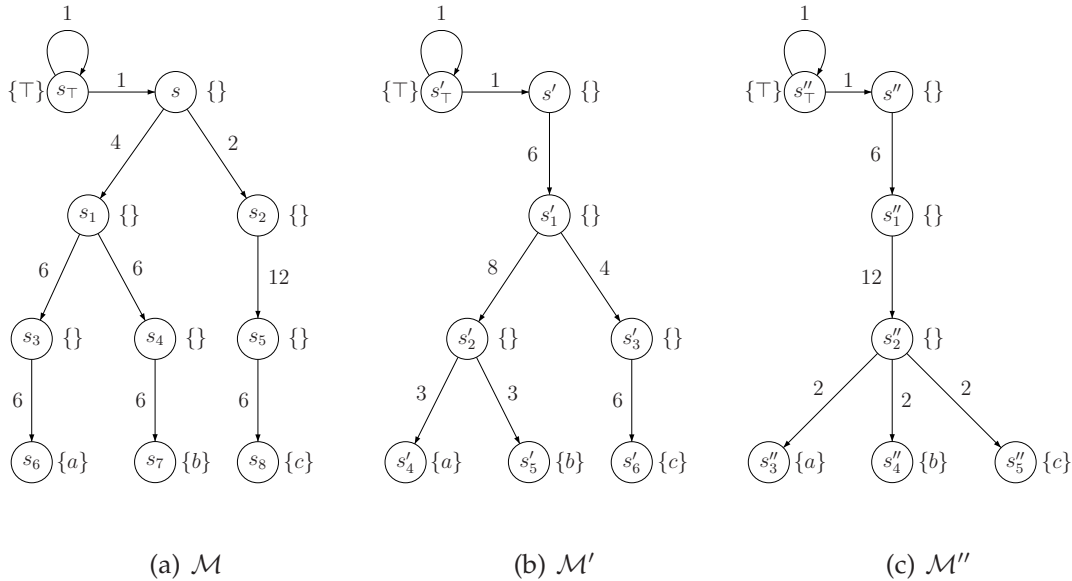


Figure 3.7: Repeated minimization using MTME.

### 3.2.2 Equivalence of CTMCs under MTME

Our next goal is to show that a CTMC and its quotient under MTME are equivalent under MTME. This result is important as it allows us to show that properties which are preserved by MTME are also preserved by the quotient construction.

However, we first have to clearly define under which conditions we consider two

CTMCs "equivalent under MTME". The route taken is using the initial distribution as an anchor point and letting it propagate from there. This does not rule out that unreachable parts of both CTMCs disagree on basic properties. However, this does not concern us, because ultimately we are interested in properties on the reachable part.

### Compatibility between MTME and initial distribution

The minimization potential of MTME stems in part from taking into consideration common predecessors of states in an equivalence class. This leads to a situation where a state that can reach an  $a$ -state and one that can not are merged and this contradiction is resolved only in their common predecessor. In Fig. 3.2  $s_1$  and  $s_2$  are merged into  $t_1$ . Now consider the initial distribution that always starts in  $s_1$ , i.e.  $\nu(s_1) = 1$ . Then in the quotient  $\nu'(t_1) = 1$ . But with these initial distributions the CTMCs have differing properties, since  $t_1$  can reach a  $b$ -state while  $s_1$  can not.

In consequence, the quotient under MTME cannot preserve arbitrary initial distributions in a meaningful way. Thus the following definition describes which initial distributions are not affected by this problem. It can also be read the other way around as a condition an MTME has to fulfill to preserve a given initial distribution. We will also see later that this compatibility is preserved in the quotient.

The idea behind this definition is that the part of an initial distribution which falls onto a subclass mimics the weights in this subclass. This is necessary since in MTME the basic idea about a subclasses is not that the states are equal, but rather we know that, if we are in a subclass, we are in a certain state of it with a certain probability. This probability is exactly the weight. The first condition defines this. The second condition ensures that subclasses which contain initial states do not overlap, as this again can disregard the weights of a class.

**Definition 3.9** (Compatibility between MTME and initial distribution  $\nu$ ). Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC. Further, let  $\sim \subseteq S \times S$  be an MTME.

We say  $\sim$  is compatible with initial distribution  $\nu$  iff

- for all  $C \in S/\sim$ ,  $s_0 \in \text{pred}(C)$ ,  $SC = \text{Subc}(C, s_0)$  and  $s \in SC$ :  
 $\nu(s) = \text{weight}(s_0, s, SC) \cdot \nu(SC)$ .
- for all  $C \in S/\sim$ ,  $SC, SC' \in \text{Subc}(C)$ ,  $SC \neq SC'$ :  
 $\nu(SC) > 0, \nu(SC') > 0 \Rightarrow SC \cap SC' = \emptyset$

Since the identity relation  $\{(s, s) \mid s \in S\}$  is compatible with every initial distribution, there always exists an compatible MTME for every CTMC.

So the question remains if for every initial distribution there also exists a *non-trivial* MTME which is compatible with it. This in fact is not the case: Consider the example in Fig. 3.2. There are only two equivalence relations which fulfill the necessary conditions (1) and (2) of the definition of MTME. These are the trivial equivalence relation with partition  $\sim_1 = \{\{s\}, \{s_1\}, \{s_2\}, \{s_a\}, \{s_b\}\}$  and the equivalence relation with partition  $\sim_2 = \{\{s\}, \{s_1, s_2\}, \{s_a\}, \{s_b\}\}$ . Now consider the initial distribution  $\nu$  with  $\nu(s_1) = \frac{1}{3}$ ,  $\nu(s_2) = \frac{2}{3}$  and 0 otherwise. This initial distribution is not compatible with  $\sim_2$  and thus there does not exist a non-trivial MTME with the desired property.

## Equivalence of CTMCs under MTME

Using the concept of compatibility, it is now possible to define when we consider two CTMCs to be equivalent under MTME. Formally, "equivalent under MTME" is an equivalence relation on the set of all CTMCs.

**Definition 3.10** (Equivalence of CTMCs under MTME). Let  $\mathcal{M}_1 = (S_1, R_1, L_1, \nu_1)$  and  $\mathcal{M}_2 = (S_2, R_2, L_2, \nu_2)$  be CTMCs.

We consider  $\mathcal{M}_1$  and  $\mathcal{M}_2$  equivalent under MTME iff there exists an MTME  $\sim$  for the union CTMC  $\mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2$  such that for all equivalence classes  $C \in (S_1 \cup S_2)/\sim$

$$\nu_1(C \cap S_1) = \nu_2(C \cap S_2)$$

and  $\nu_1, \nu_2$  are compatible with  $\sim$ .

Now we can finally prove that any CTMC and its quotient under MTME are indeed equivalent under MTME.

**Theorem 3.2.** Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $\sim$  be an MTME on  $\mathcal{M}$ . Let  $\nu$  be compatible with  $\sim$ . Let  $\mathcal{M}_\sim = (S_\sim, R_\sim, L_\sim, \nu_\sim)$  be the quotient of  $\mathcal{M}$  under  $\sim$ . Then  $\mathcal{M}$  and  $\mathcal{M}_\sim$  are equivalent under MTME.

*Proof.* We will first construct a new equivalence for the union of  $\mathcal{M}$  and  $\mathcal{M}_\sim$  in the natural way, that is, we take each equivalence class of  $\sim$  and join it with its representation in the quotient: Let  $\sim_q \subseteq (S \cup S_\sim) \times (S \cup S_\sim)$  be the equivalence relation induced by  $\{[s]_\sim \cup \{[s]_\sim\} \mid s \in S\}$ . Our first proof obligation that results from Def. 3.10 is to show that  $\nu(C_q \cap S) = \nu_\sim(C_q \cap S_\sim)$  for all equivalence classes  $C_q \in (S \cup S_\sim)/\sim_q$ .

$$\begin{aligned} \nu(C_q \cap S) &= \nu(C) = \sum_{s \in C} s \text{ for some } C \in S/\sim \\ &= \nu_\sim(C) \text{ by definition of the quotient} \end{aligned}$$

The remaining proof obligation is to show that  $\sim_q$  is an MTME. Conditions (1) and (2), dealing with labeling and exit rate, follow directly from the definition of the quotient. It remains condition (3):

Let  $C, D \in S/\sim_q$ ,  $s_0, s'_0 \in \text{pred}(C)$ ,  $SC = \text{Subc}(C, s_0)$  and  $SC' = \text{Subc}(C, s'_0)$ . We have to cover three cases for  $SC$  and  $SC'$  corresponding to the sets  $\sim_q$  is constructed from. Subclasses are subsets of an equivalence class that share a common predecessor. Since there are no transitions between  $S$  and  $S_\sim$ ,  $SC$  and  $SC'$  are either a subset of  $S$  or a subset of  $S/\sim$ .

Case 1:  $SC \subseteq S, SC' \subseteq S$

We have to show that  $\text{wrate}(s_0, SC, D) = \text{wrate}(s'_0, SC', D)$ . Since  $\sim$  is an MTME, this immediately follows.

Case 2:  $SC \subseteq S/\sim, SC' \subseteq S/\sim$

By the construction of  $\sim_q$ , it holds that  $C = [s]_\sim \cup \{[s]_\sim\}$  for some  $s \in S$ . So in this case  $SC = \{[s]_\sim\} = SC'$  for some  $s \in S$ .

Case 3:  $SC \subseteq S, SC' \subseteq S/\sim$  or vice versa

Without loss of generality, let  $SC \subseteq S$ ,  $SC' \subseteq S/\sim$ . We have to show that  $wrate(s_0, C, D) = wrate(s'_0, C, D)$ .

There exists an  $s \in S$  such that  $C = [s]_{\sim} \cup \{[s]_{\sim}\}$  and  $SC' = \{[s]_{\sim}\}$ .

$$wrate(s'_0, C, D) = wrate(s'_0, [s]_{\sim} \cup \{[s]_{\sim}\}, D)$$

By the definition of  $wrate$ , this term is equal to

$$\sum_{s' \in \{[s]_{\sim}\}} weight(s'_0, s', \{[s]_{\sim}\}) \cdot R(s', D) = weight(s'_0, [s]_{\sim}, \{[s]_{\sim}\}) \cdot R([s]_{\sim}, D) = R([s]_{\sim}, D).$$

We use the definition of the quotient under MTME and get

$$R([s]_{\sim}, D) = wrate(s_0, SC, D).$$

From cases 1 to 3 we can conclude that  $\sim_q$  also fulfills condition (3) of the definition of MTME and thus is an MTME.  $\square$

### 3.3 Comparison of equivalence relations

#### Comparison between MTE and bisimulation

Bernardo shows for the original version of MTE that bisimulation of SMPC terms implies MTE ([Ber07a]). This is consistent with the case on CTMCs.

**Theorem 3.3.** *Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC,  $s, s' \in S$  with  $s$  bisimilar  $s'$ . Then  $s$  mt-equivalent  $s'$ .*

*Proof.* We have to show that  $Pr_s(C_\sigma) = Pr_{s'}(C_\sigma)$  for all extended traces  $\sigma \in \mathcal{ET}$ .

Let  $\sim_{bis}$  be the maximal bisimulation on  $\mathcal{M}$ . The proof is by induction over the length of  $\sigma \in \mathcal{ET}$  where  $\sigma = (r_1, A_1) \circ (r_2, A_2) \circ \dots \circ (r_n, A_n)$ .

**Induction Base:**  $n = 1$

$Pr_s(C_{(r_1, A_1)}) = 1$  if  $E(s) = r_1$  and  $L(s) = A_1$  and 0 otherwise. Since  $s$  and  $s'$  are bisimilar, they also share the same labeling and exit rate. So  $Pr_{s'}(C_{(r_1, A_1)}) = Pr_s(C_{(r_1, A_1)})$ .

**Induction Step:**  $(n - 1) \rightarrow n$

We consider two cases:

**Case 1:**  $E(s) \neq r_1$  or  $L(s) \neq A_1$ :

Then also  $E(s') \neq r_1$  or  $L(s') \neq A_1$  since  $s$  and  $s'$  are bisimilar and no path in  $C_\sigma$  starts with  $s$  or  $s'$ . So  $Pr_s(C_\sigma) = Pr_{s'}(C_\sigma) = 0$ .

**Case 2:**  $E(s) = r_1$  and  $L(s) = A_1$ :

Let  $\sigma = (r_1, A_1) \circ (r_2, A_2) \circ \dots \circ (r_n, A_n)$ . Then

$$Pr_s(C_\sigma) = \sum_{Cyl \in C_\sigma} Pr_s(Cyl).$$

Let  $\sigma = (r_1, A_1) \circ \sigma'$ . By definition of  $Pr_s$  this is equal to

$$= \sum_{Cyl' \in C_{\sigma'}} \sum_{s_1 \in S} \frac{R(s, s_1)}{E(s)} \cdot Pr_{s_1}(Cyl').$$

We now split the sum over  $S$  by equivalence classes.

$$= \sum_{Cyl' \in C_{\sigma'}} \sum_{C \in S / \sim_{bis}} \sum_{s_1 \in C} \frac{R(s, s_1)}{E(s)} \cdot Pr_{s_1}(Cyl')$$

$E(s)$  is independent of  $s_1$  and can be pulled to the front of the sum.

$$= \frac{1}{E(s)} \sum_{Cyl' \in C_{\sigma'}} \sum_{C \in S / \sim_{bis}} \sum_{s_1 \in C} R(s, s_1) \cdot Pr_{s_1}(Cyl')$$

We can now use the induction hypothesis.  $Cyl'$  has length  $n - 1$  and so for all  $s_1, s'_1 \in S$  with  $s_1$  bisimilar  $s'_1$  it holds that  $Pr_{s_1}(Cyl') = Pr_{s'_1}(Cyl')$ . This means that  $Pr_{s_1}(Cyl')$  is independent from the choice of  $s_1$  within  $C$ . So let  $s'_1 \in C$ .

$$= \frac{1}{E(s)} \sum_{Cyl' \in C_{\sigma'}} \sum_{C \in S / \sim_{bis}} Pr_{s'_1}(Cyl') \cdot \sum_{s_1 \in C} R(s, s_1)$$

We use the definition of  $R(s, C)$ .

$$= \frac{1}{E(s)} \sum_{Cyl' \in C_{\sigma'}} \sum_{C \in S / \sim_{bis}} Pr_{s'_1}(Cyl') \cdot R(s, C)$$

Since  $s$  bisimilar  $s'$  it holds that  $R(s', C) = R(s, C)$ .

$$= \frac{1}{E(s')} \sum_{Cyl' \in C_{\sigma'}} \sum_{C \in S / \sim_{bis}} Pr_{s'_1}(Cyl') \cdot R(s', C) = Pr'_{s'}(C_{\sigma'})$$

This concludes the induction step and together with the induction base we have shown the result. □

This result can also be expanded to the equivalence relations themselves in the following way.

**Corollary 3.4.** *Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $\sim$  the maximal bisimulation on  $\mathcal{M}$ . Then  $\sim$  is also an MTE.*

Only the maximal bisimulation is also an MTE. For any other bisimulation there exist  $s, s' \in S$  where these two states are bisimilar but not in relation. But we have just seen that these are also mt-equivalent.

We have already seen in Fig. 3.2 that there exists an MTE which is not a bisimulation. So bisimulation and MTE do not define the same equivalence.

### Comparison between MTME and bisimulation

One of the goals in the construction of MTME was to have at least the same minimization potential as bisimulation. This is shown in the form that every bisimulation is also an MTME. Also every bisimulation interpreted as an MTME can be used to construct the quotient under MTME and this quotient is identical to the bisimulation quotient modulo naming of states.

**Lemma 3.5.** Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $\sim_{bis} \subseteq S \times S$  a bisimulation.

1. The bisimulation  $\sim_{bis}$  is an MTME.

2. The bisimulation quotient and the quotient under MTME  $\sim_{bis}$  are isomorphic.

*Proof.* (1) Let  $\sim_{bis}$  be a bisimulation. Then we have to prove for all  $s_1 \sim_{bis} s_2$ :

- $L(s_1) = L(s_2)$ , which follows directly by definition of bisimulation.
- $E(s_1) = E(s_2)$ :

$$E(s_1) = \sum_{s' \in S} R(s_1, s') = \sum_{C \in S/\sim_{bis}} \sum_{s' \in C} R(s_1, s') = \sum_{C \in S/\sim_{bis}} R(s_1, C)$$

$R(s_1, C) = R(s_2, C)$  by definition of bisimulation.

$$\sum_{C \in S/\sim_{bis}} R(s_1, C) = \sum_{C \in S/\sim_{bis}} R(s_2, C) = E(s_2)$$

Additionally, we have to prove for all  $C, D \in S/\sim_{bis}$ ,  $s_0, s'_0 \in \text{pred}(C)$

$$\text{wrate}(s_0, C, D) = \text{wrate}(s'_0, C, D)$$

Let  $s_1, s_2 \in C$ . Since  $C \in S/\sim_{bis}$  we know that  $s_1 \sim_{bis} s_2$ . So by definition of bisimulation  $R(s_1, D) = R(s_2, D)$  for all  $s_1, s_2 \in C$  (\*).

Let  $SC = \text{Subc}(C, s_0)$  and  $SC' = \text{Subc}(C, s'_0)$ . Then for all  $s^* \in C$

$$\begin{aligned} \text{wrate}(s_0, C, D) &= \sum_{s \in SC} \frac{R(s_0, s)}{R(s_0, SC)} \cdot R(s, D) \stackrel{(*)}{=} R(s^*, D) \cdot \sum_{s \in SC} \frac{R(s_0, s)}{R(s_0, SC)} \\ &= R(s^*, D) \cdot \frac{\sum_{s \in SC} R(s_0, s)}{R(s_0, SC)} = R(s^*, D) \\ &= \sum_{s' \in SC'} \frac{R(s'_0, s')}{R(s'_0, SC')} \cdot R(s', D) = \text{wrate}(s'_0, C, D) \end{aligned}$$

In conclusion  $\sim_{bis}$  fulfills all conditions of MTME, so every bisimulation is an MTME.

(2): Let  $\sim_{bis}$  be a bisimulation. It follows by (1) that  $\sim_{bis}$  is also an MTME. Our proof obligation is to show that the components of the bisimulation quotient and the quotient under MTME are equal. By definition, this is already the case for state space, labeling function and initial distribution. It remains to also prove equality of the rate functions.

The rate function of the quotient under MTME is defined as  $R'([s]_{\sim}, D) = \text{wrate}(s_0, [s]_{\sim}, D)$  where  $s_0 \in \text{cpred}([s]_{\sim})$  and the one of the bisimulation quotient as  $R_{\sim}([s]_{\sim}, D) = R(s, D)$ . We have already seen in the proof of (1) that  $\text{wrate}(s_0, [s]_{\sim}, D) = R(s, D)$ , so both rate functions are equal.

Thus both quotients are identical up to isomorphism. □

Again Fig. 3.2 serves as a counterexample that the reverse is not true. Consider the MTME on  $\mathcal{M}$  with partition  $\{\{s\}, \{s_1, s_2\}, \{s_a\}, \{s_b\}\}$ .  $s_1$  and  $s_2$  are not bisimilar, so it is not a bisimulation.

## Comparison between MTE and MTME

MTME, although being an equivalence relation on the state space, does not compare states directly. The comparison is done between subclasses, i.e. sets of states, and each state in a subclass is weighted. MTE however compares states directly since it originally is an equivalence which compares CTMCs with a single initial state. So in order to connect MTE with MTME, we examine subclasses which are singleton sets, i.e. of the form  $SC = \{s\}$ . In such a subclass  $\{s\}$  the weight of  $s$  is 1 and thus it matches MTE which compares single states.

It turns out that for two singleton subclasses belonging to the same equivalence class, their contained states are mt-equivalent.

**Theorem 3.6.** *Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC,  $\sim$  be an MTME and  $C \in S/\sim$  an equivalence class of  $\sim$ . For all  $\{s\}, \{s'\} \in \text{Subc}(C)$  it holds  $s$  mt-equivalent  $s'$ .*

This theorem is a special case of theorem 4.1, which connects MTME with extended traces. A proof can be found there.

The reverse does not hold. If  $s$  mt-equivalent  $s'$  there might not exist an MTME which puts these states in the same equivalence class. We have already seen an example for this in Fig. 3.6 where no MTME exists which puts  $t$  and  $t'$  in relation in the union CTMC of  $\mathcal{M}$  and  $\mathcal{M}'$  although  $t$  mt-equivalent  $t'$ .

We now want to analyze the relation between MTE and MTME at the conceptual level. We use the most basic example for MTE which can be found in Fig. 3.2. Here,  $s$  mt-equivalent  $t$  as the extended traces  $\sigma_a = (2, \{\}) \circ (4, \{\}) \circ (0, \{a\})$  and  $\sigma_b = (2, \{\}) \circ (4, \{\}) \circ (0, \{b\})$  both encountered with probability  $\frac{1}{2}$ . Their successors are implicitly weighted with the same  $\frac{1}{2}$ . This is mimiced in MTME where  $s$  and  $t$  induce a subclass with these weights. However, MTME can not perfectly emulate the expressiveness of extended traces and thus cannot completely mimic MTE.

## Verification complexity

Another important property of equivalence relations is their verification complexity. Verification complexity is the complexity of the problem to decide, given a candidate equivalence relation, if this candidate is indeed, for example, an MTE.

According to [Ber07b] (Section 5.8), the verification complexity of MTE is  $O(n^5)$  and bisimulation can be verified in  $O(m \cdot \log n)$ , where  $n$  is the number of states and  $m$  the number of transitions of the CTMC.

The following straightforward algorithm for the verification of MTME shows that the verification complexity of MTME is at most  $O(n^3)$ . It checks the conditions of the MTME definition step by step :

Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $\sim$  an equivalence relation.

- $s \sim s' \Rightarrow E(s) = E(s')$  and  $L(s) = L(s')$ : This can be checked in  $O(n)$  by iterating through all  $s \in S$ .
- for all  $C, D \in S/\sim$  and all  $s_0, s'_0 \in \text{pred}(C) : \text{wrate}(s_0, C, D) = \text{wrate}(s'_0, C, D)$ : We loop through all  $C, D \in S/\sim$ . For every  $s_0 \in \text{pred}(C)$  we compute  $\text{wrate}(s_0, C, D)$

and compare it to previous values for this combination of  $C$  and  $D$ . If they do not match we have shown that  $\sim$  is not an MTME.  $wrate(s_0, C, D)$  can be computed in  $O(|C|)$ . The computation of  $wrate$  and the loop for  $C$  have a combined complexity of  $O(n)$  since  $\sum_{C \in S/\sim} |C| = |S| = n$ . The loops for  $D$  and  $s_0$  have each complexity  $O(n)$ , which leads to a complexity of  $O(n^3)$  for this part.

The second part dominates the complexity of the algorithm and thus it has a complexity of  $O(n^3)$ . So with the construction of MTME we could also reduce the verification complexity compared to MTE.

### Summary

Table 3.1 summarizes the properties of the equivalence relations. The first column shows if for each CTMC there exists exactly one relation which fulfills its properties. The second column contains if there exists a unique maximal relation of this type.

	unique	maximal relation	verification complexity
Bisimulation	no	yes	$O(m \cdot \log n)$
Maximal bisimulation	yes	yes	$O(m \cdot \log n)$
MTE	yes	yes	$O(n^5)$
MTME	no	no	$O(n^3)$

Table 3.1: Summarized properties of equivalence relations.



# Chapter 4

## Preserved properties

This chapter investigates which properties are preserved by MTE and MTME. First, it analyzes the quotient under MTME in terms of which properties are not preserved. It then shows in several steps that time-bounded reachability and time-bounded until formulae are preserved.

### 4.1 Properties which are not preserved

It has been shown by Desharnais and Panangaden in [DP03] that CSL characterizes bisimulation, i.e.  $s$  and  $s'$  are bisimilar if and only if they satisfy the same CSL formulae. Since bisimulation is contained in MTME and they are not identical, there have to be CSL formulae which are not preserved by MTME.

For a first example of properties which are not preserved we ignore the timed and probabilistic nature of CTMCs and argue on the level of the underlying transition system.

The additional minimization over bisimulation can be achieved because MTME modifies the branching behavior. In Fig. 4.1, which is a slight variant of the minimal example distinguishing MTE from bisimulation, the decision to eventually go to an  $a$ - or  $b$ -state is taken in the initial state  $s$  for  $\mathcal{M}$ . In  $\mathcal{M}'$  this decision is postponed by one step, which allows to replace  $s_1$  and  $s_2$  by a single state  $t_1$ . But this means that there exists the  $c$ -state  $t_1$  which is able to go to both an  $a$ - or a  $b$ -state. In  $\mathcal{M}$  there exists no such  $c$ -state, since the decision has already been taken one step earlier. So  $\mathcal{M}$  and  $\mathcal{M}'$  can be distinguished by the CSL formula  $\mathcal{P}_{>0}(\diamond(c \wedge \mathcal{P}_{>0}(\diamond a) \wedge \mathcal{P}_{>0}(\diamond b)))$ . The branching comes into play since the  $\diamond$ -operators are nested. With  $\mathcal{P}_{>0}(\diamond c)$  we specify a state which is not an initial state. In contrast to the initial state, this state can be distinguished from its counterpart(s) in the other CTMC by  $\mathcal{P}_{>0}(\diamond a) \wedge \mathcal{P}_{>0}(\diamond b)$ . So for properties which are preserved we have to examine path properties.

### 4.2 Preservation of time-bounded reachability

This section shows that time-bounded reachability and time-bounded until are preserved by MTME. This is done in three steps. In a first step, we show that subclasses induced by MTME are indistinguishable by extended traces. This is the general case

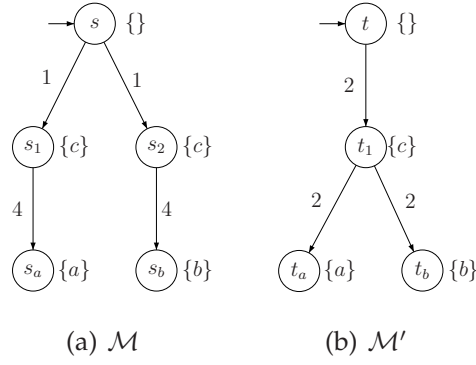


Figure 4.1: Example: Modification of branching behavior by MTME.

of Theorem 3.6, which connects MTME and MTE. Then we show that two states which are in such a way indistinguishable satisfy the same time-bounded until formulae. In a last step we add initial distributions to the picture and show that time-bounded until is preserved between CTMCs that are equivalent under MTME.

### Subclasses of MTME are indistinguishable by extended traces

We have already seen in Theorem 3.6, but not yet proven, that two subclasses of cardinality 1 are indistinguishable by extended traces, i.e. mt-equivalent. This can be extended to all subclasses by weighting the states. The weight of  $s$  in subclass  $SC$  will be the one we already know from the definition of MTME as  $weight(s_0, s, SC)$ , where  $s_0$  induces  $SC$ .

**Theorem 4.1.** *Let  $\sim$  be an MTME over a CTMC  $\mathcal{M} = (S, R, L, \nu)$  and  $C \in S/\sim$ . For all  $\sigma \in \mathcal{ET}$ ,  $s_0, s'_0 \in pred(C)$ ,  $SC = Subc(C, s_0)$ ,  $SC' = Subc(C, s'_0)$*

$$\sum_{s_1 \in SC} weight(s_0, s_1, SC) \cdot Pr_{s_1}(C_\sigma) = \sum_{s'_1 \in SC'} weight(s'_0, s'_1, SC') \cdot Pr_{s'_1}(C_\sigma).$$

*Proof.* We will do an induction over the length of  $\sigma$ .

**Induction Base:**  $n = 0$

Let  $C \in S/\sim$ ,  $\sigma_0 = \varepsilon \in \mathcal{ET}$  and  $SC, SC' \in Subc(C)$ , then

$$\sum_{s_1 \in SC} weight(s_0, s_1, SC) \cdot Pr_{s_1}(C_{\sigma_0}) = \sum_{s_1 \in SC} weight(s_0, s_1, SC) \cdot Pr_{s_1}(\{Cyl(\varepsilon)\}).$$

By definition of  $Pr_s$ :

$$= \sum_{s_1 \in SC} weight(s_0, s_1, SC) \cdot Pr_{s_1}(Cyl(\varepsilon))$$

Since the weights are normalized to 1

$$= 1 \cdot 1$$

By doing the same in reverse, we obtain

$$= \sum_{s'_1 \in SC} \text{weight}(s'_0, s'_1, SC') \cdot Pr_{s'_1}(C_{\sigma_0}).$$

**Induction Hypothesis:**

For  $n \in \mathbb{N}_0$  and for all  $\sigma \in (\mathbb{R}_0^+ \times 2^{AP})^n$ ,  $s_0, s'_0 \in \text{pred}(C)$ ,  $SC = \text{Subc}(C, s_0)$ ,  $SC' = \text{Subc}(C, s'_0)$

$$\sum_{s_1 \in SC} \text{weight}(s_0, s_1, SC) \cdot Pr_{s_1}(C_{\sigma}) = \sum_{s'_1 \in SC'} \text{weight}(s'_0, s'_1, SC') \cdot Pr_{s'_1}(C_{\sigma}).$$

**Induction step:**  $n \rightarrow n + 1$

Let  $C \in S/\sim$ ,  $\sigma_{n+1} = (r_1, A_1) \circ \dots \circ (r_{n+1}, A_{n+1}) \in (\mathbb{R}_0^+ \times 2^{AP})^{n+1}$ ,  $SC, SC' \in \text{Subc}(C)$ .

**Case 1:**  $E(s_1) \neq r_1$  or  $L(s_1) \neq A_1$

This means that  $Cyl(s_1) \cap C_{\sigma_{n+1}} = \emptyset$ , since all paths described by  $C_{\sigma_{n+1}}$  start with a state  $s$  with  $E(s) = r_1$  and  $L(s) = A_1$ . Since  $s_1 \sim s'_1$ ,  $s'_1$  has by definition of MTME the same labeling and exit rate, and so the same argument applies to  $s'_1$ .

Consequently  $Pr_{s_1}(C_{\sigma_{n+1}}) = 0 = Pr_{s'_1}(C_{\sigma_{n+1}})$  and

$$\sum_{s_1 \in SC} \text{weight}(s_0, s_1, SC) \cdot Pr_{s_1}(C_{\sigma_{n+1}}) = 0 = \sum_{s_1 \in SC'} \text{weight}(s_0, s_1, SC) \cdot Pr_{s_1}(C_{\sigma_{n+1}}).$$

**Case 2:**  $E(s_1) = r_1$  and  $L(s_1) = A_1$

$$\sum_{s_1 \in SC} \text{weight}(s_0, s_1, SC) \cdot Pr_{s_1}(C_{\sigma_{n+1}})$$

Let  $\sigma_{n+1} = (r_1, A_1) \circ \sigma_n$  and apply the definition of  $Pr_s$ .

$$= \sum_{s_1 \in SC} \text{weight}(s_0, s_1, SC) \cdot \sum_{s_2 \in S} \frac{R(s_1, s_2)}{E(s_1)} \cdot Pr_{s_2}(C_{\sigma_n})$$

We split the sum over  $s_2 \in S$  by the partition  $S/\sim$ ,

$$= \sum_{s_1 \in SC} \text{weight}(s_0, s_1, SC) \cdot \sum_{D \in S/\sim} \sum_{s_2 \in D} \frac{R(s_1, s_2)}{E(s_1)} \cdot Pr_{s_2}(C_{\sigma_n})$$

$$= \sum_{D \in S/\sim} \sum_{s_1 \in SC} \text{weight}(s_0, s_1, SC) \cdot \sum_{s_2 \in D} \frac{R(s_1, s_2)}{E(s_1)} \cdot Pr_{s_2}(C_{\sigma_n})$$

multiply by  $\frac{R(s_1, D)}{R(s_1, D)}$  and

$$= \sum_{D \in S/\sim} \sum_{s_1 \in SC} \text{weight}(s_0, s_1, SC) \cdot \sum_{s_2 \in D} \frac{R(s_1, D)}{R(s_1, D)} \cdot \frac{R(s_1, s_2)}{E(s_1)} \cdot Pr_{s_2}(C_{\sigma_n})$$

rearrange.

$$= \sum_{D \in S/\sim} \sum_{s_1 \in SC} \text{weight}(s_0, s_1, SC) \cdot R(s_1, D) \cdot \sum_{s_2 \in D} \cdot \frac{1}{E(s_1)} \cdot \frac{R(s_1, s_2)}{R(s_1, D)} \cdot Pr_{s_2}(C_{\sigma_n})$$

Now observe that  $E(s)$  is by definition of  $\sim$  constant for all  $s \in C$  with  $C \in S/\sim$ . Thus let  $out_C = E(s_1)$  and rearrange.

$$\begin{aligned} &= \sum_{D \in S/\sim} \sum_{s_1 \in SC} \text{weight}(s_0, s_1, SC) \cdot R(s_1, D) \cdot \sum_{s_2 \in D} \cdot \frac{1}{out_C} \cdot \frac{R(s_1, s_2)}{R(s_1, D)} \cdot Pr_{s_2}(C_{\sigma_n}) \\ &= \frac{1}{out_C} \sum_{D \in S/\sim} \sum_{s_1 \in SC} \text{weight}(s_0, s_1, SC) \cdot R(s_1, D) \cdot \sum_{s_2 \in D} \frac{R(s_1, s_2)}{R(s_1, D)} \cdot Pr_{s_2}(C_{\sigma_n}) \end{aligned}$$

From the induction hypothesis follows that the term  $\sum_{s_2 \in D} \frac{R(s_1, s_2)}{R(s_1, D)} \cdot Pr_{s_2}(C_{\sigma_n})$  is independent from the choice of  $s_1$ . This allows us to apply the definition of  $\sim$  to the term  $\sum_{s_1 \in SC} \text{weight}(s_0, s_1, SC) \cdot R(s_1, D)$  and switch from  $SC$  to  $SC'$ .

$$= \frac{1}{out_C} \sum_{D \in S/\sim} \sum_{s_1 \in SC'} \text{weight}(s_0, s_1, SC) \cdot R(s_1, D) \cdot \sum_{s_2 \in D} \frac{R(s_1, s_2)}{R(s_1, D)} \cdot Pr_{s_2}(C_{\sigma_n})$$

By doing the reverse we finally get to

$$\sum_{s_1 \in SC'} \text{weight}(s_0, s_1, SC) \cdot Pr_{s_1}(C_{\sigma_{n+1}}).$$

□

### Indistinguishability implies time-bounded reachability

We want to show that states which are indistinguishable by extended traces also satisfy the same time-bounded reachability properties. The premise ties in with Theorem 4.1, which will be used in the proof, although deeper in the CTMC. This theorem allows us to study the proof for a simpler case first, before going over to the general case of MTME.

**Theorem 4.2.** *Let  $\mathcal{M} = (S, R, L, \nu)$  and  $s, s' \in S$  such that for all  $\sigma \in \mathcal{ET}$  holds  $Pr_s(C_\sigma) = Pr_{s'}(C_\sigma)$ . Then for all propositional formulae  $\varphi$  over  $AP$*

$$Pr_s(\diamond_{\leq t} \varphi) = Pr_{s'}(\diamond_{\leq t} \varphi).$$

Before we come to the actual proof of this theorem (see page 40), we present notations and a lemma to aid in the proof.

We start with a notation:  $\mathcal{ET}_{\mathcal{M}, \neg\varphi, k}$  defines a subset of the set of extended traces. It restricts the exit rates to those which actually occur in the CTMC, restricts the atomic propositions to those satisfying  $\neg\varphi$  and traces of length  $k$ . The restriction on the exit rates ensures that  $\mathcal{ET}_{\mathcal{M}, \neg\varphi, k}$  is countable.

**Definition 4.1.** Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC,  $\varphi$  a propositional logic formula over  $AP$  and  $k \in \mathbb{N}_0$  then

$$\mathcal{ET}_{\mathcal{M}, \neg\varphi, k} = (\{E(s) \mid s \in S\} \times \{A \in 2^{AP} \mid A \models \neg\varphi\})^k.$$

$TAPaths(\sigma)$  contains all sequences of states which are compatible with  $\sigma$ . In other words, it projects out the time-abstract paths contained in the union of  $C_\sigma$ .

**Definition 4.2.** Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC and  $\sigma \in \mathcal{ET}$  be an extended trace. Then the set of time-abstract paths compatible to  $\sigma$  is defined as

$$TAPaths(\sigma) = \{(s_0, \dots, s_k) \in S^m \mid Cyl(s_0, \mathbb{R}_0^+, s_1, \dots, \mathbb{R}_0^+, s_k) \in C_\sigma\}.$$

We are now interested in the probability to visit a sequence of states with known exit rates faster than a given time threshold. This can be expressed by the sum of the exponential distributions associated with the states.

The sum of  $n$  exponentially distributed random variables  $X_1, \dots, X_n$  is solely determined by the rate parameters  $r_1, \dots, r_n$  of these exponential distributions. Consequently, also the probability for this sum to be under a certain threshold is determined by these rate parameters.  $PrExpSum_{\leq t}(r_1, \dots, r_n)$  serves as a notation for this probability.

**Definition 4.3.** Let  $1 \leq i \leq n$  and  $X_i$  be an exponentially distributed random variable with rate parameter  $r_i$ . Then we define

$$PrExpSum_{\leq t}(r_1, \dots, r_n) = Prob\left(\sum_{i=1}^n X_i \leq t\right).$$

We are further interested in the probability to visit a given sequence of states within a time-bound. The following lemma stems from the observation that the probability to reach the  $k$ -th state below a given time-bound and the probability to visit a given sequence of states are stochastically independent.

**Lemma 4.3.** Let  $s_0, \dots, s_k \in S$  and  $t \in \mathbb{R}^+$ . Then

$$\begin{aligned} & Pr_{s_0}(\{\pi \in Cyl(s_0, \mathbb{R}_0^+, s_1, \dots, \mathbb{R}_0^+, s_k) \mid \sum_{i=0}^k t_i \leq t, \pi = s_0 \xrightarrow{t_0} s_1 \dots s_k \xrightarrow{t_k} \dots\}) \\ &= Pr_{s_0}(Cyl(s_0, \mathbb{R}_0^+, s_1, \dots, \mathbb{R}_0^+, s_k)) \cdot PrExpSum_{\leq t}(E(s_0), \dots, E(s_{k-1})). \end{aligned}$$

*Proof.* Let  $s_0, \dots, s_k \in S$  and  $t \in \mathbb{R}^+$ .

$$Pr_{s_0}(\{\pi \in Cyl(s_0, \mathbb{R}_0^+, \dots, \mathbb{R}_0^+, s_k) \mid \sum_{i=0}^k t_i \leq t, \pi = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \dots s_k \xrightarrow{t_k} \dots\})$$

Let  $D_{\leq t}(s_0, \dots, s_k)$  be the decomposition of the above set of paths into cylinder sets. This decomposition exists, since it describes time-bounded reachability, which is measurable.

$$= \sum_{Cyl(s_0, I_0, \dots, I_{k-1}, s_k) \in D_{\leq t}(s_0, \dots, s_k)} Pr_{s_0}(Cyl(s_0, I_0, s_1, I_1, \dots, I_{k-1}, s_k))$$

Now we can use the definition of  $Pr_{s_0}$  repeatedly.

$$= \sum_{Cyl(s_0, I_0, \dots, I_{k-1}, s_k) \in D_{\leq t}(s_0, \dots, s_k)} \prod_{i=0}^{k-1} \frac{R(s_i, s_{i+1})}{E(s_i)} \cdot \int_{\inf I_i}^{\sup I_i} E(s_i) \cdot e^{-E(s_i) \cdot x} dx$$

We know that the  $s_i$  are constant for all  $Cyl(s_0, I_0, \dots, I_{k-1}, s_k) \in D_{\leq t}(s_0, \dots, s_k)$ .

$$= \prod_{j=0}^{k-1} \frac{R(s_j, s_{j+1})}{E(s_j)} \cdot \sum_{Cyl(s_0, I_0, \dots, I_{k-1}, s_k) \in D_{\leq t}(s_0, \dots, s_k)} \prod_{i=0}^{k-1} \int_{\inf I_i}^{\sup I_i} E(s_i) \cdot e^{-E(s_i) \cdot x} dx$$

The cylinder sets in  $D_{\leq t}(s_0, \dots, s_k)$  contain all sequences  $t_0, \dots, t_{k-1}$  with  $\sum_{i=0}^{k-1} t_i \leq t$  exactly once. Additionally,  $E(s_i) \cdot e^{-E(s_i) \cdot x}$  is the density function of an exponential distribution with rate  $E(s_i)$ . So the sum in the term describes exactly the probability for the sum of  $k$  exponentially distributed random variables to be less than  $t$ .

$$\begin{aligned} &= \prod_{j=0}^{k-1} \frac{R(s_j, s_{j+1})}{E(s_j)} \cdot PrExpSum_{\leq t}(E(s_0), \dots, E(s_{k-1})) \\ &= Pr_{s_0}(Cyl(s_0, \mathbb{R}_0^+, s_1, \dots, \mathbb{R}_0^+, s_k)) \cdot PrExpSum_{\leq t}(E(s_0), \dots, E(s_{k-1})) \end{aligned}$$

□

After these preparations, we can now prove Theorem 4.2.

*Proof.* Let  $\mathcal{M} = (S, R, L, \nu)$ ,  $s, s' \in S$  and  $\varphi$  a propositional formula over  $AP$ .

First of all, we express time-bounded reachability as a set of paths  $\pi = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \dots$ . Every path which fulfills the property has a minimal index  $k$  at which a state  $s_k$  is reached that satisfies  $\varphi$ . Additionally, this state has to be reached within at most  $t$  time units. In other words, the sum of the sojourn times  $t_0$  to  $t_{k-1}$  has to be less than  $t$ . If there exists a  $k'$  such that  $s_{k'}$  satisfying  $\varphi$  is reached within at most  $t$  time units, then also every other with  $k < k'$  with  $s_k$  satisfying  $\varphi$  is reached within at most  $t$  time units. This allows us to take the minimal index for  $k$  instead of requiring that there exists a  $k$  with the desired properties.

$$Pr_s(\diamond_{\leq t} \varphi) = Pr(\{\pi \in Paths_s \mid \exists_{\min} k \in \mathbb{N}_0 \text{ with } s_k \models \varphi, \sum_{i=0}^{k-1} t_i \leq t\})$$

This set can be subdivided further by the minimal index  $k$  of the first state in a path which satisfies  $\varphi$ . Since these subsets form a partition, we can express the above probability as the sum of the probabilities associated with them. This means, such a set is the set of all paths where  $s_k$  is the first state which satisfies  $\varphi$  and is reached within  $t$  time units.

$$= \sum_{k=0}^{\infty} Pr(\{\pi \in Paths_s \mid s_k \models \varphi, \sum_{i=0}^{k-1} t_i \leq t, \text{ for } 0 \leq i < k, L(s_k) \models \neg \varphi\})$$

In a next step, we will subdivide further by paths which are compatible with a given extended trace of length  $k$ . These have the form  $\sigma = (r_0, A_0), \dots, (r_{k-1}, A_{k-1})$ . So now each set is the set of all paths where  $s_k$  is the first state which satisfies  $\varphi$ , is reached within  $t$  time units and on the way to  $s_k$ , we encounter the same sequence of labelings and exit rates.

$$= \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) \mid s \in S\}}$$

$$Pr_s(\{\pi \in Paths_s \mid \sum_{i=0}^{k-1} t_i \leq t, L(s_i) = A_i \text{ and } E(s_i) = r_i \text{ for } 0 \leq i \leq k\})$$

Now we go down to the level of single time abstract paths. The notation  $\sigma \circ (r_k, A_k)$  describes the extended trace  $\sigma$  with  $(r_k, A_k)$  appended.

$$= \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) \mid s \in S\}} \sum_{(s_0, s_1, \dots, s_k) \in TAPaths(\sigma \circ (r_k, A_k))}$$

$$Pr(\{\pi \in Cyl(s_0, \mathbb{R}_0^+, s_1, \mathbb{R}_0^+, \dots, \mathbb{R}_0^+, s_k) \mid \sum_{i=0}^{k-1} t_i \leq t\})$$

Theorem 4.3 allows us to express the probabilities of timing and paths separately.

$$= \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) \mid s \in S\}} \sum_{(s_0, s_1, \dots, s_k) \in TAPaths(\sigma \circ (r_k, A_k))}$$

$$Pr_s(Cyl(s_0, \mathbb{R}_0^+, s_1, \mathbb{R}_0^+, \dots, \mathbb{R}_0^+, s_k)) \cdot PrExpSum_{\leq t}(E(s_0), \dots, E(s_k))$$

The sequence of exit rates is constant for all  $\pi \in TAPaths_{\sigma \circ (r_k, A_k)}$  and thus is  $PrExpSum_{\leq t}(E(s_0), \dots, E(s_k))$ .

$$= \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) \mid s \in S\}} PrExpSum_{\leq t}(E(s_0), \dots, E(s_k)) \cdot$$

$$\sum_{(s_0, s_1, \dots, s_k) \in TAPaths(\sigma \circ (r_k, A_k))} Pr_s(Cyl(s_0, \mathbb{R}_0^+, s_1, \mathbb{R}_0^+, \dots, \mathbb{R}_0^+, s_k))$$

The innermost sum is by definition  $Pr_s(C_{\sigma \circ (r_k, A_k)})$ .

$$= \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) \mid s \in S\}} PrExpSum_{\leq t}(E(s_0), \dots, E(s_k)) \cdot$$

$$Pr_s(C_{\sigma \circ (r_k, A_k)})$$

This allows us to apply the premise of the theorem.

$$= \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) \mid s \in S\}} PrExpSum_{\leq t}(E(s_0), \dots, E(s_k)) \cdot$$

$$Pr_{s'}(C_{\sigma \circ (r_k, A_k)})$$

By doing these steps backwards we finally arrive at  $Pr_{s'}(\Diamond_{\leq t}\varphi)$ .

□

## MTME between CTMCs implies time-bounded reachability

Now that we have seen the mechanics of the proof for the above simpler case, we can consider the general case of MTME. It shows that two CTMCs which are equivalent under MTME satisfy the same time-bounded reachability properties. Remember that we have already shown that a CTMC and its MTME-quotient are equivalent. So this is the last step towards showing that time-bounded reachability is preserved in the quotient.

**Theorem 4.4.** *Let  $\mathcal{M}_1 = (S_1, R_1, L_1, \nu_1)$  and  $\mathcal{M}_2 = (S_2, R_2, L_2, \nu_2)$  be CTMCs which are equivalent under MTME.*

*Then for all propositional formulae  $\varphi$  over AP it holds that*

$$Pr_{\nu_1}(\diamond_{\leq t}\varphi) = Pr_{\nu_2}(\diamond_{\leq t}\varphi).$$

*Proof.* Let  $\mathcal{M} = (S, R, L, \nu)$  be the union CTMC of  $\mathcal{M}_1$  and  $\mathcal{M}_2$ . There exists an MTME  $\sim$  on  $\mathcal{M}$  such that  $\nu_1$  and  $\nu_2$  are compatible with it, since  $\mathcal{M}_1$  and  $\mathcal{M}_2$  are equivalent under MTME.

$$Pr_{\nu_1}(\diamond_{\leq t}\varphi) = \sum_{s \in S} \nu_1(s) \cdot Pr_s(\diamond_{\leq t}\varphi) = \sum_{C \in S/\sim} \sum_{s \in C} \nu_1(s) \cdot Pr_s(\diamond_{\leq t}\varphi).$$

We now further subdivide the equivalence class  $C$  into its subclasses. However, in general, subclasses do not partition an equivalence class. In other words, the subclasses may not be disjoint and we may not be able to express them as a sum. However, for all subclasses  $SC$  which are not part of the initial distribution, e.g.  $\nu_1(SC) = 0$ , the remaining term equals 0.

So consider only the remaining subclasses  $Subc_{\nu_1}(C) := \{SC \in Subc(C) \mid \nu_1(SC) > 0\}$ . By definition of compatibility between  $\nu_1$  and  $\sim$ , we know that these are in fact disjoint (see Def. 3.9).

$$\sum_{C \in S/\sim} \sum_{s \in C} \nu_1(s) \cdot Pr_s(\diamond_{\leq t}\varphi) = \sum_{C \in S/\sim} \sum_{SC \in Subc_{\nu_1}(C)} \sum_{s \in SC} \nu_1(s) \cdot Pr_s(\diamond_{\leq t}\varphi)$$

Using the proof of Theorem 4.2, we can now substitute  $Pr_s(\diamond_{\leq t}\varphi)$ .

$$\begin{aligned} &= \sum_{C \in S/\sim} \sum_{SC \in Subc_{\nu_1}(C)} \sum_{s \in SC} \nu_1(s) \cdot \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) \mid s \in S\}} \\ &\quad PrExpSum_{\leq t}(E(s_0), \dots, E(s_k)) \cdot Pr_s(C_{\sigma o(r_k, A_k)}) \end{aligned}$$

We now change the position of  $\sum_{SC \in Subc_{\nu_1}(C)} \sum_{s \in SC} \nu_1(s)$  in the formula.

$$\begin{aligned} &= \sum_{C \in S/\sim} \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) \mid s \in S\}} PrExpSum_{\leq t}(E(s_0), \dots, E(s_k)) \\ &\quad \cdot \sum_{SC \in Subc_{\nu_1}(C)} \sum_{s \in SC} \nu_1(s) \cdot Pr_s(C_{\sigma o(r_k, A_k)}) \end{aligned}$$

Since  $\nu_1$  is compatible with  $\sim$ , we can use Definition 3.9 and substitute the term  $\nu_1(s)$ . Let  $s_0(SC)$  be the state which induces  $SC$ . In general, this state is not uniquely determined, since the same subclass can have been induced by different predecessors. In this case however we know that  $\nu_1$  is compatible with  $\sim$  and as a consequence of Definition 3.9 (first condition), this does not change the value of  $weight(s_0(SC), s, SC)$ .

$$\begin{aligned}
&= \sum_{C \in S/\sim} \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) | s \in S\}} PrExpSum_{\leq t}(E(s_0), \dots, E(s_k)) \\
&\quad \cdot \sum_{SC \in Subc_{\nu_1}(C)} \sum_{s \in SC} \nu_1(SC) \cdot weight(s_0(SC), s, SC) \cdot Pr_s(C_{\sigma o(r_k, A_k)}) \\
&= \sum_{C \in S/\sim} \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) | s \in S\}} PrExpSum_{\leq t}(E(s_0), \dots, E(s_k)) \\
&\quad \cdot \sum_{SC \in Subc_{\nu_1}(C)} \nu_1(SC) \sum_{s \in SC} weight(s_0(SC), s, SC) \cdot Pr_s(C_{\sigma o(r_k, A_k)})
\end{aligned}$$

Now we can apply Theorem 4.1. It tells us that the term  $\sum_{s \in SC} weight(s_0(SC), s, SC) \cdot Pr_s(C_{\sigma o(r_k, A_k)})$  is constant for  $SC \in Subc(C)$ . Additionally  $\sum_{SC \in Subc_{\nu_1}(C)} \nu_1(SC) = \nu_1(C)$ .

$$\begin{aligned}
&= \sum_{C \in S/\sim} \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) | s \in S\}} PrExpSum_{\leq t}(E(s_0), \dots, E(s_k)) \\
&\quad \cdot \nu_1(C) \cdot \sum_{s \in SC} weight(s_0(SC), s, SC) \cdot Pr_s(C_{\sigma o(r_k, A_k)}) \text{ where } SC \in Subc(C)
\end{aligned}$$

By Definition 3.10, it holds that  $\nu_1(C) = \nu_2(C)$ .

$$\begin{aligned}
&= \sum_{C \in S/\sim} \sum_{k=0}^{\infty} \sum_{\sigma \in \mathcal{E}T_{\mathcal{M}, \neg\varphi, k}} \sum_{A_k \models \varphi} \sum_{r_k \in \{E(s) | s \in S\}} PrExpSum_{\leq t}(E(s_0), \dots, E(s_k)) \\
&\quad \cdot \nu_2(C) \cdot \sum_{s \in SC} weight(s_0(SC), s, SC) \cdot Pr_s(C_{\sigma o(r_k, A_k)}) \text{ where } SC \in Subc(C)
\end{aligned}$$

Going backwards from here, we finally obtain  $Pr_{\nu_2}(\Diamond_{\leq t}\varphi)$ . □

### Extension to until formulae

A very similar proof can be done in order to show that  $Pr_s(\psi U_{\leq t}\varphi) = Pr'_s(\psi U_{\leq t}\varphi)$ . In the above proofs, we require that for all states prior to reaching the  $\varphi$ -state  $\neg\varphi$  holds. For an until formula this is then changed to  $\neg\varphi \wedge \psi$ . It is important to note that this proof only works for time-bounded until, i.e. an interval  $[0, t]$  and not for interval-bounded until, i.e. an interval  $[t_1, t_2]$ . The reason is that the proof relies on taking the first state  $s_k$  which satisfies  $\varphi$ . But for an interval-bounded until with interval  $[t_1, t_2]$  this state could have been reached before  $t_1$  time-units have passed.



# Chapter 5

## Minimization algorithm

This chapter presents an algorithm that computes an MTME for a given CTMC. After describing the idea and motivating the algorithm, we show an example run. Then we show total correctness of the algorithm and further results which support the argument that the full minimization potential of MTME can be achieved with the algorithm. The algorithm requires to take a non-deterministic choice, which accounts for the fact that in general there exist multiple maximal MTMEs for a CTMC. Finally an algorithm is constructed which heuristically takes this non-deterministic choice.

### 5.1 Algorithm

In order to construct a minimization algorithm for MTME, we go back to its definition (Def. 3.6): Conditions (1) and (2) say that all states within an equivalence class share the same labeling and exit rate. These are static in the sense that they can be checked for an equivalence class independently of the rest of the MTME. This means, we can make sure they are satisfied at the beginning of the algorithm by an initial partition. As long as we only refine this partition, we do not have to care about them any further. The question if condition (3) is fulfilled for some equivalence class, however, depends on the rest of an MTME. This notion that condition (3) is fulfilled for some equivalence class  $C$  and partition  $\mathfrak{C}$  will play an important role in the algorithm and we will call it weighted rate consistency.

**Definition 5.1.** Let  $\mathcal{M} = (S, R, L, \nu)$  be a CTMC.  $\mathfrak{C} \subseteq 2^S$  a partition of  $S$  and  $C \in \mathfrak{C}$ .  $C$  is weighted rate consistent (wr-consistent) with respect to  $\mathfrak{C}$  iff for all  $D \in \mathfrak{C}$ , for all  $s_0, s'_0 \in \text{pred}(C)$ ,  $SC = \text{Subc}(C, s_0)$ ,  $SC' = \text{Subc}(C, s'_0)$

$$\text{wrate}(s_0, SC, D) = \text{wrate}(s'_0, SC', D).$$

If for all  $C \in \mathfrak{C}$  it holds that  $C$  is weighted rate consistent with respect to  $\mathfrak{C}$ , condition (3) is fulfilled for the equivalence relation that is induced by  $\mathfrak{C}$ .

We have seen before, that for a given CTMC there are multiple MTMEs and, more importantly, that there is no unique maximal MTME. This is reflected in the algorithm in the form of a non-deterministic choice. We develop a heuristic for this non-deterministic choice in Section 5.2.

## Data structures

The algorithm uses slightly different mathematical constructs to describe MTME.

- $\mathcal{C}$  is a partition of  $S$  and represents the current solution. It is refined during the algorithm.
- $\mathcal{C}'$  is a subset of  $\mathcal{C}$  and contains those equivalence classes that are potentially not wr-consistent with respect to  $\mathcal{C}$  and thus might violate condition (3) of MTME.
- Subclasses are tuples in the algorithm. Let  $C \subseteq S$  and  $s_0 \in \text{pred}(C)$ . Then  $SC \in \text{Subc}(C, s_0)$  is modeled as  $(s_0, SC) \in S \times 2^S$ . So in the algorithm we have to be a bit more precise and distinguish subclasses that have the same members but are induced by different predecessors.
- While we split an equivalence class of  $\mathcal{C}$ , we construct new equivalence classes  $C_i \subseteq S \times 2^S$  as a set of subclasses. After the construction we can go back to a simple set of states by using  $\text{states}(C_i) = \bigcup_{(s_0, SC) \in C_i} SC$ .

## Algorithm 1

**Input:** CTMC  $\mathcal{M} = (S, R, L, \nu)$  with  $\text{pred}(s) \neq \emptyset$

### Part 1: Initial partitioning

Let  $R \subseteq S \times S$  be the relation defined by:  $(s, s') \in R \Leftrightarrow L(s) = L(s') \wedge E(s) = E(s')$   
 Let  $\mathcal{C} = S/R$

### Part 2: Main loop head

$\mathcal{C}' = \mathcal{C}$  (\*  $\mathcal{C}'$  contains the classes which still have to be refined \*)  
 while  $\mathcal{C}' \neq \emptyset$   
   Let  $C_{old} \in \mathcal{C}'$   
   If  $C_{old}$  wr-consistent with respect to  $\mathcal{C}$   
      $\mathcal{C}' = \mathcal{C}' \setminus C_{old}$   
   else

### Part 3: Expanding $C_{old}$ into all potential subclasses

$\text{Subclasses} = \emptyset \subseteq 2^{S \times 2^S}$   
 for all  $s_0 \in \text{pred}(C_{old})$   
    $SC = \text{post}(s_0) \cap C_{old}$  (\* =  $\text{Subc}(C_{old}, s_0)$  \*)  
   for all  $SC' \in 2^{SC} \setminus \emptyset$   
      $\text{Subclasses} = \text{Subclasses} \cup (s_0, SC')$

**Part 4: Splitting those subclasses**

Let  $\sim_{split} \subseteq (S \times 2^S) \times (S \times 2^S)$  be the equivalence relation on *Subclasses* defined by:

$$(s_0, SC) \sim_{split} (s'_0, SC') \Leftrightarrow \forall D \in \mathfrak{C} : \sum_{s \in SC} weight(s_0, s, SC) \cdot R(s, D) \\ = \sum_{s' \in SC'} weight(s'_0, s', SC') \cdot R(s', D)$$

Let  $splitSC = Subclasses / \sim_{split}$

**Part 5: Choosing new equivalence classes**

Choose  $C_1, \dots, C_n \subseteq S \times 2^S$  such that

- $states(C_1), \dots, states(C_n)$  form a partition of  $C_{old}$
- for all  $1 \leq i \leq n$  there exists  $C' \in splitSC$  such that  $C_i \subseteq C'$
- for all  $s \in C_{old}, s_0 \in pred(s)$  there exists  $i \in \{1, \dots, n\}$  and  $(s_0, SC) \in C_i$  with  $s \in SC$

Let  $newClasses = \{states(C_1), \dots, states(C_n)\}$

**Part 6: Main loop tail**

$$\mathfrak{C} = (\mathfrak{C} \setminus \{C_{old}\}) \cup newClasses$$

$$\mathfrak{C}' = \mathfrak{C}$$

end (\* if \*)

end (\* while \*)

**Output**

Output:  $\mathfrak{C}$

**Part 1**

For an MTME, states within an equivalence class share the same labeling and exit rate (conditions (1) and (2) of MTME). So the state-space is initially partitioned by these two criteria. The resulting partition is called  $\mathfrak{C}$  and will be further refined during the course of the algorithm.

**Part 2**

The further refinement can be seen as a fixpoint computation. The fixpoint is reached, when every equivalence class of the current solution is wr-consistent and thus the solution as a whole fulfills condition (3) of MTME.  $\mathfrak{C}$  contains the current solution and  $\mathfrak{C}'$  contains equivalence classes which potentially violate condition (3).

One equivalence class is inspected with regard to condition (3). If it does not fulfill the condition, it is partitioned in such a way that the new classes do fulfill the condition.

### Part 3

Our goal is to partition  $C_{old}$  in such a way into new classes that these classes are wr-consistent. So let  $\{C_1, \dots, C_n\}$  be this partition of  $C_{old}$ . Condition (3) speaks of all subclasses of such a new class  $C_i$ . However, we do not know beforehand how we have to partition  $C_{old}$ . In other words, each new class  $C_i$  could be any subset of  $C_{old}$ . For this reason Part 3 generates all subclasses of all subsets of  $C_{old}$ . So  $Subclasses = \{(s_0, SC) \in S \times 2^S \mid s_0 \in pred(C_{old}), SC \subseteq Subc(C_{old}, s_0)\}$ .

### Part 4

For each of those subclasses  $(s_0, SC)$  we now consider the weighted rate  $wrate(s_0, SC, D)$  into all equivalence classes  $D$  and split them accordingly. The result is a set of sets of subclasses  $splitSC$ . For every set of subclasses that is an element of  $splitSC$ , we know that all subclasses in this set share the same weighted rates. This is exactly what condition (3) of the MTME definition demands for subclasses of a class.

### Part 5

However two conditions are not met yet:

- $\{C_1, \dots, C_n\}$  has to partition  $C_{old}$ .
- Every new class  $C_i$  must cover predecessors of the contained states. In other words, if  $s_0$  and  $s'_0$  are both predecessors of  $s$  and  $(s_0, SC) \in C_i$  with  $s \in SC$  then also  $(s'_0, SC')$  with  $s \in SC'$  has to be in  $C_i$ . Otherwise  $wrate(s_0, s, D)$  could be different from  $wrate(s'_0, s, D)$  for some equivalence class  $D$  and this would violate condition (3).

These conditions are exactly mirrored in those of the algorithm. At this point we have to take a non-deterministic choice.

This flexibility is needed in order to be able to compute any MTME. Which MTME we get as a result of the algorithm solely depends on the sequence of choices we take at this point.

### Part 6

What remains to do for this iteration of the loop is to do the actual refinement of  $\mathcal{C}$  by replacing  $C_{old}$  with  $newClasses$ . Since we have changed  $\mathcal{C}$ , we cannot be sure anymore that the equivalence classes we already eliminated from  $\mathcal{C}'$  as being wr-consistent still are. So we have to reset  $\mathcal{C}'$  to the full set  $\mathcal{C}$ .

## 5.1.1 Example run of the algorithm

We show a run of the algorithm on the CTMC in Fig. 5.1. In Part 1, the initial partitioning is done. States that share the same labeling and exit rate are put into the same class. This results in the initial partitioning

$$\mathcal{C} = \{\{s_0\}, \{s_1\}, \{s_2\}, \{s_3, s_4, s_5, s_6\}, \{s_a\}, \{s_b\}, \{s_c\}\}.$$

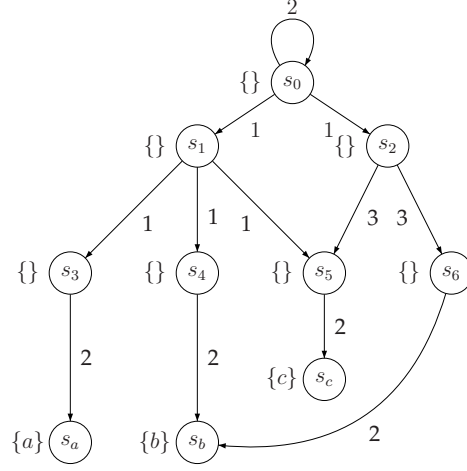


Figure 5.1: Example for Algorithm 1.

The set of possibly not wr-consistent classes  $\mathcal{C}'$  is initialized with  $\mathcal{C}$  and the main loop starts in Part 2. We choose a  $C_{old}$  from this set and check it for wr-consistency. If it is wr-consistent, we can eliminate it from  $\mathcal{C}'$  and consider a new  $C_{old}$ . Otherwise we have to split  $C_{old}$  in order to make it wr-consistent. Since classes with only one member are trivially wr-consistent and are immediately removed from  $\mathcal{C}'$ , the only interesting case is  $C_{old} = \{s_3, s_4, s_5, s_6\}$ .

Part 3 constructs all possible subclasses of  $C_{old}$ . For this we go through all members of  $pred(C_{old}) = \{s_1, s_2\}$  and construct the associated subclasses of  $C_{old}$ . These are  $\{s_3, s_4, s_5\}$  for  $s_1$  and  $\{s_5, s_6\}$  for  $s_2$ . *Subclasses* then contains all non-empty subsets of these, annotated with the inducing predecessor.

$$\begin{aligned} Subclasses = & \{(s_1, \{s_3, s_4, s_5\}), (s_1, \{s_3, s_4\}), (s_1, \{s_4, s_5\}), (s_1, \{s_3, s_5\})\} \\ & \cup \{(s_1, \{s_3\}), (s_1, \{s_4\}), (s_1, \{s_5\}), (s_2, \{s_5, s_6\}), (s_2, \{s_5\}), (s_2, \{s_6\})\}. \end{aligned}$$

The job of Part 4 is to split *Subclasses* according to their weighted rate with all equivalence classes in  $\mathcal{C}$  as destination. In this example, we present this one splitter, i.e. destination of the weighted rate, at a time and start with  $\{s_b\}$ . Let  $P_{s_b, r}$  with  $r \in \mathbb{R}_0^+$  be the class which contains  $(s_0, SC)$  if  $wrate(s_0, SC, s_b) = r$ . We get

- $P_{s_b, 0} = \{(s_1, \{s_3, s_5\}), (s_1, \{s_3\}), (s_1, \{s_5\}), (s_2, \{s_5\})\}$ ,
- $P_{s_b, \frac{2}{3}} = \{(s_1, \{s_3, s_4, s_5\})\}$ ,
- $P_{s_b, 1} = \{(s_1, \{s_3, s_4\}), (s_1, \{s_4, s_5\}), (s_2, \{s_5, s_6\})\}$  and
- $P_{s_b, 2} = \{(s_1, \{s_4\}), (s_2, \{s_6\})\}$ .

This process is repeated for the remaining splitters to obtain

$$splitSC = \{\{(s_1, \{s_3, s_5\})\}, \{(s_1, \{s_3\})\}, \{(s_1, \{s_5\}), (s_2, \{s_5\})\},$$

$\{(s_1, \{s_3, s_4, s_5\})\}, \{(s_1, \{s_3, s_4\})\}, \{(s_1, \{s_4, s_5\}), (s_2, \{s_5, s_6\})\}, \{(s_1, \{s_4\}), (s_2, \{s_6\})\}\}$ .

Now we have to make a non-deterministic choice in Part 5 and construct from *splitSC* wr-consistent equivalence classes which can replace  $C_{old}$ . The first of two possibilities is  $C_1 = \{(s_1, \{s_3\})\}$  and  $C_2 = \{(s_1, \{s_4, s_5\}), (s_2, \{s_5, s_6\})\}$ .  $C_1$  covers  $s_3$  and  $C_2$  covers  $s_4, s_5$  and  $s_6$ . So we cover all states.

This possibility results in the least number of classes. Since our goal is minimization, choosing such a possibility is a good choice heuristically. It leads to the replacement of  $C_{old} = \{s_3, s_4, s_5, s_6\}$  by  $\{s_3\}$  and  $\{s_4, s_5, s_6\}$ .

The second possible choice contains only singleton sets as subclasses:  $C'_1 = \{(s_1, \{s_3\})\}$ ,  $C'_2 = \{(s_1, \{s_5\}), (s_2, \{s_5\})\}$  and  $C'_3 = \{(s_1, \{s_4\}), (s_2, \{s_6\})\}$ . If we take this choice we replace  $C_{old}$  by  $\{s_3\}, \{s_5\}$  and  $\{s_4, s_6\}$ .

With both possibilities we have come to a fixpoint, since all classes turn out to already be wr-consistent. The resulting partition for the first possibility is

$$\mathfrak{C} = \{\{s_1\}, \{s_2\}, \{s_3\}, \{s_4, s_5, s_6\}, \{s_a\}, \{s_b\}, \{s_c\}\},$$

while by the second choice we obtain exactly the maximal bisimulation

$$\mathfrak{C} = \{\{s_1\}, \{s_2\}, \{s_3\}, \{s_4, s_6\}, \{s_5\}, \{s_a\}, \{s_b\}, \{s_c\}\}.$$

This shows again that the choice in Part 5 does actually matter, as it determines the size of the minimization.

### 5.1.2 Correctness proof

This section will prove the correctness of the above algorithm for the computation of MTME. More precisely, we want to show that, for every CTMC, the result of the algorithm constitutes an MTME for this CTMC.

#### Outline of the proof

The MTME definition consists of one implicit and two explicit conditions, which directly translate into proof obligations:

Let  $\mathfrak{C} \subseteq 2^S$  be the output of the algorithm and  $\sim = \{(s, s') \in S \times S \mid C \in \mathfrak{C}, s, s' \in C\}$  the relation induced by  $\mathfrak{C}$ .

- $\sim$  is an equivalence relation.
- For all  $s_1 \sim s_2$ :  $L(s_1) = L(s_2)$  and  $E(s_1) = E(s_2)$ .
- For all  $C, D \in \mathfrak{C}$  and all  $s_0, s'_0 \in \text{pred}(C)$ ,

$$\text{wrate}(s_0, C, D) = \text{wrate}(s'_0, C, D).$$

The algorithm consists of an initialization part followed by the main loop. This structure will be reflected in the proofs for the above proof obligations. They will first show that the respective property is fulfilled after the initialization and then show that it is

also a loop invariant for the main loop.

In order to also show total correctness, we will have to show additionally that the algorithm terminates. It is not obvious that the nondeterministic choice we have to take in Part 5 is always possible. So this will have to be shown. Apart from that, we have to show termination of the main loop.

### Lemmata

In order to more easily distinguish the value of  $\mathcal{C}$  at different points of the algorithm, we use subscripts.  $\mathcal{C}_{init}$  is the first value of  $\mathcal{C}$  resulting from the initial partitioning.  $\mathcal{C}_{pre}$  and  $\mathcal{C}_{post}$  denote the value at the beginning and the end of an iteration of the main loop. Finally, the output of the algorithm is called  $\mathcal{C}_{out}$ .

Towards correctness, we first show some lemmata:

- $\mathcal{C}_{out}$  partitions  $S$ . This is equivalent to our proof obligation that the resulting relation is an equivalence relation.
- There always exist  $C_1, \dots, C_n$  which fulfill the conditions of Part 5.
- The cardinality of  $newClasses$  is always greater than 1.

### $\mathcal{C}$ always partitions the state space $S$

The proof consists of two parts: It will be shown that the initial assignment to  $\mathcal{C}$  is a partition of  $S$  and that this property is also a loop invariant for the main loop.

$\mathcal{C}$  is initialized in Part 1 as the quotient of  $S$  under  $R$ . Since  $R$  is an equivalence relation, the initial value of  $\mathcal{C}$  partitions  $S$ .

It remains to show that this property is a loop invariant for the main loop. The only assignment to  $\mathcal{C}$  happens at the end of the main loop:  $\mathcal{C}_{post} = (\mathcal{C}_{pre} \setminus \{C_{old}\}) \cup newClasses$ . As we want to prove a loop invariant, we can assume that  $\mathcal{C}_{pre}$  partitions  $S$ . So all we have to show is that  $newClasses$  partitions  $C_{old}$  and additionally, that  $C_{old}$  is actually a member of  $\mathcal{C}_{pre}$ .

We start with the latter. It always holds  $\mathcal{C}' \subseteq \mathcal{C}$ : Initially  $\mathcal{C}'$  equals  $\mathcal{C}$ . Furthermore in any pass of the loop,  $\mathcal{C}'$  is either assigned the value of  $\mathcal{C}$  or some element is removed from  $\mathcal{C}'$ . So the  $C_{old} \in \mathcal{C}'$ , which is split in one iteration of the loop, is also a member of  $\mathcal{C}$ .

$newClasses$  equals  $\{states(C_1), \dots, states(C_n)\}$  and Part 5 explicitly demands that  $\{states(C_1), \dots, states(C_n)\}$  partitions  $C_{old}$ .

So the assignment in question replaces  $C_{old}$  by a partition of  $C_{old}$  and consequently  $\mathcal{C}$  remains a partition of  $S$ . This also shows that in any iteration of the loop  $\mathcal{C}$  is either unchanged or refined, that is replaced by a strictly finer partition of  $S$ .

So we can conclude that, under the assumption that the loop terminates,  $\mathcal{C}_{out}$  partitions  $S$ .

### There always exist $C_1, \dots, C_n$ which fulfill the conditions of Part 5

There always exists a trivial solution. The idea is to choose all subclasses  $(s_0, SC)$  where  $|SC| = 1$ . In other words, we take  $SplitSC$  and remove all subclasses which are

not of the form  $(s_0, \{s\})$ . So let

$$\{C_1, \dots, C_n\} = \{C \subseteq S \times 2^S \setminus \emptyset \mid C' \in \text{splitSC}, C = C' \cap \{(s_0, \{s\}) \mid s \in C_{old}, s_0 \in \text{pred}(C_{old})\}\}.$$

We have already seen such a choice for  $C_1, \dots, C_n$  as the second possibility in the example run of the algorithm.

We now show that each condition of Part 5 is met:

- $\text{states}(C_1), \dots, \text{states}(C_n)$  form a partition of  $C_{old}$  :
  - $\bigcup_{1 \leq i \leq n} \text{states}(C_i) \subseteq C_{old}$ :  
We work on a CTMC where every state has at least one predecessor. So for any  $s \in C_{old}$  there also exists some  $C' \in \text{splitSC}$  with  $(s_0, \{s\}) \in C'$ . By construction of the solution also  $(s_0, \{s\}) \in C_i$  for some subclass  $C_i$ . So if  $s \in C_{old}$  then also  $s \in \text{states}(C_i)$  for some  $i$ .
  - $\bigcup_{1 \leq i \leq n} \text{states}(C_i) \supseteq C_{old}$ :  
By construction, any  $C_i$  can only contain  $(s_0, \{s\})$  where  $s \in C_{old}$ .
  - $\text{states}(C_i) \cap \text{states}(C_j) = \emptyset$  for all  $1 \leq i < j \leq n$ :  
By contradiction. Let  $i \neq j$ . Assume  $(s_0, \{s\}) \in C_i$  and  $(s'_0, \{s\}) \in C_j$ . From the construction of the trivial solution it follows that there exist  $C'_1, C'_2 \in \text{splitSC}$  with  $C'_1 \neq C'_2$  and  $(s_0, \{s\}) \in C'_1$  and  $(s'_0, \{s\}) \in C'_2$ . This can only be the case if there exists a  $D \in \mathfrak{C}$  such that  $\text{weightedRate}(s_0, \{s\}, D) \neq \text{weightedRate}(s'_0, \{s\}, D)$ . But this is a contradiction, since  $\text{weight}(s_0, \{s\}) = \text{weight}(s'_0, \{s\}) = 1$ .
- for all  $1 \leq i \leq n$  there exists  $C' \in \text{splitSC}$  such that  $C_i \subseteq C'$ : This follows directly from the definition of the trivial solution.
- for all  $s \in C, s_0 \in \text{pred}(s)$  there exists  $i \in \{1, \dots, n\}$  and  $(s_0, SC) \in C_i$  with  $s \in SC$ :  
Let  $s \in C, s_0 \in \text{pred}(s)$ . By construction of  $\text{splitSC}$  there exists  $C' \in \text{splitSC}$  with  $(s_0, \{s\}) \in C'$ . By construction of  $\{C_1, \dots, C_n\}$ , it also holds that  $(s_0, \{s\}) \in C_i$  for some  $i$ .

### The cardinality of $\text{newClasses}$ is always greater than 1

Assume  $|\text{newClasses}| = 1$ . Then  $\text{newClasses}$  has to be of the form  $\{C_1\}$  where  $C_1 = \{SC_1, \dots, SC_n\}$ . Part 4 ensures that  $C_1$  is wr-consistent. Additionally  $\text{newClasses}$  has to partition  $C_{old}$  because of the first condition in Part 5, which means  $C_1 = C_{old}$ . This implies  $C_{old}$  is also wr-consistent.

But now observe that Part 5 can only be reached if  $C_{old}$  is not wr-consistent due the if-clause in Part 2. This is in contradiction to our previous result. Thus always

$$|\text{newClasses}| > 1.$$

### Conditions (1) and (2) of MTME are fulfilled for $\mathfrak{C}_{out}$

These conditions require that states which are considered equal under an MTME share the same labeling and exit rates. Part 1 splits the state space exactly by these conditions. So the resulting initial partition fulfills this property. This partition is then only further refined, so the property is preserved.

### Condition (3) of MTME is fulfilled for $\mathfrak{C}_{out}$

The main loop ends when  $\mathfrak{C}' = \emptyset$ . By inspection of the algorithm, we can see that this can only happen when every  $C \in \mathfrak{C}$  is wr-consistent with respect to  $\mathfrak{C}$  and thus removed from form  $\mathfrak{C}'$  in Part 5. This is exactly the case when  $\mathfrak{C}$  fulfills condition (3).

### Correctness

**Theorem 5.1.** *Let  $\mathcal{M}$  be a CTMC and serve as input to Algorithm 1. Then the output  $\mathfrak{C}$  is an MTME on  $\mathcal{M}$ .*

*Proof.* We have seen that the output is an equivalence and fulfills conditions (1)-(3) of the MTME definition. Therefore it is an MTME.  $\square$

### Termination of the main loop

In every iteration of the loop either  $\mathfrak{C}$  is refined, i.e. gains cardinality, or  $\mathfrak{C}$  is left unchanged and  $\mathfrak{C}'$  after the iteration is a proper subset of itself at the beginning of the iteration. It suffices to consider just the cardinality of these sets.

So consider the termination relation  $\succ \subseteq (\mathbb{N} \times \mathbb{N}) \times (\mathbb{N} \times \mathbb{N})$  defined as

$$(c_1, c'_1) \succ (c_2, c'_2) \text{ iff } (c_1 < c_2) \text{ or } (c_1 = c_2 \text{ and } c'_1 > c'_2)$$

The first component represents the cardinality of  $\mathfrak{C}$  in the algorithm.  $\mathfrak{C}$  can at most have cardinality  $|S|$  and at least cardinality 1, as it is a partition of  $S$ . The second component represents the cardinality of  $\mathfrak{C}'$ . It can be at most the empty set with cardinality 0 and has at least cardinality  $|S|$ . So there are only finitely many tuples describing a valid state of the algorithm and the largest element in  $\succ$  is  $(|S|, 0)$  in the sense that there does not exist a  $(c, c')$  with  $(|S|, 0) \succ (c, c')$ .

At the beginning of a loop iteration let  $c_1 = |\mathfrak{C}|$  and  $c'_1 = |\mathfrak{C}'|$ . After the end of this iteration let  $c_2 = |\mathfrak{C}|$  and  $c'_2 = |\mathfrak{C}'|$ . We now want to show that always  $(c_1, c'_1) \succ (c_2, c'_2)$ . We have to distinguish the two cases depending on if  $C_{old}$  is wr-consistent or not. If it is,  $\mathfrak{C}$  remains unchanged, i.e.  $c_1 = c_2$ , and  $c'_2 = c'_1 - 1$ . So in this case  $(c_1, c'_1) \succ (c_2, c'_2)$ .

If  $C_{old}$  is not wr-consistent we split  $C_{old}$ . We have already seen that the resulting new  $\mathfrak{C}$  is strictly finer than the old  $\mathfrak{C}$ . So  $c_1 < c_2$  and thus also in this case  $(c_1, c'_1) \succ (c_2, c'_2)$ .

The argument for termination is now, that with every iteration we get a larger element with respect to  $\succ$ . As there are only finitely many valid tuples, we can at most arrive at  $(|S|, 0)$  where we definitely have to terminate.

Hence, we conclude that the algorithm always terminates. Together with the (partial) correctness we have just also shown total correctness.

### 5.1.3 Further properties

Up to now, we have only shown that the algorithm computes an MTME for every CTMC. Remember that every bisimulation is also an MTME. So for all we have proven until now, the algorithm could always compute bisimulation. Since our goal is minimization, we have to rule out this possibility. We will therefore prove that for every pair of a CTMC  $\mathcal{M}$  and an MTME  $\sim$  there exists a sequence of choices in Part 5 of the algorithm such that the algorithm computes an equivalence which is coarser than  $\sim$ . This assures us that, with the right non-deterministic choices, we can leverage the full minimization potential of MTME with this algorithm.

**Theorem 5.2.** *Let  $\mathcal{M}$  be a CTMC and  $\sim$  an MTME on  $\mathcal{M}$ . Let  $\mathcal{M}$  be the input to Algorithm 1. Then there exists a sequence of choices for  $C_1, \dots, C_n$  in Part 5 such that the output  $\mathfrak{C}$  is coarser than  $S/\sim$ .*

*Proof.* The proof idea is to split  $C_{old}$  in Part 5 exactly like  $\sim$  suggests. Core of the proof is then to show that this choice fulfills the conditions of Part 5 and we get an output which is coarser than  $S/\sim$ . The latter will again be shown as a loop invariant. So we have as proof obligations:

- The initial value of  $\mathfrak{C}$ , resulting from the initial partitioning, is coarser than  $S/\sim$ .
- If the value of  $\mathfrak{C}$  at the start of an iteration  $\mathfrak{C}_{pre}$  is coarser than  $S/\sim$ , then there exists a choice in Part 5 such that the value of  $\mathfrak{C}$  at the end of the iteration  $\mathfrak{C}_{post}$  is coarser than  $S/\sim$ .

The first proof obligation is trivial since the initial partitioning is the coarsest partition which puts states which share labeling and exit rate together. Any MTME, including  $\sim$ , is finer than this partition.

We start the proof of the second obligation with an observation about  $C_{old}$ . Since  $\mathfrak{C}_{pre}$  is coarser than  $S/\sim$ ,  $C_{old} \in \mathfrak{C}'$  and  $\mathfrak{C}' \subseteq \mathfrak{C}_{pre}$  there exists  $S' \subseteq S/\sim$  with  $C_{old} = \bigcup_{C \in S'} C$ . In other words, the subset  $S'$  of  $S/\sim$  partitions  $C_{old}$ . Let  $S' = \{Cl_1, \dots, Cl_n\}$ . We want  $S'$  to become the wr-consistent replacement *newClasses* of  $C_{old}$ . So we construct the non-deterministic choice as

$$C_i = \{(s_0, Subc(Cl_i, s_0)) \mid s_0 \in pred(Cl_i)\}.$$

We now have to show that the three conditions of Part 5 are fulfilled.

- $states(C_1), \dots, states(C_n)$  form a partition of  $C_{old}$ :  
 $\{Cl_1, \dots, Cl_n\}$  is a partition of  $C_{old}$ . So by construction this is trivially true.
- for all  $1 \leq i \leq n$  there exists  $C' \in splitSC$  such that  $C_i \subseteq C'$ :  
 $splitSC$  is the quotient of *Subclasses* under  $\sim_{split}$ . So we can show instead that for all  $1 \leq i \leq n$  and for all  $(s_0, SC), (s'_0, SC') \in C_i$  holds  $(s_0, SC) \in Subclasses$  and  $(s_0, SC) \sim_{split} (s'_0, SC')$ .
  - We first show  $(s_0, SC) \in Subclasses$ : Remember that *Subclasses*  
 $= \{(s_0, SC) \in S \times 2^S \mid s_0 \in pred(C_{old}), SC \subseteq Subc(C_{old}, s_0)\}$ . By construction of  $C_i$ ,  $(s_0, SC) = (s_0, Subc(Cl_i, s_0))$  where  $Cl_i$  is a subset of  $C_{old}$  and  $s_0 \in pred(Cl_i)$ . Since  $Subc(Cl_i, s_0) \subseteq Subc(C_{old}, s_0)$ ,  $(s_0, SC) \in Subclasses$ .

$$\begin{aligned}
& - (s_0, SC) \sim_{split} (s'_0, SC'): \text{ Let } D \in \mathfrak{C}_{pre}. \text{ Since } \mathfrak{C}_{pre} \text{ is coarser than } S/\sim \text{ there} \\
& \text{ exists a partition } D_p \text{ of } D \text{ with } D_p \subseteq S/\sim. \\
& \text{ So } \sum_{s \in SC} \text{weight}(s_0, s, SC) \cdot R(s, D) = \sum_{s \in SC} \text{weight}(s_0, s, SC) \cdot \sum_{D' \in D_p} R(s, D') = \\
& \sum_{D' \in D_p} \sum_{s \in SC} \text{weight}(s_0, s, SC) \cdot R(s, D') = \sum_{D' \in D_p} \text{wrate}(s_0, Cl_i, D'). \\
& \text{ Since } Cl_i \in S/\sim \text{ and } D_p \subseteq S/\sim, \sum_{D' \in D_p} \text{wrate}(s_0, Cl_i, D') = \sum_{D' \in D_p} \text{wrate}(s'_0, Cl_i, D') = \\
& \sum_{s \in SC'} \text{weight}(s'_0, s, SC') \cdot R(s, D) \text{ and thus } (s_0, SC) \sim_{split} (s'_0, SC').
\end{aligned}$$

- for all  $s \in C_{old}, s_0 \in pred(s)$  there exists  $i \in \{1, \dots, n\}$  and  $(s_0, SC) \in C_i$  with  $s \in SC$ :

Let  $s \in C_{old}, s_0 \in pred(s)$ . Then there exists an  $Cl_i$  with  $s \in Cl_i$  and by construction  $(s_0, Subc((Cl_i, s_0))) \in C_i$ .

We have just shown that our constructed choice passes the conditions of Part 5. Also this choice is constructed in a way such that  $newClasses = S'$  where  $S'$  is a partition of  $C_{old}$  and a subset of  $S/\sim$ . So  $\mathfrak{C}_{post} = (\mathfrak{C}_{pre} C_{old}) \cup newClasses$  remains coarser than  $S/\sim$ .

We have satisfied all proof obligations, which concludes the proof.  $\square$

We have seen in the correctness proof that in Part 5 there always exists a trivial solution. In fact, always choosing this trivial solution yields bisimulation. The algorithm in this case degenerates into the partition refinement algorithm for bisimulation. Remember that in the trivial solution, we only use subclasses of the form  $(s_0, \{s\})$ . For such a subclass wr-consistency degenerates into the condition of bisimulation that bisimilar states share the same rate into other equivalence classes:

Let  $(s_0, \{s\}) \in S \times 2^S$  and  $D \in \mathfrak{C}$ . Then  $\sum_{s' \in \{s\}} \text{weight}(s_0, s, \{s\}) \cdot R(s, D) = R(s, D)$ .

So for these subclasses, Part 4 does exactly the same as the partition refinement algorithm for bisimulation, except for the fact that it is done for all  $D \in \mathfrak{C}$  at the same time.

## 5.2 Heuristics for the non-deterministic choice

In this section, we want to explore algorithms for the non-deterministic choice in Part 5. We will first present a simple heuristic to measure the quality of a choice regarding minimization. Then we analyze the conditions in Part 5 and formulate them as an exact (set) cover problem with additional constraints. Finally, we propose an algorithm to solve this problem.

### Heuristical target function

Our goal in this thesis is to minimize CTMCs. The benefit of minimization lies in the reduction of the state space. So naturally we want an MTME to have as few equivalence classes as possible. Our heuristic for the quality of a choice in Part 5 will therefore be to minimize the number of new classes which replace  $C_{old}$ . This can only

	a	b	c	d	e
$T_1$	1	1	0	1	0
$T_2$	0	1	0	0	1
$T_3$	0	0	1	0	1

Figure 5.2: Example: Exact cover.

be a heuristic as a choice we make at one point might prevent a better choice at some later point in the algorithm. Part 5 can now be seen as a minimization problem.

### Exact (set) cover

Part 5 can be seen as a modified exact set cover. The exact set cover problem deals with the task to determine if from a set of subsets of a universe  $U$  we can choose elements such that they partition  $U$ .

**Definition 5.2.** Let  $U$  be a set called universe and  $T = \{T_1, \dots, T_n\}$  a set of subsets of  $U$ . Then the exact set cover problem is to determine if there exists a subset  $T' \subseteq T$  such that  $T'$  partitions  $U$ .

Let, for example,  $U = \{a, b, c, d, e\}$  and  $T = \{\{a, b, d\}, \{b, e\}, \{c, e\}\}$ . Then we have to choose a subset  $T' \subseteq T$  which partitions  $U$ . In this case  $T' = \{\{a, b, d\}, \{c, e\}\}$ . In an algorithm, it is advantageous to represent the exact set cover problem as a 0-1-matrix. Every row corresponds to an element of  $T$  and every column to an element of  $U$ . There is a 1 in a cell corresponding to  $T_i \in T$  and  $e \in U$  if and only if  $e \in T_i$ . In other words, every row contains the characteristic vector of its corresponding set  $T_i$ . For the above example, we get the matrix in Fig. 5.2. The task for such a matrix is now to choose a subset of rows such that there is exactly a single 1 in every column. If we choose  $T_1$  to be in the solution  $T'$  and  $a \in T_1$  we say that  $T_1$  covers  $a$ .

So, how does this relate to the problem in Part 5? The connection lies in the third condition, which requires that for all pairs  $s \in C_{old}$  and  $s_0 \in pred(C_{old})$  there exists a  $C_i$  in the solution which covers it.

The underlying exact cover is then defined by:

- Universe  $U = \{(s_0, s) \in S \times S \mid s \in C_{old}, s_0 \in pred(s)\}$ .
- $T$  is derived from *Subclasses*. If  $(s_0, SC) \in Subclasses$  then  $\{s_0\} \times SC \in T$ .

Strictly speaking, Part 5 as described in the algorithm does allow a situation where there can be for example  $(s_0, \{s_1, s_2\}), (s_0, \{s_2\}), (s_0, \{s_1\}) \subseteq C_i$  and thus we have  $(s_0, s_1)$  covered more than once. This does however not influence  $states(C_i)$  as these are only different representations resulting in the same *newClasses* and so we do not ignore any valid solutions by requiring exactly one cover. In other words, for every such solution where an element of the universe is covered more than once there exists a solution which leads to the same *newClasses* which covers every element exactly once.

$pred(s)$	$s_1$	$s_1$	$s_1$	$s_2$	$s_2$
$s$	$s_3$	$s_4$	$s_5$	$s_5$	$s_6$
$r_1$	1	0	1	0	0
$r_2$	1	0	0	0	0
$r_3$	0	0	1	0	0
$r_4$	0	0	0	1	0
$r_5$	1	1	1	0	0
$r_6$	1	1	0	0	0
$r_7$	0	1	1	0	0
$r_8$	0	0	0	1	1
$r_9$	0	1	0	0	0
$r_{10}$	0	0	0	0	1

Figure 5.3: Example: Modified exact cover for Part 5.

We use Part 5 of the example run in Section 5.1.1 also as an example here. This leads to the matrix in Fig. 5.3.

The second condition of Part 5 says that the elements of every  $C_i$  have to be in the same equivalence class of  $splitSC$ . So for the solution it is important which subclasses, corresponding to rows in the matrix, are in the same equivalence class. We group these equivalence classes together into sets  $G_i$  containing rows  $r_{i,j}$  and call these groups. Figure 5.4 shows this grouping for the example. Our minimization goal is to construct a solution with as few  $C_i$  as possible. This is equivalent to using rows from as few groups as possible in a solution.

Now we have to address the remaining condition, which requires that  $C_{old}$  is partitioned by  $\{states(C_1), \dots, states(C_n)\}$ . We have already made sure that  $\bigcup_i states(C_i) = C_{old}$ , as we have to cover the whole universe. So for a partition we only have to additionally require that the  $states(C_i)$  are pairwise disjoint. So we have to make sure that if  $(s_0, s)$  is covered by a row in  $G_i$  then all  $(s'_0, s) \in U$  have to also be covered by a row in  $G_i$ .

Figure 5.5 shows one possible solution. In every column there is exactly a single 1. Also  $(s_1, s_5)$  and  $(s_2, s_5)$  are covered by rows from the same group. The quality of this solution is 2, as we use two groups, which is also the minimum. In contrast to this, choosing  $r_{1,1}$ ,  $r_{6,2}$  and  $r_{7,1}$  does not constitute a solution even though it is a set cover, since  $(s_1, s_5)$  is covered by a row in  $G_1$  and  $(s_2, s_5)$  by a row from  $G_6$ .

### Reducing problem size by preprocessing

Problem instances of this modified exact cover can be reduced with a bit of simple preprocessing. The idea is to remove rows that can never be in a solution since taking them results in cover obligations which cannot be satisfied.

For example, Group  $G_4$  in Fig. 5.4 contains only row  $r_{4,1}$  and covers  $(s_1, s_5)$  but not  $(s_2, s_5)$ . But we know that if we take this row we have to cover both within the group.

$pred(s)$		$s_1$	$s_1$	$s_1$	$s_2$	$s_2$
$s$		$s_3$	$s_4$	$s_5$	$s_5$	$s_6$
$G_1$	$r_{1,1}$	1	0	1	0	0
$G_2$	$r_{2,1}$	1	0	0	0	0
$G_3$	$r_{3,1}$	0	0	1	0	0
$G_3$	$r_{3,2}$	0	0	0	1	0
$G_4$	$r_{4,1}$	1	1	1	0	0
$G_5$	$r_{5,1}$	1	1	0	0	0
$G_6$	$r_{6,1}$	0	1	1	0	0
$G_6$	$r_{6,2}$	0	0	0	1	1
$G_7$	$r_{7,1}$	0	1	0	0	0
$G_7$	$r_{7,2}$	0	0	0	0	1

Figure 5.4: Example: Modified exact cover with groups.

$pred(s)$		$s_1$	$s_1$	$s_1$	$s_2$	$s_2$
$s$		$s_3$	$s_4$	$s_5$	$s_5$	$s_6$
$G_2$	$r_{2,1}$	1	0	0	0	0
$G_6$	$r_{6,1}$	0	1	1	0	0
$G_6$	$r_{6,2}$	0	0	0	1	1

Figure 5.5: One possible solution of Fig. 5.4.

So  $r_{4,1}$  cannot be in any solution and can be removed from the problem instance. The group  $G_4$  is now empty can also be removed.

A group  $G \subseteq T$  can be reduced by constructing the following two sets. Both can be computed by simple iterations over the underlying sets.

The set of elements of the universe we cannot cover with any row from  $G$  is given by  $U_{noCover} = U \setminus \bigcup_{G' \in G} G'$ . If  $(s_0, s) \in U_{noCover}$ , we can remove all rows which contain  $(s'_0, s)$ , since both have to be covered within the same group. This way we get a new group  $G_{new} = G \setminus \{G' \in G \mid s_0, s'_0, s \in S, (s_0, s) \in G', (s'_0, s) \in U_{noCover}\}$ . By removing rows from  $G$  we possibly cover fewer elements of  $U$  and we might be able to remove other rows. So we repeat this process until either a fixpoint is reached or the group is empty. In the latter case we can remove the group altogether.

### Avoiding repeated computation of weighted rate consistency

In Part 2, we have to check  $C_{old} \in \mathcal{C}'$  for wr-consistency in order to decide if we have to split it or not. This check can be very expensive as it involves computing the weighted rates of  $C_{old}$  into all equivalence classes of  $\mathcal{C}$ . If  $\mathcal{C}$  and  $\mathcal{C}'$  are implemented as a list and we take  $C_{old}$  from the beginning of the list, we can avoid repeated computations: Once we have determined for a specific  $C_{old}$  that it is wr-consistent, we not only remove it from  $\mathcal{C}'$  but also move it to the end of  $\mathcal{C}$ . This is possible, since  $\mathcal{C}'$  is always a subset of  $\mathcal{C}$ . This prevents that at the beginning of the main loop we have to recheck classes

which are likely still wr-consistent. This optimization should not be underestimated as in some cases it was able to speed up the prototype implementation of the algorithm by a factor of 80.

### 5.2.1 Combine a modified Knuth's Algorithm X with branch & bound

Knuth presents in [Knu00] an algorithm for the exact cover problem. It is called Algorithm X and finds all solutions of an exact cover problem. However the real finesse lies in its implementation using so called Dancing Links. The implementation of Algorithm X using Dancing Links is called DLX.

Algorithm X is mostly a straightforward depth-first-search, also known as backtracking. Roughly speaking, this means that we systematically build up a partial solution by adding rows one at a time and updating the rest of the problem accordingly. If we cannot add any more rows without violating the properties of an exact cover we backtrack. This way, we build a search tree and can enumerate all solutions.

So how does this work in detail? Algorithm X can be seen in Fig. 5.6. Using the exact cover problem from Fig. 5.2 as an example, a run of the algorithm looks like this:

- We choose a column we want to cover. For example column  $b$ .
- In this column we choose a row containing a 1 and add it to our solution. For example  $T_2$ .
- We now go through all 1s in that row. For each corresponding column we delete that column and every row that has a 1 in that column. In our example we delete  $T_2$  itself, column  $b$  and row  $T_1$  because  $T_2$  and  $T_1$  have a 1 in column  $b$ . Finally we delete column  $e$  and row  $T_3$  since there are 1s in column  $e$ , rows  $T_2$  and  $T_3$ .
- This leaves us with a matrix with 3 columns but no rows, which is not empty in the sense of this algorithm. So our non-deterministic choice did not lead to a solution and we backtrack.
- This time we choose row  $T_1$  in column  $b$ . This leads to the elimination of columns  $a, b, d$  and rows  $T_1, T_2$ .
- Now we can choose to cover  $c$  and our only choice of row is  $T_3$ , which is eliminated in the next step.
- We have achieved an empty matrix and can print out the solution.
- We backtrack in order to find additional solutions, but eventually return to the beginning without finding any further solutions.

### Dancing Links

Backtracking is often implemented with a stack: For any decision we take we put a snapshot of the current state of the problem on the stack. So when we backtrack we can undo the decision by restoring the previous state from the stack.

<p>If <math>A</math> is empty, the problem is solved; terminate successfully.  Otherwise choose a column, <math>c</math> (deterministically).  Choose a row, <math>r</math>, such that <math>A[r, c] = 1</math> (non-deterministically)  Include <math>r</math> in the partial solution.  For each <math>j</math> such that <math>A[r, j] = 1</math>,      delete column <math>j</math> from matrix <math>A</math>;      for each <math>i</math> such that <math>A[i, j] = 1</math>,          delete row <math>i</math> from matrix <math>A</math>.  Repeat this algorithm recursively on the reduced matrix <math>A</math>.</p>
--

Figure 5.6: Algorithm X, taken from [Knu00].

However, for an exact cover problem this snapshot can be big and thus incur a large overhead. A better strategy is often to use a global representation of the current solution, change it when we make a decision and undo this change when we backtrack. We are in need of fast operations for applying and undoing decisions during the search. Dancing Links provides such operations for Algorithm X.

Dancing Links builds on operations for removing and reintroducing elements from doubly-linked circular lists. In a doubly-linked circular list every element has pointers to the previous and next element (doubly-linked) and first and last element also point to each other (circular).

Let  $e$  be an element of a list and let  $L[e]$  give the previous (left) neighbor and  $R[e]$  the next (right) neighbor of  $e$  in the list.

We can remove an element from the list by the assignments  $L[R[x]] \leftarrow L[x]$  and  $R[L[x]] \leftarrow R[x]$ . Interestingly, undoing this operation is equally easy and only requires knowing  $x$ . In order to reintroduce  $x$  into the list we assign  $L[R[x]] \leftarrow x$  and  $R[L[x]] \leftarrow x$ . If we add an additional header element  $h$  to a list these operations also support the empty list. A list is then empty if and only if the pointers of the header element point to itself, i.e.  $R[h] = h$ .

### Exact cover using Dancing Links

We want to express an exact cover problem as doubly-linked circular lists in such a way that we can implement Algorithm X on this basis using Dancing Links.

How this is done can be seen in Fig. 5.7. An exact cover problem is described by a 0-1-matrix. The idea is now to take the 1s and put them in two lists: One list for each row connecting a 1 with its left and right neighbors and one list for each column. Additionally, every column list has a column header  $ch$  as an additional element. These column headers are themselves in a list together with the root header  $h$ . Not shown in Fig. 5.7 is another pointer from every 1 to its column header.

In summary, every object has 5 pointers:  $L$  and  $R$  for the horizontal list,  $U$  and  $D$  for the vertical list and  $C$  for the pointer to the column header.

The Dancing Links implementation of Algorithm X, abbreviated DLX, is shown in Fig. 5.8. It is invoked as  $search(0)$ . In addition to the lists we have already seen, it uses variables  $O_k$  for the row we have added at depth  $k$  of the search. Choosing a column

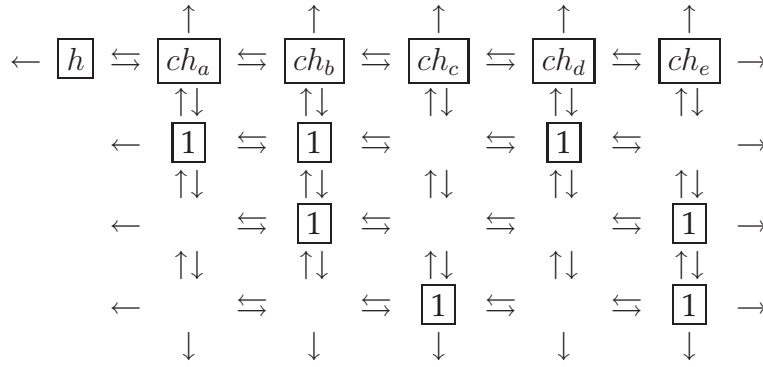


Figure 5.7: Example for Dancing Links.

to cover can be done by taking the successor of the root header  $R[h]$ . Alternatively we can keep track about the number of remaining 1s in each column and choose the one with the lowest number. The idea is to reduce the branching factor of the search tree at the cost of a bit of additional record keeping. The details of this improvement can be found in the original paper.

The Dancing Links technique is used for covering and uncovering columns.

To cover a column  $c$  we have to do the following steps:

Set $L[R[c]] \leftarrow L[c]$ and $R[L[c]] \leftarrow R[c]$ .	Remove $c$ from column header list
For each $i \leftarrow D[c], D[D[c]], \dots$ , while $i \neq c$ ,	Go through column $c$
for each $j \leftarrow R[i], R[R[i]], \dots$ , while $j \neq i$ ,	Go through row $i$
set $U[D[j]] \leftarrow U[j], D[U[j]] \leftarrow D[j]$ .	Remove $j$ from column list

In order to uncover a column we do this in reverse:

For each $i \leftarrow U[c], U[U[c]], \dots$ , while $i \neq c$ ,	Go through column $c$ in reverse
for each $j \leftarrow L[i], L[L[i]], \dots$ , while $j \neq i$ ,	Go through row $i$ in reverse
set $U[D[j]] \leftarrow j, D[U[j]] \leftarrow j$ .	Reintroduce $j$ to column list
Set $L[R[c]] \leftarrow c$ and $R[L[c]] \leftarrow c$ .	Reintroduce $c$ to column header list

A few aspects, which are not essential to the understanding of DLX, have been left out of this presentation. These include reading out and printing a solution and reducing the branching factor by keeping track of the size of columns and choosing columns accordingly. They can be found in the original paper [Knu00].

### Modified Algorithm X for Part 5

Now that we know DLX, which is an efficient technique for the enumeration of the solutions of an exact cover problem, we want to modify it so that we can use it on our original problem. We have already seen how the problem can be expressed as an exact cover with side conditions. Also we are interested in a solution which minimizes a target function.

```

search(k)
If  $R[h] = h$ , print the current solution and return.
Otherwise choose a column object  $c$ .
Cover column  $c$ .
For each  $r \leftarrow D[c], D[D[c]], \dots$ , while  $r \neq c$ ,
    set  $O_k \leftarrow r$ ;
    for each  $j \leftarrow R[r], R[R[r]], \dots$ , while  $j \neq r$ ,
        cover column  $C[j]$ ;
    search( $k + 1$ );
    set  $r \leftarrow O_k$ 
    for each  $j \leftarrow L[r], L[L[r]], \dots$ , while  $j \neq r$ ,
        uncover column  $C[j]$ .
Uncover column  $c$  and return.

```

Figure 5.8: Dancing Links implementation of Algorithm X, taken from [Knu00].

### Cover obligation

We will discuss the handling of the side condition first. It says that if a row from  $G_i$  is part of the current solution and it contains  $(s_0, s)$  then also all other  $(s'_0, s) \in U$  have to be covered by rows from this group.

For example, if we choose  $r_{6,1}$  in Fig. 5.4, we have to cover  $(s_2, s_5)$  with a row from  $G_6$  since  $r_{6,1}$  contains  $(s_1, s_5)$ . So we have to keep track of columns we have to cover within the current group and cover all of them before we can start covering other columns again. So when choosing a column  $c$  to cover we can have two cases:

- Either we are not restricted in our choice and can choose any column and afterwards any row containing a 1 in that column,
- or we are restricted and have to choose a column we are required to cover owing to the side condition. In this column we can additionally only choose a row containing a 1 in that column if it is also in the same group.

### Implementation

From a high level point of view, the changes to accommodate the side condition do not seem to contain large difficulties. However, their implementation in a way which does not diminish the performance advantage of using Dancing Links requires some thought. As a consequence, we have to use global data structures for the current state of the problem and easily reversible operations to change this state.

In order to be able to determine which columns belong to the same state  $s$  for  $(s_0, s) \in U$ , we introduce a new list containing the headers of these columns. So these lists are sublists of the list of column headers. They use the pointers  $SL[ch]$  and  $SR[ch]$  for the neighboring column header. The header of such a list will be called a state header and every column header  $ch$  will have a pointer  $SH[ch]$  to reach it.

We will use an array  $CO$  containing state headers to keep track of the cover obligations. The position of the first uncovered obligation is saved as  $coFirst$  and the first unused position as  $coLast$ . So if  $coFirst = coLast$  we know that at the moment we do not have to cover any more columns while being restricted to the current group.

The values of  $coFirst$  and  $coLast$  are saved at the beginning of  $search(k)$  and restored at the end. This way we do not have to explicitly remove cover obligations when we backtrack.

In order to not clutter the presentation of the algorithm with non-essential detail we will assume that  $coFirst$  automatically points to the first uncovered obligation. In the actual implementation we use an array of booleans containing the information if a column has been covered, this array is updated whenever we cover or uncover a column. Every time we access  $coFirst$  we can simply update it using this array.

We have already seen that we can be restricted in our choice of row to the current group. So we need an efficient data structure to find these rows and thus use circular doubly-linked lists, which are sublists of the column lists. Every such list contains all 1s that are in the same group and the same column together with a group header. Its pointers are called  $GU[c]$  and  $GD[c]$  for group up and group down. We also have to be able to find such a group list given the column and the group number. For this we introduce a new field in each column header containing an array of pointers to its group headers.  $GH[c][i]$  gives the group header of  $G_i$  in column  $c$ .

What we also need is a way to get the group of an element  $r$ . This can for example be implemented by adding a pointer to a group identifier to every object representing a 1 in the matrix. We will call this pointer  $Group[r]$ .

In summary we have as new data structures:

- $SL[ch], SR[ch]$ : A doubly-linked list containing all column header which share the same state  $s$  for  $(s_0, s) \in U$ . It contains the state header.
- $SH[ch]$ : A pointer from a column header to its state header.
- $currentGroup$ : The group of the column we added last to the solution.
- $CO[]$ : An array of state headers, keeping track which columns have to be covered being restricted to the current group.
- $coFirst$ : Index of the first uncovered state header in  $CO$ .
- $coLast$ : Index of the first unused cell in  $CO$ , i.e. the index of the last uncovered state header plus one.
- $GU[c], GD[c]$ : A doubly-linked list for each part of a column which is in the same group. It contains the group header.
- $GH[c][i]$ : A pointer to the group header of group  $G_i$  in column  $c$ .
- $Group[r]$ : The group a row is contained in.

## Branch & bound

In general we could enumerate all solutions and pick the one which minimizes our target function. But we can do better by reducing the search space using branch & bound.

Branch & bound is a standard technique which cuts the search tree by not visiting any branches that cannot obtain a better solution quality than an already known solution. Adding it to the algorithm is straightforward. Our solution quality is the number of groups from which we have used rows. Before we start the search we determine the quality of the trivial solution and set it as our current candidate solution. Every time we go deeper in the search tree we check if we have used more groups than in the candidate solution. If this is the case we backtrack. When we find a new solution we update our candidate solution.

In order to keep track of the number of used groups we use two data structures. On the one hand we have an integer *numUsedGroups* which starts at 0, on the other we have an array of integers *usedGroups[i]* also starting at 0 in every element. *usedGroups[i]* gives the number of rows from group  $G_i$  which are in the solution. Every time we take a row from  $G_i$  into the solution we increment *usedGroups[i]* and every time we remove it we decrement it. Our target function is the number of cells in *usedGroups* which are not 0, so we increment *numUsedGroups* every time a cell in *usedGroups* is incremented from 0 to 1 and decrement it every time the value of a cell goes from 1 to 0. This way *numUsedGroups* always contains the quality of the current solution.

### 5.2.2 Fast probabilistic approach

This approach is based on the modified Algorithm X. The idea is to randomize the traversal of the search tree and stop at the first solution found or after a set amount of time has passed. We can do this several times and take the best solution. This basically trades in the assurance to find the best solution for speed and a bound on the maximal running time. It is especially helpful when there is no better solution than the trivial one and the search tree remains large even when applying branch & bound.

## 5.3 Minimization Workflow

This section describes the process through which an arbitrary CTMC can be minimized using MTME. In other words, it takes what we have seen so far and builds it together. Due to the nature of the results about preservation of properties, we also have to make sure that the initial distribution is compatible with the MTME we use for minimization. How this can be accomplished is also shown.

### Add artificial predecessors

We start with an arbitrary CTMC and use the construction from the beginning of Chapter 3 to introduce an artificial predecessor to every state which previously did not have any. Now that we know MTME we can see that it is essential for every such state to have a new artificial predecessor. If there were only one, this would interfere with the

construction of MTME as all such states would be trivially in the same equivalence class as long as they share labeling and exit rate.

### **Ensure compatibility between initial distribution and MTME**

In a next step we have to prepare the CTMC in such a way, that the MTME we compute with the minimization algorithm is compatible with the initial distribution. We do not have to know the exact initial distribution, but rather which states are potentially initial, i.e. might be assigned a non-zero probability by the initial distribution. We simply add a unique label to every state which is (potentially) initial. This way these states will each be in an equivalence class of their own and trivially the MTME is compatible with the initial distribution.

### **Running the minimization algorithm**

Now we can run the minimization algorithm on this CTMC. The resulting MTME is then used to compute the quotient under MTME. It is sometimes possible to run the algorithm again on the quotient and get an even better minimization.

### **Model checking on the quotient**

The quotient can now be used for model checking of the preserved properties. We will now argue the correctness of this workflow. Specifically, that all conditions are satisfied so that timed reachability in an original CTMC  $\mathcal{M}$  is preserved in the quotient  $\mathcal{M}_q$  resulting from this workflow.

We have seen that the algorithm is correct (Thm. 5.1) and argued that the resulting MTME is also compatible with the initial distribution. Furthermore Thm. 3.2 tells us that  $\mathcal{M}$  and  $\mathcal{M}_q$  are equivalent under MTME. Equivalence between CTMCs is exactly the precondition of Thm. 4.4 from which we can conclude that  $Pr_\nu(\diamond_{\leq t}\varphi) = Pr_{\nu_q}(\diamond_{\leq t}\varphi)$  where  $\nu$  and  $\nu_q$  are the initial distributions of  $\mathcal{M}$  and  $\mathcal{M}_q$ . So we can check timed reachability on the quotient instead of the original CTMC.

```

search(k)
If  $R[h] = h$ , print the current solution and return.
Set  $prevCoFirst \leftarrow coFirst$ .
Set  $prevCoLast \leftarrow coLast$ .
If  $coFirst = coLast$ ,
    Choose a column object  $c$  from the uncovered columns.
    Cover column  $c$ .
     $CO[coLast] \leftarrow GH[c]; coLast \leftarrow coLast + 1$ .
    For each  $r \leftarrow D[c], D[D[c]], \dots$ , while  $r \neq c$ ,
         $currentGroup \leftarrow Group[r]$ .
         $searchRow(r)$ ;
Else
    Choose column object  $c \leftarrow SR[CO[coFirst]]$ .
    Cover column  $c$ .
    Set  $gh \leftarrow GD[c][currentGroup]$ .
    For each  $r \leftarrow GD[gh], GD[GD[gh]], \dots$ , while  $r \neq gh$ ,
         $searchRow(r)$ ;
Endif.
Uncover column  $c$  and return.
Set  $coFirst \leftarrow prevCoFirst$ .
Set  $coLast \leftarrow prevCoLast$ .

searchRow(r)
set  $O_k \leftarrow r$ ;
for each  $j \leftarrow R[r], R[R[r]], \dots$ , while  $j \neq r$ ,
    cover column  $C[j]$ .
     $CO[coLast] \leftarrow GH[C[j]]; coLast \leftarrow coLast + 1$ .
 $search(k + 1)$ ;
set  $r \leftarrow O_k$ 
for each  $j \leftarrow L[r], L[L[r]], \dots$ , while  $j \neq r$ ,
    uncover column  $C[j]$ .

```

Figure 5.9: Modified Dancing Links.

# Chapter 6

## Case studies

This chapter examines the performance of MTME and especially of the minimization algorithm with the help of two case studies. The first case study, modeling a simple peer-to-peer protocol, can be seen as a worst case scenario. It cannot be minimized beyond bisimulation using MTME and it has a long runtime. In contrast to this, the second case study is a good fit for minimization by MTME and can achieve reduction rates of 20 to 40 percent compared to bisimulation. It runs considerably faster than the first case study. It models a physical supply process in which arriving products are stored and then processed in a last-in first-out fashion. The processing can have different results depending on the type of product.

### Prototype implementation

We use a prototype implementation of the algorithms from Chapter 5 written in Java. Specifically it uses Algorithm 1 and the modified Knuth's Algorithm X for Part 5. All described optimizations were implemented.

The prototype uses as an input format the plaintext representations of CTMCs which can be exported by the PRISM model checker. A PRISM model can contain excessive labeling which hinders minimization. So for each input a small piece of program code can be written which maps the labeling used in the PRISM model onto a new labeling which only contains those atomic propositions we are interested in.

Before we can use Algorithm 1, we have to make sure that every state has at least one predecessor. This is done using the construction from the beginning of Chapter 3.

All tests were done on an Intel Core2Duo running at 3GHz and a 64-bit Java VM. The depth-first search lends itself to parallelization, but this was not implemented, and so only one core of the processor is used.

## 6.1 Case study 1: Simple peer-to-peer protocol

This case study models a simple peer-to-peer protocol as a CTMC and has been taken from the PRISM model checker website [PRI09]. It has already been studied with regards to symmetry reduction in [KNP06] and bisimulation minimization in [KKZJ07]. It works like Bittorrent (cf. [Coh03]): A file is cut into  $k$  pieces, which are initially only possessed by a single client. Each one of  $n$  additional clients wants to download all

pieces. The labeling consists of atomic propositions  $done_i$  being present when in the respective state client  $i$  has already received all pieces.

We study two instances of the protocol:  $\mathbf{P2P}_{2,4}$  where  $n = 2$  and  $k = 4$ , which has 256 states and 1025 transitions, and  $\mathbf{P2P}_{2,5}$  where  $n = 2$  and  $k = 5$ , which has 1204 states and 5121 transitions.

We first examine  $\mathbf{P2P}_{2,4}$ . One state of the CTMC has no predecessor, so we have to introduce one new state and one new atomic proposition, bringing the number of states to 257.

The maximal bisimulation of this CTMC has 35 equivalence classes. Using Algorithm 1 to compute the bisimulation, by taking the trivial solution in every iteration, takes 588ms. Since this introduces unnecessary overhead a normal implementation of partition refinement for bisimulation can be estimated as being one order of magnitude faster.

In comparison, the prototype needs 1h13min of cpu time for the computation of the maximal MTME, which has 35 classes as well. This is about a factor of 2200 slower.

We now analyze a run of Algorithm 1 on  $\mathbf{P2P}_{2,4}$  in order to identify which part of the algorithm contributes which fraction of the runtime.

The initial partitioning results in 5 classes and does effectively not contribute to the runtime with less than 10ms.

Table 6.1 contains in each row an invocation of the modified Algorithm X. It contains the size of the class  $C_{old}$  we split. This class induces subclasses which are split into classes again. The table contains their cardinalities before and after preprocessing. We can see that preprocessing works reasonably well, especially when considering the reduction in the number of classes.

The resulting numbers of new classes are low compared to the cardinality of  $C_{old}$ , which is expected of a CTMC which is easily minimized by bisimulation.

The number of visited nodes in the search tree and the time the search took are quite interesting. All but one invocation are quite fast with visited nodes at most in the thousands and runtimes of only a few milliseconds<sup>1</sup>. One invocation however dominates the runtime of Part 5 with 128 million visited states and 1h15min runtime<sup>2</sup>. In fact, it dominates the whole algorithm.

$\mathbf{P2P}_{2,5}$  shows exactly the same picture as  $\mathbf{P2P}_{2,4}$ . The runtime of the whole algorithm is in excess of 24 hours cpu time, after which the run was aborted. This large runtime is again caused by a single invocation of the modified Algorithm X, which could be shown by aborting just this search after a number of iterations. The rest of the algorithm then has a runtime of less than 30s.

When we have a look at the structure of the CTMC in this case study, we see that each state basically corresponds to a bit-vector where every bit represents a piece which has been received by a client. Every state has all states as a predecessor where the corresponding bit-vector contains one bit less which is 1. So the number of predecessors is large for most states. Also the initial partitioning puts all but a few states into one equivalence class which then has to be split. These two circumstances contribute to

---

<sup>1</sup>Millisecond accuracy is possible nowadays in Java. Still, the values in Fig 6.1 only represent the minimum runtime of three runs in order to minimize the effect of background activity on the test system.

<sup>2</sup>This is wall time in contrast to the 1h13min cpu time for the whole algorithm.

	$ C_{old} $	before		after		# new classes	visited nodes	runtime in ms
		#classes	#subcl.	#classes	#subcl.			
1	79	25	1200	16	1020	9	$128 \cdot 10^6$	$4.5 \cdot 10^6$
2	100	22	1158	7	748	7	1184	51
3	42	7	354	4	276	4	169	5
4	24	22	504	6	112	3	3347	7
5	24	5	216	2	108	2	106	2
6	24	12	324	2	108	2	21	3

Table 6.1: Invocations of Part 5 for  $P2P_{2,4}$ .

the high number of states which have to be visited.

The results suggest that the MTME minimization algorithm, at least in this version, cannot be used blindly without risking an explosion in runtime. Modifying it in a way which bounds the runtime of the search, e.g. as proposed in Section 5.2.2, seems to be a better choice.

Apart from the runtime, we are also interested in memory consumption. The memory consumption is estimated using the peak memory consumption of the Java process and subtracting from it the peak memory consumption of a Java process solving a problem instance with only 1 state. Using this method we get a peak memory consumption of 121Mb for  $P2P_{2,4}$  and 340Mb for  $P2P_{2,5}$ . Although taking just these two data points is a weak base for conclusions, they hint into the direction that in the average case we might not have to deal with memory requirements exponential in the number of states.

## 6.2 Case study 2: Last-in first-out supply process

This case study models a system in which a product arrives, is stored in a stack-like (last-in first-out) fashion and is then processed. Physical storage in such a way has a number of advantages over a first-in first-out, queue-like, system. Among them are that it needs less space, since storing and retrieving a product can be done from the same side. So only room for access from one side to the storage area is needed.

The products in the stack are differentiated by type, for example by supplier, and we know at which rate each type arrives. The processing can have different results, this could for example be different qualities which are only discovered when processed, and the probability of these outcomes depends on the type. The space in the stack is limited and we also model the event that the stack becomes full.

One property we could be interested in is the probability to process only products of inferior quality for a certain amount of time. This can be important if in another process, which we have not modeled, this results in running out of raw materials of appropriate quality.

This model is suited for MTME since the type of product which arrives does not become relevant for the observable behavior of the system, i.e. timing and labeling, until

the product is processed. So MTME can abstract from the concrete configuration of the stack and move the decision which specific type of product arrived to a later point in the CTMC.

For the formalization as a CTMC we stick with the terminology of a stack of *products* which is processed with varying *results*. We write a stack of size  $n$  as a sequence  $(p_1, \dots, p_n)$  where the top of the stack is  $p_n$ . If  $numTypes$  is the number of types,  $p_i \in \{1, \dots, numTypes\}$ . The maximal size of the stack is given by  $stackSize$ . The empty stack, i.e.  $n = 0$ , is denoted by  $\varepsilon$ .

We have three kinds of states:

- Stack states  $s_{(p_1, \dots, p_n)}$  where  $(p_1, \dots, p_n)$  is the configuration of the stack. They are labeled with the empty set.
- Result states  $r_{i, (p_1, \dots, p_n)}$  where  $i$  is the result and  $(p_1, \dots, p_n)$  the stack they will return to. They are labeled with  $\{result_i\}$ . The number of results is given by  $numResults$ .
- $s_{full}$  is the state which indicates that the stack was full while a product arrived. It is labeled with  $\{full\}$ .

The rates of transitions are determined by two functions and one constant:

- $rate_a$  gives the arrival rate for every type of product:  
 $rate_a : \{1, \dots, numTypes\} \rightarrow \mathbb{R}_{\geq 0}$ . The sum of these rates is denoted by  $arrivalRate$ .
- Results depend on the type of product:  
 $rate_r : \{1, \dots, numTypes\} \times \{1, \dots, numResults\} \rightarrow \mathbb{R}_{\geq 0}$ . Every product has the same rate  $processRate$  at which it is processed, so  $\sum_{j=1}^{numResults} rate_r(i, j) = processRate$  for all  $i \in \{1, \dots, numTypes\}$ .
- The rate to return from a result is constant and given by  $rate_{return}$ .

A stack state  $s_{(p_1, \dots, p_n)}$  has a transition to  $r_{i, (p_1, \dots, p_{n-1})}$  with rate  $rate_r(p_n, i)$  for all  $i \in \{1, \dots, numResults\}$ . If the stack has maximal size, i.e.  $n = stackSize$ , it has a transition with rate  $arrivalRate$  to  $q_{full}$ . Otherwise it has a transition to  $s_{(p_1, \dots, p_n, i)}$  with rate  $rate_a(i)$  for all  $i \in \{1, \dots, numTypes\}$ . A result state  $r_{i, (p_1, \dots, p_n)}$  goes with rate  $rate_{return}$  to  $s_{(p_1, \dots, p_n)}$ .

An example can be seen in Fig. 6.1. The labeling has been omitted from the figure in order to make it more readable.  $r_{1, \varepsilon}$  is labeled with  $\{result_1\}$ ,  $r_{2, \varepsilon}$  with  $\{result_2\}$  and  $s_{full}$  with  $\{full\}$ . All other states have the empty set as label.

An instance of this case study is identified by maximal stack size and the number of product types and results. We denote it by  $LIFO_{stackSize, numTypes, numResults}$ .  $rate_a$  and  $rate_r$  are determined randomly by first choosing each function value uniformly from  $[0, 1]$  and then normalizing to  $arrivalRate$  or  $processRate$  respectively. This way, the minimization results are unlikely to be skewed by any unwanted structure of the rates

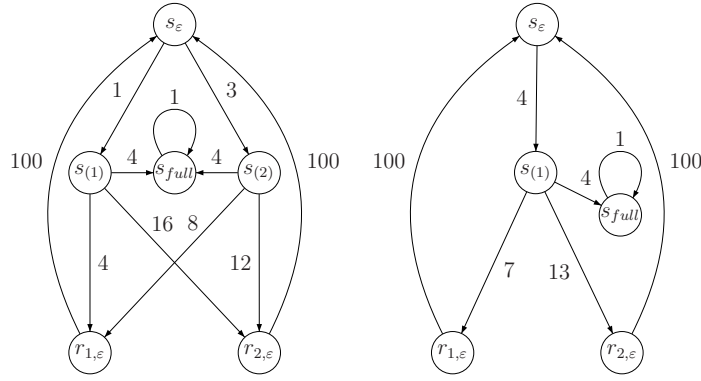


Figure 6.1:  $LIFO_{1,2,2}$  and its minimization.

in contrast to manually choosing them.

Table 6.2 shows the minimization results for different parameters. It compares the minimization using bisimulation to the one that can be achieved using MTME and a repeated application of MTME. Bisimulation does not achieve any minimization for any of the CTMCs while MTME can achieve minimization between 20 and 40 percent. The minimization is several orders of magnitude faster for the same number of states compared to the first case study. The time given for repeated minimization is the total time, i.e. it includes the time for the first run of the minimization algorithm.

	#states	bisimulation		MTME			repeated MTME		
		#classes	factor	time	#classes	factor	time	#classes	factor
$LIFO_{3,5,3}$	249	249	1	.88s	199	.80	1.03s	199	.80
$LIFO_{4,5,3}$	1249	1249	1	33.5s	749	.60	34.2s	749	.60
$LIFO_{5,5,3}$	6249	6249	1	125s	4919	.79	222s	4899	.78
$LIFO_{6,5,3}$	31249	31249	1	5427s	26784	.85	9960s	23579	.76
$LIFO_{4,6,3}$	2332	2332	1	4.3s	1612	.69	8.7s	1612	.69

Table 6.2: Results of  $LIFO$  runs.

We can conclude from this case study that, contrary to the results of the first case study, there exist system models where MTME can achieve substantially more minimization than bisimulation in an acceptable runtime.

Repeated minimization using MTME does not always lead to better results, but also should not be dismissed as in some cases it improves the minimization by up to 12 percent. In those cases where no further reduction was possible, the additional time was spent establishing that a fixpoint had been reached. In all other cases a fixpoint was reached in one additional run. This suggests that we do just one additional round and not check if a fixpoint has been reached. This way we cut down on the runtime of repeated MTME minimization.

In the first case study, one invocation of the algorithm for Part 5 hugely dominates

the runtime of the whole algorithm and there are only a total of six invocations for the smaller instance  $\mathbf{P2P}_{2,4}$ . This case study shows a completely different picture: The smallest instance,  $\mathbf{LIFO}_{3,5,3}$ , which is comparable in size to  $\mathbf{P2P}_{2,4}$ , has already 59 invocations of Part 5 and none of the invocations dominates the runtime. The same situation can be found in all other instances. For the larger ones, the runtime of invocations ranges between a few milliseconds and a few seconds, where the latter are rare. For  $\mathbf{LIFO}_{4,5,3}$ , which has a total runtime of 33.5s, these invocations amount to 27 seconds. For the same instance, checking for wr-consistency in Part 2 takes 0.4 seconds and splitting the subclasses in Part 4 5 seconds. Also for these parts we have many very short runtimes and a few longer ones.

The memory consumption has again be approximated by the peak memory consumption of the java process. Although this is not an optimal measure, it still is a solid upper-bound for the memory consumption of the algorithm.

$\mathbf{LIFO}_{i,5,3}$  for  $i = 3, 4, 5, 6$  have memory consumptions of 253MB, 458MB, 490MB and 916MB. Each of these CTMCs has about 5 times more states than the previous one. So this data strongly suggests that the rise in memory consumption is at most linear in the number of states.

# Chapter 7

## Conclusion

In Chapter 3, after informally describing the original Markovian testing equivalence (MTE) developed by Bernardo, we translated a characterization of it to CTMCs with the help of cylinder sets. Unfortunately, it turned out that MTE is not suitable as a minimization relation, as it does not put all states into relation which should be merged. The problem is that MTE is originally an equivalence between two states and not an equivalence relation on the state space. So it does not explicitly define what happens deeper into the CTMC, where we need to put sets of states into relation with each other in order to get a minimization relation.

We solved this problem by developing Markovian testing minimization equivalence (MTME). We managed to avoid an equivalence relation on the power set of the state space  $2^S$  by splitting each equivalence class into subclasses. In other words, we were able to limit the equivalence relation on  $2^S$  to single equivalence classes. For MTME it was then possible, in contrast to MTE, to define a quotient CTMC and thus minimize the original CTMC.

In preparation to show which properties are preserved by the MTME quotienting construction we defined equivalence under MTME between CTMCs and showed that original CTMC and quotient are equivalent.

Furthermore, we compared MTE and bisimulation with the result that mt-equivalence implies bisimilarity but not every bisimulation is an MTE. In other words, when comparing two states, MTE is stronger than bisimulation but this does not extend to the equivalence relations on the state space. In contrast, every bisimulation is an MTME. This is an important result, as our goal in this thesis was to develop a minimization technique which can achieve at least as much state space reduction as bisimulation. Also, we showed that MTME can be strictly coarser than bisimulation. So we do not simply match the minimization potential of bisimulation, but can improve on it.

We investigated which properties are preserved by MTE and MTME in Chapter 4. First, we argued, based on an example, that branching properties cannot be preserved in general since the quotient under MTME modifies the branching structure of CTMCs. In fact, this modification is essentially the leverage point which allows us to surpass the minimization potential of bisimulation. We then showed in several steps that time-bounded reachability is preserved by MTME and argued that the proof can be easily extended to time-bounded until. Time-bounded reachability is a prop-

erty which is often used in real-world applications of model checking.

We developed a quotienting algorithm for MTME in Chapter 5, which is based on partition refinement. The refinement step has to operate on  $2^S$ , so it becomes combinatorially more costly than the one known from bisimulation. Also, since there can be multiple maximal MTME for any given CTMC, this algorithm has to include a non-deterministic choice.

We could not only show correctness and termination of the algorithm, but also a result which is similar to completeness. Namely, that for every CTMC  $\mathcal{M}$  and MTME  $\sim$  on  $\mathcal{M}$ , the algorithm can output an MTME which is coarser than  $\sim$ .

In Section 5.2 we presented an algorithm which heuristically determines a non-deterministic choice in the quotienting algorithm, which promises a good reduction in the state space. It is based on Knuth's Algorithm X with Dancing Links, which had to be heavily modified in order to include the additional constraints and a target function. It had to be taken great care to not diminish the performance advantages of Dancing Links.

An implementation of both algorithms served as a basis for two case studies, presented in Chapter 6. The model of a simple peer-to-peer protocol proved to be a worst-case scenario. MTME could not achieve a reduction of the state space beyond the one achievable by bisimulation. Also the runtime of the algorithm was several orders of magnitude slower than bisimulation. The second case study showed that MTME can achieve state space reductions that go beyond what is possible with bisimulation. The quotienting algorithm still takes more time than the one for bisimulation but the runtime for similarly sized instances is several orders of magnitude lower than in the first case study. Most importantly we achieve state space reductions of 20 to 40 percent more than bisimulation.

## 7.1 Further work and open questions

With time-bounded reachability we have already seen an important property which is preserved by MTME. However it is desirable to identify more.

Metric temporal logic (MTL), as for example described in [OW08], seems to be a good candidate, as it is a timed and probabilistic path logic. MTL can be defined using continuous or pointwise semantics. Using pointwise semantics MTL formulae are interpreted over timed words  $(A_1, \tau_1)(A_2, \tau_2) \dots$  where  $A_i \subseteq AP$  and  $\tau_i \in \mathbb{R}_0^+$ , when translated to CTMCs as used in this thesis. A timed word describes a path:  $A_i$  is the labeling of the  $i$ -th state and  $\tau_i$  the time which has passed until the  $i$ -th state is reached. Timed words bear a certain resemblance to extended traces. MTE can be characterized as extended trace equivalence; while with MTME we have to take greater care and compare subclasses, it is essentially also extended trace equivalence. This suggests that MTL, at least using the pointwise semantics, is preserved by MTE and MTME.

We can also look at it from a different angle: We have seen that the minimization potential of MTME stems from modifying the branching behavior of the CTMC. MTL formulae however do not examine branching behavior, so this suggests as well that MTL might be preserved by MTE and MTME.

Bernardo shows in [Ber07a] that stationary probabilities are preserved by the original version of MTE. It is likely that this extends to MTE and MTME on CTMCs as presented in this thesis. This would be a very desirable result as it adds to the properties which are preserved by MTME.

Additional open questions and avenues for further work include:

- Should the need arise, extending MTME to CTMCs with actions should be straightforward. In fact, Bernardo defines his Markovian testing equivalence on a process calculus which has actions but no labeling.
- MTME has been constructed as the best possible translation of MTE to an equivalence on the state space. It might be possible to define a reduced version of MTME which cannot achieve the same reductions but is easier to compute. Similarly the algorithm tries to compute the best possible MTME in terms of reduction. By giving up the demand that an algorithm should be able to compute all MTMEs and instead concentrating on easily computable ones, we might be able to drastically improve on the runtime.
- Incorporating MTME minimization into the generation of a CTMC is possible if we can determine if all successors and predecessors of a state have been generated. In this case we can decide if two states can be merged right away. The interesting question remains at which point we actually merge states, since merging too early can prevent minimization opportunities at a later point. So some strategy would be needed to find an optimal time for the merging.



# Bibliography

- [ADD00] Robert B. Ash and Catherine A. Doléans-Dade. *Probability & Measure Theory*. Academic Press, second edition, 2000.
- [Ber07a] Marco Bernardo. Non-bisimulation-based Markovian behavioral equivalences. *J. Log. Algebr. Program.*, 72(1):3–49, 2007.
- [Ber07b] Marco Bernardo. A survey of Markovian behavioral equivalences. In *Formal Methods for Performance Evaluation, 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*, volume 4486 of *Lecture Notes in Computer Science*, pages 180–219. Springer, 2007.
- [BHHK03] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 29(7):2003, 2003.
- [BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, May 2008.
- [Coh03] Bram Cohen. Incentives build robustness in bittorrent. Technical report, bittorrent.org, 2003.
- [DP03] Josée Desharnais and Prakash Panangaden. Continuous stochastic logic characterizes bisimulation of continuous-time Markov processes. *J. Log. Algebr. Program.*, 56(1-2):99–115, 2003.
- [KKZJ07] Joost-Pieter Katoen, Tim Kemna, Ivan Zapreev, and David N. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In O. Grumberg and M. Huth, editors, *Tools and algorithms for the construction and analysis of systems, Braga, Portugal*, volume 4424 of *Lecture notes in computer science*, pages 87–101, Berlin, 2007. Springer.
- [KNP06] Marta Kwiatkowska, Gethin Norman, and David Parker. Symmetry reduction for probabilistic model checking. In *Proc. 18th International Conference on Computer Aided Verification (CAV'06)*, volume 4114 of *LNCS*, pages 234–248. Springer, 2006.
- [Knu00] Donald E. Knuth. Dancing links. In *Millennial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare*, pages 187–214. Palgrave, 2000.

- [OW08] Joël Ouaknine and James Worrell. Some recent results in metric temporal logic. In *FORMATS '08: Proceedings of the 6th international conference on Formal Modeling and Analysis of Timed Systems*, pages 1–13, Berlin, Heidelberg, 2008. Springer-Verlag.
- [PRI09] PRISM. Case studies - simple peer-to-peer protocol. <http://www.prismmodelchecker.org/casestudies/peer2peer.php>, 2009. [Online; accessed 7-December-2009].