Model Checking Algorithms for Markov Reward Models

Lucia Cloth

CTIT Ph.D.-thesis Series No. 06-81
Centre for Telematics and Information Technology
University of Twente, P.O. Box 217, NL-7500 AE Enschede
ISSN 1381-3617

Printing: Verlagshaus Mainz GmbH, Aachen, Germany

ISBN 90-365-2317-6

MODEL CHECKING ALGORITHMS FOR MARKOV REWARD MODELS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. W.H.M. Zijm,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 13 januari 2006 om 16.45 uur

door

Lucia Cloth

geboren op 16 september 1976
te Aachen, Duitsland

Dit proefschrift is goedgekeurd door
de promotor, prof. dr. ir. B.R. Haverkort.

Abstract

Model checking Markov reward models unites two different approaches of model-based system validation. On the one hand, Markov reward models have a long tradition in model-based performance and dependability evaluation. On the other hand, a formal method like model checking allows for the precise specification and verification of complex qualitative system properties. The logic CSRL (an extension of CTL) provides the specific means to formulate desired properties of Markov reward models, including constraints on time and accumulated reward. The most involved operator of CSRL is the so-called until operator with quantitative constraints in the form of a time and a reward interval. Its model checking is closely connected to the computation of the distribution of accumulated reward in the Markov reward model. So far, suitable numerical algorithms have only been published for the restricted case where the time and the reward interval have lower bounds equal to zero. In this thesis we close the gap and present the theoretical basis as well as the numerical algorithms needed for model checking CSRL until formulas with arbitrary time and reward intervals.

CSRL is useful for the assessment of many interesting properties, for example, the survivability of an information or communication system. A system is survivable if it is able to recover properly after is has been affected by a disaster. We translate this general definition of survivability into a CSRL formula which then can easily be instantiated for the system model under study.

The basic building blocks for CSRL formulas are atomic propositions that are assigned to the states of a Markov reward model. We extend CSRL to pathCSRL, where we can reason about the actions occurring in a Markov reward model as well. The logic pathCSRL additionally includes regular expressions as a more flexible mechanism for defining path properties and, hence, effectively extends the expressive power of CSRL.

Throughout the thesis we illustrate all concepts and techniques with a small running example. A number of larger case studies are also provided.

Thanks to all who made it possible.

Contents

Chapter 1

Introduction

Information and Communication Technology (ICT) has become an essential part of the wheelwork of economical and social life. The smooth operation of many important systems, like stock exchanges, the coordination of train traffic, the monitoring in intensive care, or military devices, depends heavily on the correct operation of ICT. It is therefore crucial to design and deliver ICT systems that are proven to meet their demands.

The title of this thesis is "Model checking algorithms for Markov reward models". In the following we explain the terms appearing in this title and explore how they can be utilised for the evaluation of ICT systems.

In the design and planning phase of ICT systems it is not possible to detect flaws or to prove desired properties of the system itself, simply because it does not yet exist. However, even systems that are already operating might not be prepared for non-interfering testing, measurement or monitoring. In this case, *model*-based techniques can help [47]. A model is a (usually miniature) representation of something real. For evaluation purposes we want such a model to be an abstract, mathematically precise and unambiguous description of the relevant features of some real world system. Naturally, a model never reflects all features and properties of reality. Any conclusion drawn from the evaluation of a model can only be as good as the model itself. It is essential that the model includes enough detail to provide useful results but abstracts from less important aspects.

Stochastic models include several important aspects of real-world systems: the relevant states, the transitions between them, the time passing between the occurrence of transitions and a (probabilistic) way of determining the following state. A special class of stochastic models are continuous-time *Markov* chains (CTMCs) [100]. They have a discrete state space, the notion of time is continuous and state changes depend exclusively on the current state. CTMCs have been used intensively for performance and dependability evaluation, that is, the quantitative assessment of system behaviour. They are well understood and mathematically attractive while at the same time flexible enough to represent complex system structures. It is not necessary to specify a CTMC manually at state level. A wide variety of high-level modelling formalisms exists, such as queueing networks [29], stochastic process algebra [54, 58] or stochastic Petri nets [1]. Software tools exist that automatically generate the underlying CTMC and facilitate performance evaluation [16, 35, 93, 92, 55].

Markov *reward* models were introduced by Howard [60]. He attached yield rates to the states of a model to express the accumulation of gain or cost. Later, reward rates were given a slightly different meaning. Haverkort and Trivedi [53] distinguish between three different uses of rewards. Firstly, they can express the performance level of each state of a system, for example, the number of customers at a resource. Secondly, they can be employed for dependability analysis. In this case the reward rates are 0 or 1, depending on whether the overall system is available or not. Finally, the reward rates can support a mixed performance and dependability (performability) [73] evaluation. It is then assumed that the modelled system can also provide partial service, depending on its state. The reward rates express the chosen level of performance and operationality. In the following we interpret reward rates as an integral part of the models, as it was originally intended by Howard, and not as a mere vehicle for the assessment of performance and/or dependability.

Model checking [23] is a verification technique that decides for every single state of a model whether it does or does not satisfy a given property. Model checking is a formal method, thus it is useful for validating system correctness on a sound mathematical base. The property that needs to be checked is expressed as a formula of a logic. This logic must be suited to phrase important properties of the considered model class. For Markov reward models, the continuous stochastic reward logic (CSRL) [6, 50] has been introduced. It is an extension of the continuous stochastic logic (CSL) [3, 8] for CTMCs which in turn is an extension of the branching time temporal logic CTL [21].

For each model class and property description logic, tailored *algorithms* are needed that perform the actual model checking without further human intervention. To obtain trustworthy results, the underlying mechanisms and procedures should be proven to be correct. Only algorithms that are efficient regarding both, space and time, allow for the evaluation of models that are detailed enough for the adequate representation of a real world system.

This thesis deals with algorithms for CSRL model checking of MRMs. It brings together the world of performance and dependability evaluation using MRMs and the world of model checking.

In Chapter 2 we recapitulate the mathematical definition of CTMCs and their extension to labelled MRMs where states have associated reward rates and are labelled with so-called atomic propositions. Special attention is given to the probability space consisting of all paths of an MRM. We also introduce a small example that is going to illustrate many concepts throughout the following chapters.

Basic measures of MRMs are the topic of Chapter 3. We show how the transient and the steady-state distribution over the states of an MRM are derived and how they can be computed efficiently. We also characterise the joint transient distribution of state and accumulated reward by a system of partial differential equations.

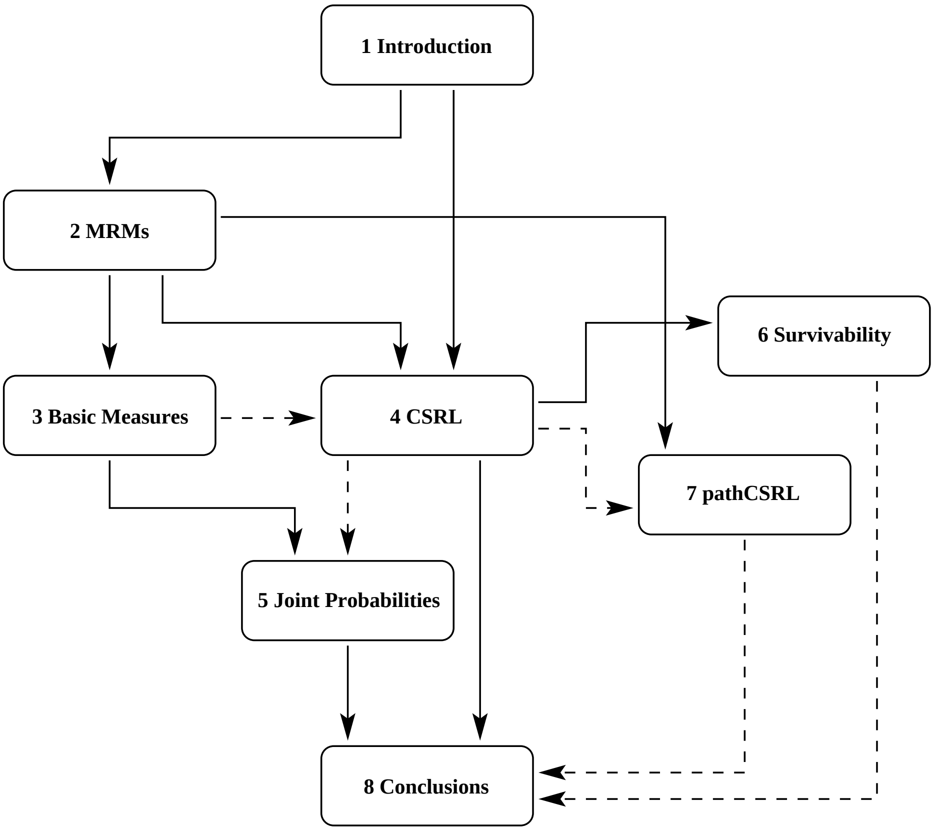
Where Chapters 2 and 3 describe MRMs and measures as traditionally used in performance modelling, with Chapter 4 we enter the world of model checking. It presents the logic CSRL which is tailored to describe properties of MRMs. For each operator of the logic we discuss how its model checking is performed. Most involved is the checking of the so-called time- and reward-bounded until operator. It relies on the calculation of the joint probability of state and accumulated reward. Numerical algorithms for the computation of the joint distribution are presented in Chapter 5. We discuss the complexity of each of the algorithms and compare their results with respect to accuracy and run time.

In Chapter 6 we show how CSRL model checking can practically be employed to assess the survivability of an ICT system after it has been affected by a disaster. Starting from a common and intuitive definition of survivability, we derive a pattern for a CSRL survivability formula that can easily be adapted and then checked for a specific model. The approach is illustrated by a case study featuring the survivability of the Google file system.

When using a high-level language for the specification of MRMs, action labels for transitions arise naturally. In Chapter 7 we present the logic pathCSRL which allows reasoning not only about state properties but also about transition labels. It further extends CSRL by more complex behaviour descriptions using regular expressions. A case study completes the chapter. It demonstrates the expressive power of pathCSRL by means of a model of the GSM handover procedure.

In Chapter 8 we summarise the contents of this thesis, discuss what has been achieved and give some pointers for further work.

The following figure gives an overview of the chapters and their relation. A solid arrow from X to Y indicates that the content of Chapter X is a prerequisite to understand Chapter Y. A dashed arrow suggests a less strong relation.



Chapter 2

Markov reward models

In this chapter we describe the class of models used throughout this thesis. A Markov reward model is a continuous-time Markov chain with state labels and a reward vector containing state reward rates. We therefore start in Section 2.1 with a short introduction to continuous-time Markov chains and then discuss some of their basic properties (Sections 2.2 – 2.4). We proceed with a detailed description of the probability space formed by the realisations of a CTMC in Section 2.5. CTMCs are extended with state labels (Section 2.6) and reward rates (Section 2.7) in order to obtain labelled Markov reward models. For reference throughout the following chapters we present a small running example in Section 2.9.

2.1 Continuous-time Markov chains

A continuous-time Markov chain (CTMC) is a stochastic process $(X_t, t \in \mathbb{R}_{\geq 0})$ with continuous parameter $t > 0$. The random variables X_t assume values from a finite and discrete set, the *state space*. We can think of arbitrary names for states, but they can always be mapped to a set $S = \{0, \ldots, N-1\}$. The continuous parameter t is interpreted as *time*. It ranges over all non-negative real numbers. We say that the CTMC resides in state $s \in S$ at time t if the realisation of X_t is s. After some time there might be a *transition* into another state.

A CTMC is specified by its *transition rates* $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$. Rate $R_{ss'}$ denotes the number of transitions that occur from state s to state s' per unit of time. If $\mathbf{R}_{ss'} = 0$, no transition is possible from s to s'. Note that we do not forbid transitions back to the same state. The total *exit* rate E_s of a state s is the sum of rates of transitions starting at state s:

$$E_s = \sum_{s' \in S} \mathbf{R}_{ss'}.$$

If a state s has no outgoing transitions, that is $E_s = 0$, then it is called *absorbing*: the CTMC is trapped in state s once having entered it. Whenever necessary, we divide the state space S into two disjoint sets: S_a for the absorbing states and S_{na} for the non-absorbing states.

A CTMC obeys the *Markov property*: the future realisation of X_t is independent of the past, given the present. Formally, for any $t_1 < \ldots < t_n < t_{n+1} \in \mathbb{R}_{\geq 0}$ and $s_1, \ldots, s_n, s_{n+1} \in S$,

$$\Pr\left\{X_{t_{n+1}} = s_{n+1} \mid X_1 = s_1, \ldots, X_n = s_n\right\} = \Pr\left\{X_{t_{n+1}} = s_{n+1} \mid X_n = s_n\right\}.$$

We have described *time-homogeneous* CTMCs, where the probability of moving from one state to another depends only on the length of the time interval in between and not on the current instant of time:

$$\Pr\{X_{t+t'} = s' \mid X_t = s\} = \Pr\{X_{t'} = s' \mid X_0 = s\},$$

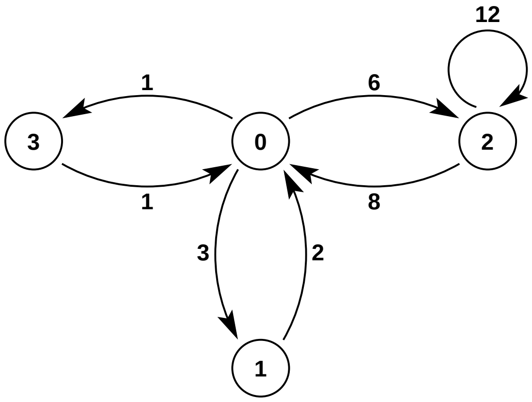
for any $t, t' \in \mathbb{R}_{\geq 0}$.

Graphical representation. CTMCs can be represented as directed graphs. States are represented by circles with the state names written inside. Arrows between circles denote the transitions. Near the arrows one can find the transition rates.

Example 1. We describe a simple CTMC that will serve as a running example. The CTMC has four states, $S = \{0, 1, 2, 3\}$. The transition rates $\mathbf{R}$ are conveniently written as a matrix:

$$\mathbf{R} = \begin{pmatrix} 0 & 3 & 6 & 1 \\ 2 & 0 & 0 & 0 \\ 8 & 0 & 12 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

The following figure gives the graphical representation of the CTMC.



There are transitions from state 0 to states 1, 2, and 3. States 1 and 3 have exactly one possible successor, state 0. State 2 has additionally a transition leading back to itself. The CTMC has no absorbing states. ♢

Since the state space together with the rate matrix completely defines the behaviour of the stochastic process we say that a pair $(S, \mathbf{R})$ is a CTMC.

2.2 Residence times

The *residence time* (or *sojourn* time) δ_s of a state s is the time that goes by until the next transition is taken. For non-absorbing states, the residence time is not a fixed value but an exponentially distributed random variable with parameter E_s. It has been shown that as a consequence of the Markov property the residence times are negative exponentially distributed, that is, the probability of residing for at most (or less than) $t \in \mathbb{R}_{\geq 0}$ time units in state $s \in S_{na}$ is

$$\Pr\{\delta_s \leq t\} = \Pr\{\delta_s < t\} = 1 - e^{-E_s t}.$$

Intervals are special subsets of $\mathbb{R}^{\infty}_{\geq 0}$, written as $[a,b]$, $[a,b)$, $(a,b]$ or (a,b) for $a,b \in \mathbb{R}^{\infty}_{\geq 0}$. The collection of all intervals in $\mathbb{R}^{\infty}_{\geq 0}$ is denoted by $\mathbb{I}$. Note that $\varnothing$ is also an interval in $\mathbb{I}$, since, for example, $(0,0) = \varnothing$.

The probability that the sojourn time of $s \in S_{na}$ falls in a given interval $I \in \mathbb{I}$ is

$$\begin{aligned} \Pr\{\delta_s \in I\} &= \Pr\{\delta_s \leq \sup I\} - \Pr\{\delta_s \leq \inf I\} \\ &= (1 - e^{-E_s \sup I}) - (1 - e^{-E_s \inf I}) \\ &= e^{-E_s \inf I} - e^{-E_s \sup I}. \end{aligned}$$

For $\sup I = \infty$ we set $e^{-E_s \cdot \infty} = 0$.

The sojourn time of an absorbing state is always ∞. Thus, the probability of the sojourn time of an absorbing state falling in an interval I is 1 if $\infty \in I$ and zero if $\infty \notin I$.

For all states $s \in S$ we have $\Pr\{\delta_s \in [0,\infty]\} = 1$ and $\Pr\{\delta_s \in \varnothing\} = 0$.

Example 2. Consider again the CTMC of Example 1. The sojourn time of state 0 is exponentially distributed with parameter $E(0) = 10$. The probability of its sojourn time falling in the interval $[0.15, 0.3)$ is given by

$$\begin{aligned} \Pr\{\delta_0 \in [0.15, 0.3)\} &= e^{-10 \cdot 0.15} - e^{-10 \cdot 0.3} \\ &= 0.2231 - 0.04979 \\ &= 0.1733. \end{aligned}$$

For state 1 we have

$$\begin{aligned} \Pr\{\delta_1 \in [0,1)\} &= e^{-2 \cdot 0} - e^{-2 \cdot 1} \\ &= 1 - 0.1353 \\ &= 0.8647. \end{aligned}$$ ◊

2.3 One-step probabilities

The sojourn time of a non-absorbing state s is exponentially distributed with parameter E_s. If the successor state s' is already known, the sojourn time is still exponentially distributed, but now with parameter $R_{ss'}$. When the CTMC enters a state, for each possible successor state there is a negative exponentially distributed random variable. These random variables are in a race condition. The minimum of these distributions wins. The minimum of several exponential distributions is again an exponential distribution with the sum of the rates of the contributing distributions as parameter. When residing in state s, the probability that the exponential distribution with rate $R_{ss'}$ is the minimum is given by

$$P_{ss'} = \frac{R_{ss'}}{E_s}.$$

We call this probability the *one-step probability* from s to s'. It is the probability that s' is the successor state of s.

Example 3. For state 0 of the CTMC of Example 1, the total outgoing rate is $E_0 = 3 + 6 + 1 = 10$. The one-step probabilities from state 0 are then

$$P_{00} = 0, \quad P_{01} = \frac{3}{10}, \quad P_{02} = \frac{6}{10}, \quad P_{03} = \frac{1}{10}.$$

The complete one-step probability matrix is given by

$$\mathbf{P} = \begin{pmatrix} 0 & \frac{3}{10} & \frac{6}{10} & \frac{1}{10} \\ 1 & 0 & 0 & 0 \\ \frac{4}{10} & 0 & \frac{6}{10} & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

◊

2.4 The initial distribution

The transition rates defined by $\mathbf{R}$ characterise the evolution of the CTMC over time. One detail is still missing for the *complete* description of the stochastic process. The generator matrix does not reveal in which state the process should start.

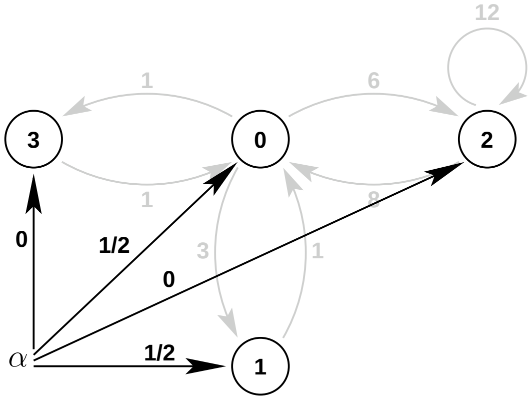
This gap is filled by means of an *initial distribution*. Formally, this is a probability vector $\alpha \in [0,1]^{|S|}$ that maps each state to the probability that the CTMC occupies it at time $t_0 = 0$. The sum over all these probabilities has to be one,

$$\sum_{s \in S} \alpha_s = 1.$$

The special initial distribution, where we have a fixed starting state $s \in S$, is denoted $\alpha^{(s)}$:

$$\begin{aligned} \alpha_s^{(s)} &= 1, \\ \alpha_{s'}^{(s)} &= 0, \quad s' \neq s. \end{aligned}$$

Example 4. The following figure depicts an initial distribution for the running example.



States 0 and 1 are initial states each with probability $\frac{1}{2}$. The initial probability for states 2 and 3 is 0:

$$\alpha = \left(\frac{1}{2}, \frac{1}{2}, 0, 0 \right).$$

If we want state 0 as the only starting state, we get the initial distribution $\alpha^{(0)} = (1, 0, 0, 0)$. ◊

2.5 The probability space on paths

A CTMC is a collection of random variables X_t, each describing the likely state of the stochastic process at a given time t. Looking at a particular realisation of all these random variables we get one single instantiation of the stochastic process. Such an instantiation is called a *path*. For each $t \geq 0$, it indicates the exact current state of the CTMC. We can see a path also as an enumeration of states, in the order they are visited. Each state has attached its sojourn time. Depending on the type of states visited, a path is either a finite or an infinite sequence. If the CTMC visits only non-absorbing states, a path is an infinite sequence of pairs consisting of a state and its sojourn time. The set of infinite paths is then

$$\{((s_0, t_0), (s_1, t_1), \cdots \mid (s_i, t_i)) \in (S_{na} \times \mathbb{R}_{\geq 0}) \text{ for all } i \in \mathbb{N}_{>0}\} .$$

The *length* of an infinite path ω is $|\omega| = \infty$. In the case that the CTMC hits an absorbing state, it can only visit a finite number of non-absorbing states before. The set of finite paths is

$$\bigcup_{n=0}^{\infty} \{((s_0, t_0), (s_1, t_1), \cdots (s_{n-1}, t_{n-1})(s_n, \infty)) \mid (s_i, t_i) \in (S_{na} \times \mathbb{R}_{\geq 0}) \text{ for } i = 1, \cdots, n-1, s_n \in S_a\} .$$

The length of such a finite path is given by the number of transitions involved, that is, $|\omega| = n$. For $i \leq |\omega|$, the ith state of any ω can be accessed by $s_i(\omega) = s_i$. The residence time of the ith state is given by $t_i(\omega) = t_i$. We denote the set of all infinite and finite paths with Ω, for a single path we use the letter ω.

Example 5. For the CTMC of Example 1,

$$\omega_1 = (0, 0.25), (1, 0.25), (0, 0.25), (1, 0.25), (0, 0.25), (1, 0.25), \cdots$$

is the infinite path where after a residence time of 0.25 time units in state 0 there is always a transition to state 1 and after again 0.25 time units there is a transition back to state 0. The CTMC has no absorbing states and so there are no finite paths. ♢

A path fixes the behaviour of the CTMC throughout the complete time line. For every $t \geq 0$ it determines the realisation of the random variable $X_t : \Omega \to S$. For $\omega = (s_0, t_0), (s_1, t_1), \cdots \in \Omega$ an infinite path,

$$X_t(\omega) = s_n \iff \sum_{i=0}^{n-1} t_i \leq t \text{ and } \sum_{i=0}^{n} t_i > t.$$

$X_t(\omega)$ is defined to be the state the CTMC occupies on path ω at time t. The value of $X_t(\omega)$ is the unique s_n which is entered before or at t ($\sum_{i=1}^{n-1} t_i \leq t$) and left after t ($\sum_{i=1}^{n} t_i > t$).

For a finite path ω, the outcome of $X_t(\omega)$ is defined similarly.

Example 6. Continuing the previous example, we have $X_{1.1}(\omega_1) = 0$. ♢

The random variables X_t are defined over a shared probability space (consisting of a universe, a σ-algebra and a probability measure) and map into the state space S of the CTMC. The universe of the probability space is the set of all paths Ω. In the following we derive a σ-algebra over Ω and a probability measure. Together they correctly describe the probability of paths of a CTMC specified by its rate matrix $\mathbf{R}$ and its initial distribution α.

2.5.1 Cylinder sets

The first step is to identify those subsets of Ω that generate the σ-algebra. They are defined by imposing restrictions on the first $n \in \mathbb{N}$ states of a path. Restrictions are made in two ways: first we fix a state sequence $s_0, s_1, \cdots, s_n$ such that there are transitions between consecutive states, i.e., $\mathbf{R}(s_i, s_{i+1}) > 0$ for $i = 1, \cdots, n-1$. Then we confine the sojourn time for each state s_i in the sequence by providing an interval $I_i \in \mathbb{I}$.

Let $\sigma = (s_0, s_1, \cdots s_n) \in S^n$ be a state sequence with $\mathbf{R}(s_i, s_{i+1}) > 0$ for $i = 0, \cdots, n-1$, and let $\mathcal{I} = I_0 \times \cdots \times I_n \in \mathbb{I}^n$ be a subset of $(\mathbb{R}^{\infty}_{\geq 0})^n$. Then the corresponding *cylinder set* $C(\sigma, \mathcal{I})$ is defined as

$$C(\sigma, \mathcal{I}) = \{((s_0, t_0), (s_1, t_1), \cdots, (s_n, t_n), \cdots) \in \Omega \mid t_i \in I_i \text{ for } i = 0, \cdots, n\}.$$

Note, that $C(\sigma, \mathcal{I})$ includes infinite as well as finite paths.

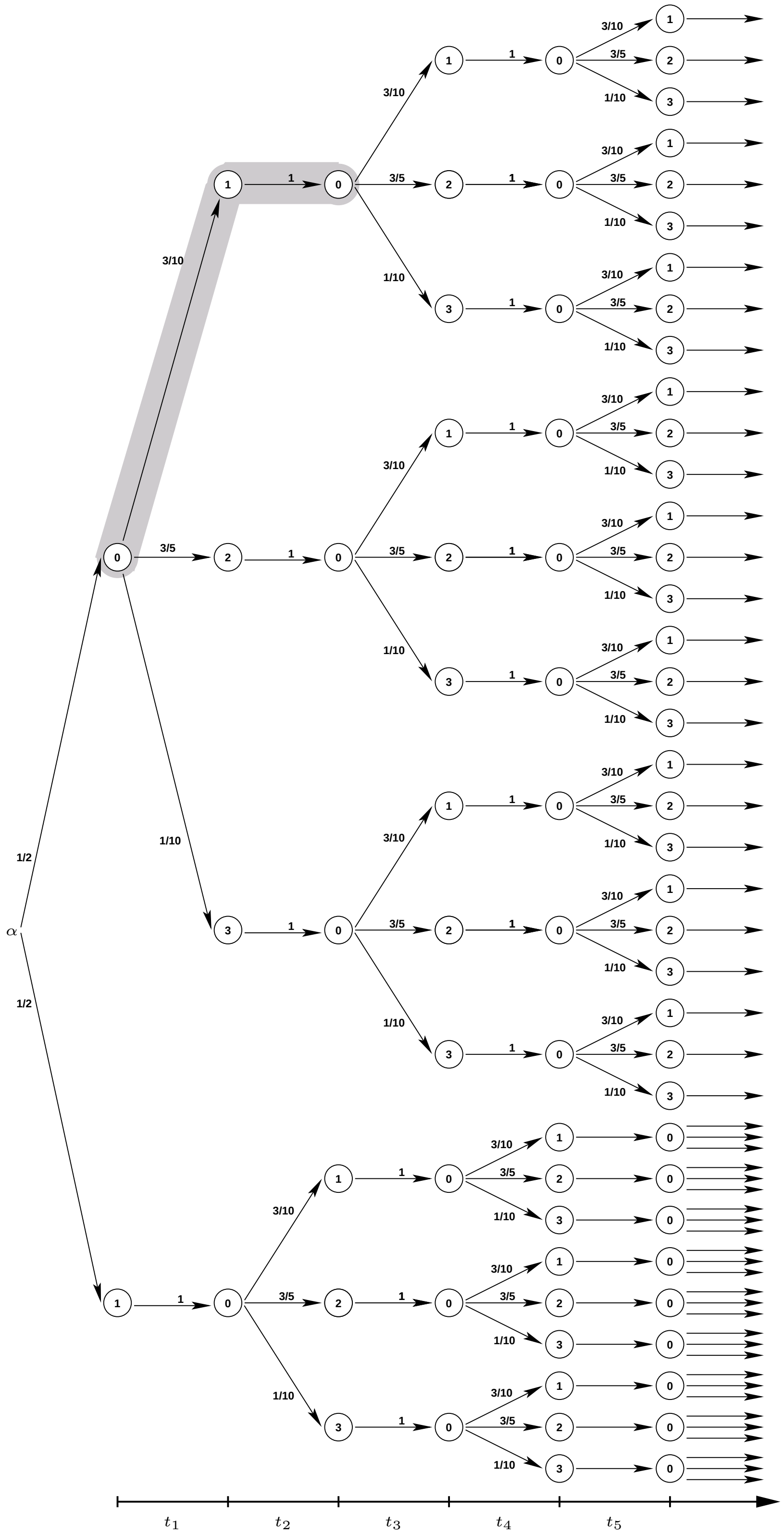


Figure 2.1: The tree of paths for the CTMC of Example 1

Example 7. Figure 2.1 visualises the set of paths of the CTMC in our running example. Only paths starting in state 0 or 1 are considered. These are the states that have a positive initial probability (cf. Example 4). After some time t_1 the CTMC jumps to another state. It has to be reachable from the first state via a transition, hence state 0 has states 1, 2 and 3 as possible successors while state 1 has only one possible successor, namely state 0. Therefore there are three branches outgoing from state 0 and only a single arrow leaving state 1. Like this, the possible sequences of states are unfolded transition by transition.

Now look at the state sequence $\sigma_1 = (0, 1, 0)$. The corresponding part of the path tree is shaded in Figure 2.1. When restricting the first three states to this sequence, only paths starting with the shaded part are considered. If we additionally specify intervals for the sojourn time of the first three states, we get a cylinder set. Let $\mathcal{I}_1 = [0.15, 0.3) \times [0, 1] \times (0.24, 0.26)$. The cylinder set $C(\sigma_1, \mathcal{I}_1)$ contains all paths, that first reside in state 0 for a time in $[0.15, 0.3]$, then in state 1 for at most 1 time unit, then again in state 0 with a sojourn time between 0.24 and 0.26. The further evolution of the paths is not restricted, any valid finite or infinite succession of states might occur. Path ω_1 of Example 5 is an element of $C(\sigma_1, \mathcal{I}_0)$. $\Diamond$

When there is no restriction on the state sequence and residence times at all we obtain the cylinder set

$$C() = \Omega.$$

We define $\mathcal{C}$ to be the collection of all cylinder sets. With $\sigma(\mathcal{C})$ the σ-algebra generated by $\mathcal{C}$, we obtain the measurable space $(\Omega, \sigma(\mathcal{C}))$. The following section shows the construction of a suitable probability measure on this space.

2.5.2 Probability measure on paths

The second step in constructing the probability space is the definition of a probability measure.

This section makes use of some concepts of real analysis without digging into details. The reader might refer to [13, 38] for further information. For the construction we proceed as follows. We show that the collection of all cylinder sets $\mathcal{C}$ is a semi-ring. As a precursor for the final probability measure, we define a countably additive set function that maps cylinder sets to probabilities. This set function extends uniquely to a probability measure on $(\Omega, \sigma(\mathcal{C}))$.

Lemma 1. $\mathcal{C}$ is a semi-ring, that is, it has the following properties.

i) $\varnothing \in \mathcal{C}$.

ii) Let $A = C(s_0, \cdots, s_n, I_0 \times \cdots \times I_n)$ and $B = C(s'_0, \cdots, s'_m, J_0 \times \cdots \times J_m)$ be cylinder sets. Their intersection $A \cap B$ is again a cylinder set.

iii) Let again A and B be two cylinder sets as defined for property 2. Then their difference $A \backslash B$ can be written as the finite union of disjoint cylinder sets. That is, there exist cylinder sets $C_0, \cdots, C_r$ with $C_i \cap C_j = \varnothing$ if $i \neq j$ such that $A \backslash B = \bigcup_{i=1}^{r} C_i$.

Proof.

i) $C((s),\varnothing)$ is an example of a cylinder set that equals $\varnothing$ for every $s \in S$.

ii) Without loss of generality, assume that $n \geq m$. If $s_0,\cdots,s_m \neq s'_0,\cdots,s'_m$, the intersection of A and B is the empty set and thus a cylinder set. If $s_0,\cdots,s_m = s'_0,\cdots,s'_m$, the intersection is given by

$$A \cap B = C(s_0,\cdots,s_m,s_{m+1},\cdots,s_n,I_0 \cap J_0 \times \cdots \times I_m \cap J_m \times I_{m+1} \times \cdots \times I_n).$$

Note that the intersection of two intervals is again an interval, possibly the empty one.

iii) If $s_0,\cdots,s_m \neq s'_0,\cdots,s'_m$, the difference $A\backslash B = A$, hence, $r = 0$ and $C_0 = A$. If the sequences are identical, we have to exclude those paths from A that have sojourn times allowed in B. We set $J_{m+1} = \cdots = J_n = [0,\infty]$ because the set B imposes no restriction on the sojourn time of the states $s_{m+1},\cdots,s_n$. The collection of products of intervals $(\mathbb{I}^n)$ is itself a semi-ring. It follows that there exist disjoint interval sequences $\mathcal{K}_0,\cdots,\mathcal{K}_r$ such that

$$(I_0 \times \cdots \times I_n)\backslash(J_0 \times \cdots \times J_n) = \bigcup_{i=1}^{r} \mathcal{K}_i.$$

The set difference is now given by

$$A\backslash B = \bigcup_{i=1}^{r} C(s_0,\cdots,s_n,\mathcal{K}_i)$$

which is a finite union of disjoint basic cylinder sets. □

We want to find a set function on $\mathcal{C}$ that correctly reflects the probability of paths. For a cylinder set C it should evaluate to the probability that the CTMC follows one of the paths in C. It should depend on the basic ingredients of the CTMC: its initial distribution α, the one-step probabilities $\mathbf{P}$ and the sojourn time distributions δ_s. One-step probabilities and sojourn time distributions are determined by the rate matrix $\mathbf{R}$.

We define the function $p_\alpha : \mathcal{C} \rightarrow [0,1]$ as follows.

$$\begin{aligned} p_\alpha(C()) = p_\alpha(\Omega) &= 1 \\ p_\alpha(C(s_0,\cdots,s_n,I_1 \times \cdots \times I_n)) &= \alpha_{s_0} \cdot \prod_{i=0}^{n-1} \mathbf{P}(s_i,s_{i+1}) \cdot \prod_{i=0}^{n} \Pr\{\delta_{s_i} \in I_i\} \end{aligned}$$

It is the product of the initial probability of the first state s_0 of the sequence, the one-step probabilities between consecutive states and the probabilities that the sojourn times fall in the corresponding intervals.

Example 8. In the tree of all paths (Figure 2.1) the branches are labelled with the one-step probabilities. Consider again the cylinder set $C(\sigma_1, \mathcal{I}_1) = C((0,1,0), [0.15, 0.3) \times [0,1] \times (0.24, 0.26))$. For the sojourn times we have $\Pr\{\delta_0 \in [0.15, 0.3)\} = 0.1733$, $\Pr\{\delta_1 \in [0,1]\} = 0.8647$ and $\Pr\{\delta_0 \in (0.24, 0.26)\} = 0.01644$. The probability of $C(\sigma_1, \mathcal{I}_1)$ consequently equals

$$p_\alpha(C(\sigma_1, \mathcal{I}_1)) = \frac{1}{2} \cdot \left(\frac{3}{10} \cdot 1\right) \cdot (0.1733 \cdot 0.8647 \cdot 0.01644) = 0.0003695. \qquad \Diamond$$

Two state sequences of different length might generate the same cylinder set. We therefore require consistency of p_α.

Lemma 2. Let $A = B$ for $A = C(s_0, \cdots, s_n, \mathcal{I})$ and $B = C(s'_0, \cdots, s'_m, \mathcal{J})$. Then

$$p_\alpha(A) = p_\alpha(B).$$

Proof. Without loss of generality assume that $n \geq m$. Equality of the two sets requires that $s_0, \cdots, s_m = s'_0, \cdots, s'_m$ and $I_0 \times \cdots \times I_m = J_0 \times \cdots \times J_m$. In $B = C(s'_0, \cdots, s'_m, \mathcal{J})$ there is no restriction on the states on positions $m+1, \cdots, n$. In $A = C(s_0, \cdots, s_n, \mathcal{I})$ the states on these positions are fixed. The two sets can consequently only be equal if the transition structure of the CTMC determines uniquely the states for these positions. Then the one-step probabilities equal one and do not make a difference between $p_\alpha(A)$ and $p_\alpha(B)$. The sojourn time in B is not restricted at positions $m+1, \cdots, n$. So the sojourn time intervals $I_{m+1}, \cdots, I_n$ at these positions have to equal the interval $[0, \infty]$. The only exception arises if s_n is an absorbing state. Then I_n might be any interval that includes ∞. In either case, the sojourn time probability is always one, and we have

$$\begin{aligned}
p_\alpha(A) &= \alpha_{s_0} \cdot \left(\prod_{i=0}^{m-1} \mathbf{P}(s_i, s_{i+1})\right) \cdot \left(\prod_{i=m}^{n-1} 1\right) \cdot \left(\prod_{i=0}^{m} \Pr\{\delta_{s_i} \in I_i\}\right) \cdot \left(\prod_{i=m+1}^{n} 1\right) \\
&= \alpha_{s'_0} \cdot \left(\prod_{i=0}^{m-1} \mathbf{P}(s'_i, s'_{i+1})\right) \cdot \left(\prod_{i=0}^{m} \Pr\{\delta_{s'_i} \in J_i\}\right) \\
&= p_\alpha(B). \qquad \Box
\end{aligned}$$

Example 9. Consider the state sequences $\sigma_1 = (0,1,0)$ and $\sigma_2 = (0,1)$. In our running example, the only possible successor of state 1 is state 0, and so we have

$$C(\sigma_1, [0.15, 0.3] \times [0,1] \times [0,\infty]) = C(\sigma_2, [0.15, 0.3] \times [0,1])$$

and

$$\begin{aligned}
p_\alpha(C(\sigma_1, [0.15, 0.3] \times [0,1] \times [0,\infty])) &= \frac{1}{2} \cdot \left(\frac{3}{10} \cdot 1\right) \cdot (0.1733 \cdot 0.8647 \cdot 1) \\
&= \frac{1}{2} \cdot \left(\frac{3}{10}\right) \cdot (0.1733 \cdot 0.8647) \\
&= p_\alpha(C(\sigma_2, [0.15, 0.3] \times [0,1])). \qquad \Diamond
\end{aligned}$$

In order for a set function on a semi-ring to extend uniquely to a probability measure on the corresponding σ-algebra, the set function has to be *countably additive.*

Lemma 3. p_α is countably additive, that is,

i) $p_\alpha(\varnothing) = 0$, and

ii) whenever $\{C_i\}$ is a disjoint sequence of sets in $\mathcal{C}$ with $\bigcup_{i=1}^{\infty} C_i \in \mathcal{C}$,

$$p_\alpha\left(\bigcup_{i=1}^{\infty} C_i\right) = \sum_{i=1}^{\infty} p_\alpha(C_i).$$

Proof.

i) If a cylinder set $C(\sigma, I_0 \times \cdots \times I_n) = \varnothing$, there is an $r \in \{0, \cdots, n\}$ such that $I_r = \varnothing$. We have that $\Pr\{\delta_{s_r} \in \mathcal{I}_r\} = 0$ and also the complete product $p_\alpha(\varnothing) = 0$.

ii) Let $C_i = C(\sigma_i, \mathcal{I}_i)$ with $\mathcal{I}_i = I_{i_1} \times \cdots \times I_{i_n}$. Since $\bigcup_{i=1}^{\infty} C_i = C(\sigma, \mathcal{I})$ is a cylinder set with $\mathcal{I} = I_1 \times \cdots \times I_n$, the C_i have to share the same state sequence: $\sigma_i = \sigma$. Additionally, the union of the products of intervals has to be again a product of intervals: $\bigcup_{i=1}^{\infty} \mathcal{I}_i = \mathcal{I}$. Define the set function $\mu_\sigma : \mathbb{I}^n \to [0,1]$ as

$$\mu_\sigma(I_0 \times \cdots \times I_n) = \prod_{j=1}^{n} \Pr\left\{\delta_{s_j} \in I_j\right\}.$$

Then

$$p_\alpha(C(\sigma, \mathcal{I})) = \alpha(s_1) \cdot \prod_{j=1}^{n-1} \mathbf{P}(s_j, s_{j+1}) \cdot \mu_\sigma(\mathcal{I}).$$

By arguments similar to those for the Lebesgue measure [13, 38], μ_σ is countably additive and we have

$$\begin{aligned} p_\alpha(C(\sigma, \mathcal{I})) &= \alpha_{s_0} \cdot \prod_{j=0}^{n-1} \mathbf{P}(s_j, s_{j+1}) \cdot \sum_{i=0}^{\infty} \mu_\sigma(\mathcal{I}_i) \\ &= \sum_{i=0}^{\infty} \alpha_{s_0} \cdot \prod_{j=0}^{n-1} \mathbf{P}(s_j, s_{j+1}) \cdot \mu_\sigma(\mathcal{I}_i) \\ &= \sum_{i=0}^{\infty} p_\alpha(C_i). \end{aligned}$$ □

The following theorem concludes our construction of the probability space $(\Omega, \sigma(\mathcal{C}), P_\alpha)$ underlying a CTMC. Its universe is the set of all paths. The measurable sets are combinations of cylinder sets, that is, sets of paths which are restricted on a finite number of steps. The probability measure P_α relies on the initial distribution, the one-step probabilities and the distribution of sojourn times.

Theorem 1. Let $P_\alpha : \sigma(\mathcal{C}) \to [0,1]$ be the unique outer measure generated by p_α, restricted to the measurable sets in $\sigma(\mathcal{C})$. Then $(\Omega, \sigma(\mathcal{C}), P_\alpha)$ is a probability space.

Proof. $(\Omega, \sigma(\mathcal{C}), P_\alpha)$ is a measure space by Caratheodory's Theorem [13, 38]. Since $p_\alpha(\Omega) = P_\alpha(\Omega) = 1$, it is also a probability space. □

If $\alpha = \alpha^{(s)}$ is an initial distribution where the complete probability mass is concentrated on state s, we simply write P_s instead of $P_{\alpha^{(s)}}$.

2.6 State labels

CTMCs as introduced in Section 2.1 simply consist of states, transitions, rates and initial probabilities. In order to be useful as a model for some kind of real world system, it has to reflect characteristic properties of the system. The states of a CTMC should mirror the (abstracted) states of a real world system. A state of a real system can be characterised by its most basic properties. In the following, the set of such basic properties is called the set of *atomic propositions*, denoted by AP.

A state *labelling* $L : S \to 2^{AP}$ attaches sets of atomic propositions to states. We say that a state $s \in S$ is labelled with an atomic proposition ap if the labelling of s contains this atomic proposition, that is, $ap \in L(s)$.

Example 10. Think of a simple processing device which can be

- `idle`,
- `active`,
- `sleeping` or
- `broken`.

These four adjectives describe the four possible states of the processing device. Its state may have other properties, for example, `intact` (meaning the opposite of `broken`) or `empty` (meaning the opposite of `active`). Hence, let the set of atomic propositions be

$$AP = \{\texttt{idle}, \texttt{active}, \texttt{sleeping}, \texttt{broken}, \texttt{intact}, \texttt{empty}\}.$$

Now take our running example CTMC, depicted in Example 1. Let state 0 be the `idle` state. In this state, the system is also `intact` and `empty`. Thus, the labelling for state 0 is

$$L(0) = \{\texttt{idle}, \texttt{intact}, \texttt{empty}\}.$$

State 1 is the `sleeping` state. Its labelling is given by

$$L(1) = \{\texttt{sleeping}, \texttt{intact}, \texttt{empty}\}.$$

The `active` state is represented by CTMC state 2 with labelling

$$L(2) = \{\texttt{active}, \texttt{intact}\}.$$

Finally, state 3 stands for the `broken` state and has labelling

$$L(3) = \{\texttt{broken}\}.$$

With this labelling, the CTMC has become a model for our simple processing device. From the `idle` state the system can either switch into `active` or `sleeping` state, or it might break. From the `active` state it has to return to the `idle` state, as is the case from the `sleeping` state. There is no outgoing transition from the broken state. Note, that the model contains the (unrealistic) assumption that the device only might break down in `idle` state. ♢

A *labelled CTMC* is a tuple $(AP, S, \mathbf{R}, L)$ consisting of a set of atomic propositions AP, a CTMC $(S, \mathbf{R})$ and a labelling L.

We have not yet introduced labels for transitions. See Chapter 7 for extending the idea of labelling in this sense.

2.7 Rewards

With *state rewards* we add an extra dimension to a CTMC. While proceeding in time, the CTMC accumulates a yield. The increase per unit of time depends on the state the CTMC resides in. Formally, the reward per unit of time is defined as the reward rate $\rho_s \in \mathbb{R}_{\geq 0}$ for every state $s \in S$. Of course, the accumulation of reward can also be interpreted as the accumulation of cost.

The reward rates define a stochastic process $(Y_t, t \in \mathbb{R}_{\geq 0})$, which is the reward accumulated from time 0 to t by the reward rate given by the CTMC X_t:

$$Y_t = \int_0^t \rho_{X_t} dx.$$

Note that Y_t is *not* Markovian, its future outcome depends on the complete past (the integral starts at 0). But Y_t is defined on the same probability space as X_t, since it is a function of X_t. Thus, for an infinite path $\omega = ((s_0, t_0)(s_1, t_1)\ldots)$, the accumulated reward at time t is given by

$$Y_t = \sum_{i=0}^{n-1} t_i \cdot \rho_{s_i} + (t - \sum_{i=0}^{n-1} t_i) \cdot \rho_{s_n} \text{ iff } \sum_{i=0}^{n-1} t_i \leq t \text{ and } \sum_{i=0}^{n} t_i > t.$$

If at time t the CTMC has reached the nth state of the path, the accumulated reward at t is the sum of the rewards accumulated in the preceeding states. For each state s_i, this is the product of reward rate ρ_{s_i} and sojourn time t_i. For state s_n, only the residual time until t is considered.

Example 11. Rewards can model costs. Not only monetary costs are possible: we can, e.g., equip the processing device of Example 10 with rewards reflecting *energy consumption.*

In the idle state 0, the reward rate is $\rho_0 = 50$. The sleeping state 1 has smaller reward rate, $\rho_1 = 20$. When being active (state 2), the energy consumption is $\rho_2 = 100$. When broken (state 3), the device has stopped consuming any energy: $\rho_3 = 5$.

Recall the path

$$\omega_1 = (0, 0.25), (1, 0.25), (0, 0.25), (1, 0.25), (0, 0.25), (1, 0.25), \ldots$$

defined in Example 5. The accumulated reward for this path at time 1.1 is

$$\begin{aligned} Y_{1.1}(\omega_1) &= \rho_0 \cdot 0.25 + \rho_1 \cdot 0.25 + \rho_0 \cdot 0.25 + \rho_1 \cdot 0.25 + \rho_0 \cdot 0.1 \\ &= 50 \cdot 0.6 + 20 \cdot 0.5 = 40. \end{aligned}$$

If we set the basic time unit to 1 minute and the basic reward unit to 1 mA, the accumulated reward for ω_1 at time 1.1 is $1.\overline{3}$mAh. ◊

A CTMC $(S, \mathbf{R})$ paired with a reward rate function $\rho : S \rightarrow \mathbb{R}_{\geq 0}$ is a *Markov reward model* (MRM). If its states are also labelled, we call the tuple $(AP, S, \mathbf{R}, L, \rho)$ a labelled MRM. It is also possible to assign *impulse rewards* to transitions. For further reference, see [28]. Throughout this thesis we deal only with state rewards as defined here.

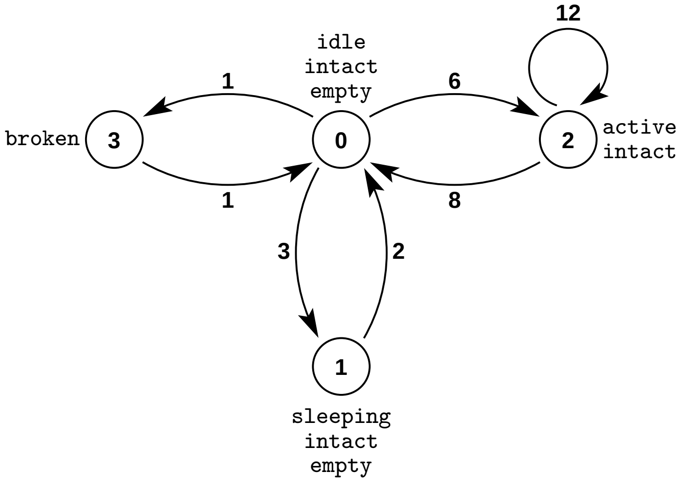
2.8 High level description of MRMs

CTMCs are usually not specified manually enumerating all states and transitions but derived automatically from a high-level formalism such as queueing networks [29], stochastic process algebra [54, 58], stochastic activity networks [75] or generalised stochastic Petri nets (GSPNs) [1]. For most of these formalisms there exist extensions that also allow for the high-level specification of rewards [10, 11, 15, 19, 20, 87]. For an overview and supporting tools see also [53, 52]. We use GSPNs enhanced with a simple way to specify state labels and reward rates to specify MRMs. These GSPNs are described in an extension of the GSPN description language CSPL [16]. Atomic proposition are defined for the states of the MRM (= tangible markings of the GSPN) by making statements about the number of tokens in the places [30]. Reward rates are defined as expressions over the number of tokens in the places. For the specification of both atomic propositions and reward rates, the full power of the programming language C is available.

The state space generation code of [48] has been adapted to derive the labelled MRM automatically from the GSPN specification.

2.9 Running example

In the following chapters we often refer to the MRM defined and used in the Examples of this chapter. In this section we give all ingredients at a glance for easy reference.



The labelled MRM $\mathcal{M} = (AP, S, \mathbf{R}, L, \rho)$ consists of

- the set of atomic propositions $AP = \{\texttt{idle}, \texttt{active}, \texttt{sleeping}, \texttt{broken}, \texttt{intact}, \texttt{empty}\}$,
- the state space $S = \{0, 1, 2, 3\}$,
- the rate matrix
$$\mathbf{R} = \begin{pmatrix} 0 & 3 & 6 & 1 \\ 2 & 0 & 0 & 0 \\ 8 & 0 & 12 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix},$$
- the labelling function L with
$L(0) = \{\texttt{idle}, \texttt{intact}, \texttt{empty}\}$,
$L(1) = \{\texttt{sleeping}, \texttt{intact}, \texttt{empty}\}$,
$L(2) = \{\texttt{active}, \texttt{intact}\}$,
$L(3) = \{\texttt{broken}\}$, and
- the reward vector $\rho = (50, 20, 100, 5)$.

2.10 Summary

In this section we introduced Markov reward models (MRMs), the modelling formalism used in the remainder of this thesis. The base of MRMs are continuous-time Markov chains. A CTMC is a stochastic process with discrete state space where the transitions are governed by negative exponentially distributed random variables. A probability space can be constructed over all possible realisations (paths) of the stochastic process. We showed how to enhance CTMCs with state labels and reward rates in order to obtain labelled MRMs.

The next chapter introduces some basic numerical measures for MRMs. They refer to the distributions of the state and/or the accumulated reward at a given point in time.

Chapter 3

Basic measures for Markov reward models

Markov reward models have a long tradition in the evaluation of performance properties like availability or reliability. Most of these properties are based on a few basic measures. One is the *transient* distribution of the MRM, that is, the probabilities of residing in each of the states at a given point in time. In the long run, an MRM might reach an equilibrium which is characterised by the *steady-state* distribution. With respect to the accumulated reward, its distribution (the *performability*) at some point in time is a basic property of an MRM.

The organisation of this chapter is as follows. We first show how to derive the generator matrix of an MRM which is needed for the computation of all the measures (Section 3.1). In Sections 3.2 and 3.3, respectively, we explain the computation of the transient and steady-state distribution of an MRM. Finally, in Section 3.4 the accumulated reward comes into play and we derive the solution of a set of PDEs characterising the joint distribution of state and reward. The actual computation of this joint distribution is postponed till Chapter 5.

3.1 The generator matrix Q

In Section 2.1 we introduced CTMCs by specifying their transition rate matrix $\mathbf{R}$. In this matrix self-loops from a state s back into itself are allowed, that is, a diagonal entry $\mathbf{R}_{ss}$ might be positive. Traditionally, CTMCs have no self-loops and are defined by their *generator* matrix $\mathbf{Q}$. The matrix $\mathbf{Q}$ has non-negative entries at all off-diagonal positions, indicating the rates of transitions between different states. Each diagonal entry is defined to be the negative row sum of the off-diagonal entries, i.e.,

$$Q_{ss} = -\sum_{s' \neq s} Q_{ss'}.$$

The structure of the generator $\mathbf{Q}$ is informally explained as follows. An entry $\mathbf{Q}_{ss'}$ can be viewed as the *derivative* of the probability of leaving state s towards state s' [100]. The probability of actually leaving state s increases with time (or is 0 if there is no transition). The probability of staying in state s decreases with time (or is 0, if state s is absorbing).

If we have a rate matrix $\mathbf{R}$ that has positive entries on the diagonal, we can still derive a proper generator without altering the stochastic behaviour of the CTMC X_t. To do so, define the generator $\mathbf{Q}$ for a given rate matrix R as

$$Q_{ss'} = \begin{cases} R_{ss'}, & s \neq s', \\ -\sum_{z \neq s} R_{sz}, & s = s'. \end{cases}$$

Using this generator, the residence time in a state s is exponentially distributed with rate $-Q_{ss}$. Recall that $E_s = \sum_{s' \in S} \mathbf{R}_{ss'}$ is the sum of rates of transitions starting in state s and that $-Q_{ss} = E_s - R_{ss}$. The probability of moving from state s to state $s' \neq s$ is then given by

$$\frac{Q_{ss'}}{-Q_{ss}} = \frac{R_{ss'}}{E_s - R_{ss}}.$$

When considering the rate matrix, the *effective* residence time (as opposed to the time before the next transition, cf. Section 2.2) of state s is composed of several intervals because a transition might lead back into s without changing state. Each transition actually leaves s with probability $1 - P_{ss}$, where $\mathbf{P}$ is the matrix of one-step probabilities (cf. Section 2.3).

However, the residence time in state s is exponentially distributed [18, p. 259] with rate

$$E_s \cdot (1 - P_{ss}) = E_s \cdot (1 - \frac{R_{ss}}{E_s}) = E_s - R_{ss} = -Q_{ss}.$$

If the CTMC actually leaves a state s towards a state s', it does so with probability $P_{ss'}$ renormalised to the probability of *not* returning to s:

$$\frac{P_{ss'}}{\sum_{z \neq s} P_{sz}} = \frac{\frac{R_{ss'}}{E_s}}{\sum_{z \neq s} \frac{R_{sz}}{E_s}} = \frac{Q_{ss'}}{-Q_{ss}}.$$

Hence, regardless of which matrix we consider, the generator $\mathbf{Q}$ or the rate matrix $\mathbf{R}$, we obtain the same effective state residence times and transition probabilities. We can therefore safely use the generator matrix $\mathbf{Q}$ as defined above to compute measures for a CTMC specified by a rate matrix $\mathbf{R}$.

Example 12. For the running example of Section 2.9, the rate matrix equals

$$\mathbf{R} = \begin{pmatrix} 0 & 3 & 6 & 1 \\ 1 & 0 & 0 & 0 \\ 8 & 0 & 12 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

The corresponding generator equals

$$\mathbf{Q} = \begin{pmatrix} -10 & 3 & 6 & 1 \\ 1 & -1 & 0 & 0 \\ 8 & 0 & -8 & 0 \\ 1 & 0 & 0 & -1 \end{pmatrix}.$$

◊

3.2 The transient distribution

The stochastic process X_t describes the behaviour of the MRM $\mathcal{M}$ over time. Each random variable X_t has a discrete distribution over all states in S. We denote the distribution of X_t given a single starting state s as

$$\Pi_{ss'}(t) = \Pr{}^{\mathcal{M}}\{X_t = s' \mid X_0 = s\},$$

which can also be written as a matrix $\mathbf{\Pi}(t) = \left(\Pi_{ss'}(t)\right)_{s,s' \in S}$.

In order to compute the transient probability $\Pi_{ss'}(t + \Delta t)$ at time $t + \Delta t$, we consider what could have happened in the short interval $(t, t + \Delta t)$ [100]:

i) the CTMC has been in state s' at time t and no transition has occurred with probability
$$\Pi_{ss'}(t) \cdot (1 - \sum_{z \neq s'} Q_{s'z} \Delta t) = \Pi_{ss'}(t) \cdot (1 + Q_{s's'} \Delta t);$$

ii) the CTMC has been in a state $z \neq s'$ at time t and exactly one transition has occurred with probability
$$\Pi_{sz}(t) Q_{zs'} \Delta t;$$

iii) more than one transition has occurred with negligible probability $o(\Delta t)$, where
$$\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} = 0.$$

Hence,

$$\Pi_{ss'}(t + \Delta t) \quad = \quad \Pi_{ss'}(t) \; + \; \Pi_{ss'}(t) Q_{s's'} \Delta t \quad + \quad \sum_{z \neq s'} \Pi_{sz}(t) Q_{zs'} \Delta t \quad + \quad o(\Delta t)$$

We move the first summand $\Pi_{ss'}(t)$ from the right-hand to the left-hand side and combine the second and the third summand:

$$\Pi_{ss'}(t + \Delta t) \quad - \quad \Pi_{ss'}(t) = \sum_{z \in S} \Pi_{sz}(t) Q_{zs'} \Delta t \quad + \quad o(\Delta t).$$

Dividing by Δt and taking the limit for $\Delta t \to 0$, the left hand side turns into the definition of the derivative of $\Pi_{ss'}(t)$ with respect to t and we obtain a differential equation for $\Pi_{ss'}(t)$:

$$\begin{aligned} \lim_{\Delta t \to 0} \frac{\Pi_{ss'}(t + \Delta t) - \Pi_{ss'}(t)}{\Delta t} &= \lim_{\Delta t \to 0} \frac{\sum_{z \in S} \Pi_{sz}(t) Q_{zs'} \cancel{\Delta t}}{\cancel{\Delta t}} + \underbrace{\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t}}_{=0} \\ \Rightarrow \quad \frac{d\Pi_{ss'}(t)}{dt} &= \sum_{z \in S} \Pi_{sz}(t) Q_{zs'}. \end{aligned} \tag{3.1}$$

For all state pairs $s, s' \in S$, the equations (3.1) form a coupled system of ordinary differential equations (ODEs) which, in matrix notation, equals

$$\frac{d\,\mathbf{\Pi}(t)}{dt} \;=\; \mathbf{\Pi}(t)\mathbf{Q}. \tag{3.2}$$

The initial value is $\mathbf{\Pi}(0) = \mathbf{I}$ since the CTMC cannot have performed any transition by time 0 and so each state is just reachable from itself (with probability 1).

We could employ generic or tailored numerical ODE solvers for the solution of (3.2). For an overview refer to [100]. However, the method of choice is based on the analytical solution of (3.2), which is given by the matrix exponential

$$\mathbf{\Pi}(t) \;=\; \mathbf{\Pi}(0) \cdot e^{\mathbf{Q}t} = \sum_{n=0}^{\infty} \frac{(\mathbf{Q}t)^n}{n!}. \tag{3.3}$$

A computation of this matrix exponential is not feasible [77, 78] for general $\mathbf{Q}$. Because one can exploit special properties of the generator $\mathbf{Q}$, the method of choice is *uniformisation*, also called Jensen's method [44] or randomisation [45].

Let $\lambda \geq \max_s\{-q_{ss} \mid s \in S\}$ be a rate that is not smaller than the biggest absolute value of the diagonal entries of $\mathbf{Q}$. We define the stochastic matrix

$$\mathbf{U} = \mathbf{I} + \frac{1}{\lambda}\mathbf{Q}.$$

Replacing $\mathbf{Q}$ by $\lambda(\mathbf{U} - \mathbf{I})$ in (3.3), we obtain

$$\begin{aligned}\mathbf{\Pi}(t) &= e^{\lambda(\mathbf{U}-\mathbf{I})t} \\ &= e^{-\lambda t} \cdot e^{\lambda t \mathbf{U}} \\ &= \sum_{n=0}^{\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \cdot \mathbf{U}^n. \end{aligned} \tag{3.4}$$

The stochastic matrix $\mathbf{U}$ defines a *discrete-time* Markov chain $(Z_n, n \in \mathbb{N})$. In contrast to the continuous-time setting, the time between two transitions is not further specified, the discrete parameter $n \in \mathbb{N}$ counts only epochs. An entry $U_{ss'}$ in the matrix $\mathbf{U}$ indicates the probability of moving from state s to state s'. Note that self-loops are allowed. The stochastic process Z_n is called the *uniformised* Markov chain. Its distribution after n steps is simply given by the nth power of $\mathbf{U}$, hence,

$$\Pr\{Z_n = s' \mid Z_0 = s\} = \mathbf{U}^n_{ss'}.$$

Since $\mathbf{U}$ is a stochastic matrix, we can compute its subsequent powers in a stable recursive fashion:

$$\mathbf{U}^0 = \mathbf{I}, \quad \mathbf{U}^n = \mathbf{U}^{n-1} \cdot \mathbf{U}, \text{ for } n > 0.$$

The factor $e^{-\lambda t}\frac{(\lambda t)^n}{n!}$ occurring in (3.4) before $\mathbf{U}^n$ is the probability that exactly n events have occurred in the interval [0,t) in the Poisson process $(N_t, t \in \mathbb{R}_{\geq 0})$ with rate λ:

$$\Pr\{N_t = n\} = P\!\!P(\lambda t, n) = e^{-\lambda t}\frac{(\lambda t)^n}{n!}.$$

Consider now the continuous-time stochastic process Z_{N_t}, where the transitions between states are ruled by the stochastic matrix $\mathbf{U}$ and the time of transition occurrence is determined by the Poisson process N_t. It has been shown [90] that the CTMC X_t and Z_{N_t} are stochastically equivalent, that is, for a start state s

$$\Pr\{X_t = s' \mid Z_0 = s\} = \Pr\{Z_{N_t} = s' \mid Z_0 = s\}.$$

Using the law of total probability, this can be rewritten as the weighted sum over the distribution after n steps.

$$\begin{aligned}\Pr\{X_t = s' \mid Z_0 = s\} &= \sum_{n=0}^{\infty} \Pr\{N_t = n\} \Pr\{Z_n = s' \mid Z_0 = s\} \\ &= \sum_{n=0}^{\infty} \mathit{PP}(\lambda t, n) \cdot \mathbf{U}^n_{ss'},\end{aligned}$$

which, in matrix notation, equals (3.4).

Yet another possibility to derive (3.4) using the uniformised Markov chain is to condition $\Pi_{ss'}(t)$ on the possible paths of the discrete-time Markov chain described by $\mathbf{U}$. A *uniformised path* is a finite sequence of states $\sigma = (s_0, \cdots, s_n) \in S^{n+1}$. The set of all uniformised paths is denoted $\mathit{uPath} = \bigcup_{n=0}^{\infty} S^{n+1}$. The *length* $|\sigma|$ of a uniformised path σ is given by the number of transitions, that is $|(s_0, \cdots, s_n)| = n$. With $\mathit{first}(\sigma)$ we denote the first state of the sequence ($\mathit{first}(s_0, \cdots, s_n) = s_0$) and with $\mathit{last}(\sigma)$ we denote the last state of the sequence ($\mathit{last}(s_0, \cdots, s_n) = s_n$). The probability that the DTMC follows the path $\sigma = (s_0, \cdots, s_n)$ is given by $P(\sigma) = U_{s_0 s_1} \cdot U_{s_1 s_2} \cdot \cdots U_{s_{n-1} s_n}$. A uniformised path $\sigma = s$ of length 0 has probability $P(s) = 1$. The probability that a path of length n has been taken by time t is $\mathit{PP}(\lambda t, n)$. We can now write $\Pi_{ss'}(t)$ as

$$\begin{aligned}\Pi_{ss'}(t) &= \sum_{n=0}^{\infty} \mathit{PP}(\lambda t, n) \sum_{\substack{\sigma \in \mathit{uPath} \\ |\sigma| = n}} \Pr\{Z_n = s' \mid Z_0 = s, \sigma\} \cdot P(\sigma \mid Z_0 = s) \\ &= \sum_{n=0}^{\infty} \mathit{PP}(\lambda t, n) \sum_{\substack{\sigma \in \mathit{uPath} \\ |\sigma| = n \\ \mathit{first}(\sigma) = s \\ \mathit{last}(\sigma) = s'}} P(\sigma), \qquad (3.5)\end{aligned}$$

because $P(\sigma \mid Z_0 = s) = 0$ if $\mathit{first}(\sigma) \neq s$ and

$$\Pr\{Z_n = s' \mid Z_0 = s, \sigma\} = \begin{cases} 1, & \mathit{last}(\sigma) = s', \\ 0, & \text{otherwise.} \end{cases}$$

For $n = 0$, we have

$$\sum_{\substack{\sigma \in uPath \\ |\sigma| = 0 \\ \mathit{first}(\sigma) = s \\ \mathit{last}(\sigma) = s'}} P(\sigma) = \begin{cases} 1, & s = s' \\ 0, & \text{otherwise}, \end{cases} = \mathbf{I}_{ss'} = \mathbf{U}^0_{ss'}.$$

Uniformised paths of length n can be constructed by appending one state to paths of length $n-1$. By induction we obtain the following for $n > 0$:

$$\sum_{\substack{\sigma \in uPath \\ |\sigma| = n \\ \mathit{first}(\sigma) = s \\ \mathit{last}(\sigma) = s'}} P(\sigma) = \sum_{z \in S} \sum_{\substack{\sigma' \in uPath \\ |\sigma| = n-1 \\ \mathit{first}(\sigma') = s \\ \mathit{last}(\sigma') = z}} P(\sigma')U_{zs'} = \sum_{z \in S} \mathbf{U}^{n-1}_{sz} U_{zs'} = \mathbf{U}^n_{ss'}.$$

Using this result in the matrix version of (3.5) we obtain again (3.4). Hence, the analytical solution of the ODE, the stochastic argumentation and the conditioning on paths in the uniformised Markov chain lead to the same expression for $\mathbf{\Pi}(t)$.

Error bound. Both $PP(\lambda t, n)$ and $\mathbf{U}^n$ can be computed efficiently, however, it is not possible to evaluate the infinite summation. We have to truncate at some $N \in \mathbb{N}$ and approximate $\mathbf{\Pi}(t)$ by

$$\mathbf{\Pi}(t) \approx \sum_{n=0}^{N} PP(\lambda t, n)\mathbf{U}^n.$$

The error introduced by this approximation for a single element $\Pi_{ss'}(t)$ of $\mathbf{\Pi}(t)$ is

$$\sum_{n=N+1}^{\infty} PP(\lambda t, n) \underbrace{\mathbf{U}^n_{ss'}}_{\in[0,1]},$$

which is bounded by

$$\varepsilon = \sum_{n=N+1}^{\infty} PP(\lambda t, n) \cdot 1 = 1 - \sum_{n=0}^{N} PP(\lambda t, n)$$

because the sum of all Poisson probabilities is 1. For a predefined maximum error ε the number N_ε of transitions that has to be taken into account can be determined a priori.

Initial distribution. Often we are not interested in the transient probability of moving from one state to another but just want to know the transient distribution $\pi(t)$ at time

t given an initial distribution (cf. Section 2.4) α. The row vector $\pi(t)$ containing the transient distribution can easily be obtained by multiplying α with the matrix $\mathbf{\Pi}(t)$:

$$\pi(t) = \alpha \cdot \mathbf{\Pi}(t).$$

Multiplying both sides of (3.2) with α, we obtain the ODE for $\pi(t)$:

$$\frac{d\pi(t)}{dt} = \pi(t)\mathbf{Q},$$

with initial value $\pi(0) = \alpha\mathbf{\Pi}(0) = \alpha\mathbf{I} = \alpha$. It can be solved with the same methods as (3.2). It is not necessary to compute the complete matrix $\mathbf{\Pi}(t)$.

Final states. In a model checking context, one is interested in reaching any of a set of final or goal states $\mathcal{G} \subseteq S$. Let $\gamma(t) \in [0,1]^{|S|}$ be the column vector containing the probabilities

$$\gamma_s(t) = \Pr\{X_t \in \mathcal{G} \mid X_0 = s\}$$

for each $s \in S$. Note that $\gamma(t)$ does *not* represent a probability distribution. We have

$$\gamma_s(0) = \begin{cases} 1, & s \in \mathcal{G}, \\ 0, & \text{otherwise.} \end{cases}$$

For any time t, the vector $\gamma(t)$ is then defined as

$$\gamma(t) = \mathbf{\Pi}(t)\gamma(0).$$

We can also compute $\gamma(t)$ directly using uniformisation to solve

$$\frac{d\gamma(t)}{dt} = \mathbf{Q}\gamma(t)$$

with initial value $\gamma(0)$.

Example 13. Recall the generator matrix $\mathbf{Q}$ of the illustrating example:

$$\mathbf{Q} = \begin{pmatrix} -10 & 3 & 6 & 1 \\ 1 & -1 & 0 & 0 \\ 8 & 0 & -8 & 0 \\ 1 & 0 & 0 & -1 \end{pmatrix}.$$

The absolute values of the diagonal elements are all ≤ 10, so we choose the uniformisation constant $\lambda = 10$. The probability matrix of the uniformised Markov chain is then

$$\mathbf{U} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} + \frac{1}{10}\begin{pmatrix} -10 & 3 & 6 & 1 \\ 1 & -1 & 0 & 0 \\ 8 & 0 & -8 & 0 \\ 1 & 0 & 0 & -1 \end{pmatrix} = \frac{1}{10}\begin{pmatrix} 0 & 3 & 6 & 1 \\ 1 & 9 & 0 & 0 \\ 8 & 0 & 2 & 0 \\ 1 & 0 & 0 & 9 \end{pmatrix}$$

We compute $\mathbf{\Pi}(1.0)$ with an accuracy of $\varepsilon = 10^{-5}$. The number of steps that has to be considered is $N_\varepsilon = 26$.

Computing the transient probabilities using uniformisation yields

$$\mathbf{\Pi}(1.0) = \begin{pmatrix} 0.1885 & 0.4981 & 0.1474 & 0.1660 \\ 0.1660 & 0.6265 & 0.1213 & 0.0862 \\ 0.1965 & 0.1568 & 0.1568 & 0.1617 \\ 0.1660 & 0.1213 & 0.1213 & 0.4541 \end{pmatrix}.$$

Figure 3.1 (a)–(d) shows the evolution of the transient probabilities over time. From there on they change until they settle to an equilibrium distribution which is independent from the starting state (see also Section 3.3).

For the initial distribution $\boldsymbol{\alpha} = (\frac{1}{2}, \frac{1}{2}, 0, 0)$ as defined in Example 4 we obtain the transient distribution

$$\pi^\alpha(1.0) = (0.1773, 0.5623, 0.1343, 0.1261).$$

These probabilities are the sum of the first two elements of each column of $\mathbf{\Pi}(1.0)$ weighted by $\frac{1}{2}$. Figure 3.1 (e) depicts the transient probabilities for the initial distribution α over time. They reach the same equilibrium distribution as in (a)–(d).

For $\mathcal{G} = \{2, 3\}$ a set of goal states ($\boldsymbol{\gamma}(0) = (0, 0, 1, 1)^T$), the probabilities of reaching one of the goal states are

$$\gamma(1.0) = (0.3134, 0.2075, 0.3185, 0.5754).$$

Each entry in $\gamma(1.0)$ is the sum of the last two entries of each row of $\mathbf{\Pi}(1.0)$. Remember that $\gamma(t)$ is not a probability distribution. Independently from the starting state the probabilities converge to the same value: on the long run it does not matter where we have started (Figure 3.1 (f)). $\diamondsuit$

Poisson probabilities

The Poisson probabilities $PP(\lambda t, n)$ can efficiently be precomputed. Starting from the *mode* $m = \lfloor \lambda t \rfloor$ (here the Poisson probability is highest) we recursively compute *weights*

$$w[j-1] = \frac{j}{\lambda t} w[j], \text{ for } j = m, \cdots, L+1,$$

and

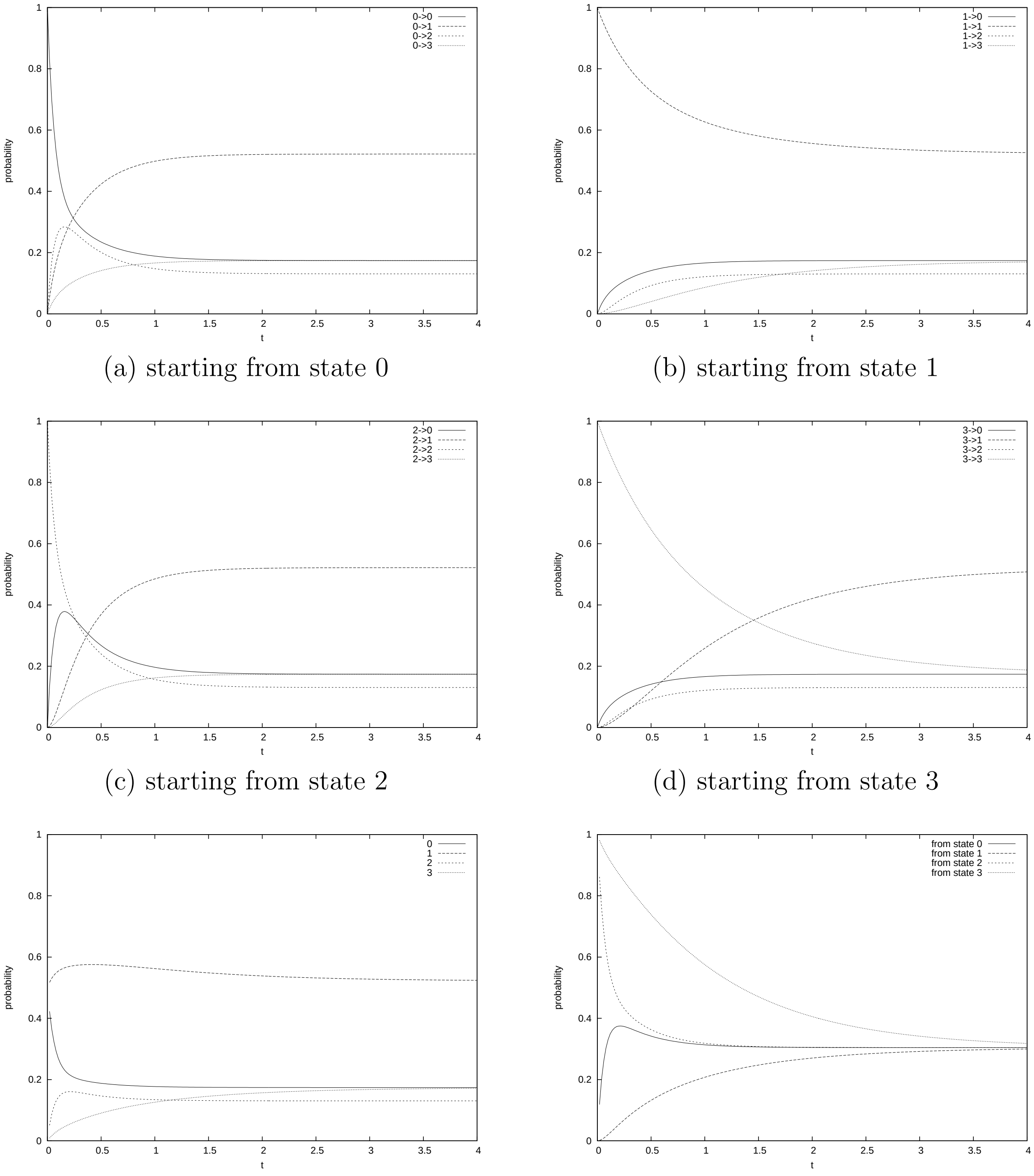
$$w[j+1] = \frac{\lambda t}{j+1} w[j], \text{ for } j = m, \cdots, R-1.$$

The actual Poisson probabilities are then given by

$$PP(\lambda t, n) = W \cdot w[n],$$

where $W = \sum_{j=L}^{N} w[j]$.

For a given accuracy ε, Fox and Glynn [40] derive the left and right truncation points L and R in such a way that at most ε probability mass is lost and underflows are avoided.



(a) starting from state 0

(b) starting from state 1

(c) starting from state 2

(d) starting from state 3

(e) initial distribution $\alpha = (\frac{1}{2}, \frac{1}{2}, 0, 0)$

(f) final states 2 and 3

Figure 3.1: Transient probabilities for the running example (Example 13)

Complexity

In order to carry out uniformisation we need space for the matrix $\mathbf{\Pi}$ and for the subsequent powers of $\mathbf{U}$. For the latter we need two copies for a proper calculation. Each of these matrices needs $\mathcal{O}(|S|^2)$ storage.

The matrix-matrix multiplication in each step requires $|S|^3$ multiplications. The number of steps needed for a given accuracy is in $\mathcal{O}(\lambda t)$ and so the overall time complexity is $\mathcal{O}(\lambda t \cdot |S|^3)$. If $\mathbf{U}$ is stored in an appropriate sparse format [14] we can replace $|S|^3$ by $\mathsf{nz}(\mathbf{U}) \cdot |S|$ where $\mathsf{nz}(\mathbf{U})$ is the number of non-zero entries of $\mathbf{U}$, which is in general much smaller than $|S|^2$.

3.3 The steady-state distribution

In many cases the transient probabilities $\mathbf{\Pi}(t)$ reach an equilibrium on the long run. For CTMCs with finite state space these *steady-state* probabilities $\mathbf{\Pi}$ always exist. They are defined to be the limit of the transient probabilities for $t \to \infty$. The probability of residing in state s' on the long run, having started in state s is then

$$\Pi_{ss'} = \lim_{t\to\infty} \Pi_{ss'}(t).$$

For a finite MRM where any state is reachable from every other state, that is, whose graph consists of a single strongly connected component, the steady state distribution is independent of the starting state and can be written as $\Pi_{ss'} = \pi_{s'}$. The steady-state probabilities are characterised by the fact that progress in time does not change the probability distribution. The derivatives are zero and the system of differential equations for the transient probabilities collapses into a system of linear equations:

$$\pi\mathbf{Q} = \mathbf{0}. \tag{3.6}$$

The vector π represents a discrete distribution and so we pick the right solution by applying the normalisation equation

$$\sum_{s\in S} \pi_s = 1. \tag{3.7}$$

Example 14. The running example MRM consists of exactly one strongly connected component. Solving the system of linear equations (3.6) results in

$$\pi = (0.1739, 0.1304, 0.5218, 0.1739)$$

On the long run, the MRM spends 17.39% of the time in state 0, 13.04% in state 1, 52.18% in state 2 and 17.39% in state 3. This is exactly the equilibrium distribution the transient probabilities reach for bigger t (cf. Figure 3.1). ◊

If the (graph of the) MRM is not strongly connected, things are more involved [7]. A *bottom strongly connected component* (BSCC) is a maximal strongly connected component of the MRM such that there are no outgoing transitions to states not contained in this component (there might be incoming transitions). Note that every absorbing state forms a BSCC by itself. States not contained in any of the BSCCs are *transient* states.

Example 15. Figure 3.2 shows the graph of an MRM with four states. State 0 is transient, absorbing state 1 forms the BSCC B_1 and states 2 and 3 constitute the BSCC B_2. ◊

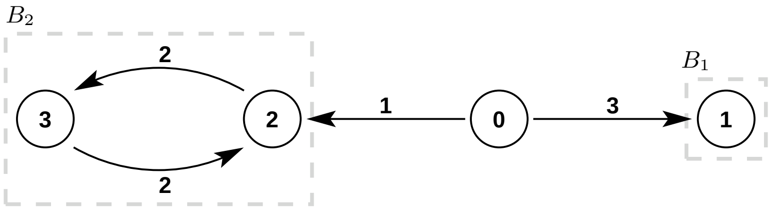


Figure 3.2: Graph of an MRM with two BSCCs

Restricting the generator to the states of one of these BSCCa $B \subseteq S$, the steady-state distribution for B can be determined using (3.6) and (3.7). The steady-state probability of a state s' in a BSCC B, having started in state s, is then

$$\Pi_{ss'} = \Pr\left\{s \to^* B\right\} \cdot \pi^B_{s'},$$

where $\Pr\left\{s \to^* B\right\}$ is the probability of eventually reaching one of the states in B, having started in state s. This probability is the solution in $[0, 1]$ of the following system of linear equations:

$$\Pr\left\{s \to^* B\right\} = \begin{cases} 1, & s \in B, \\ \sum_{s' \in S} P_{ss'} \cdot \Pr\left\{s' \to^* B\right\}, & \text{otherwise.} \end{cases}$$

Example 16. Consider the MRM of Example 15. The steady-state distribution of the BSCC B_1 is $\pi_1^{B_1} = 1$, for B_2 we have $\pi_2^{B_2} = \frac{1}{2}$ and $\pi_3^{B_2} = \frac{1}{2}$. The probabilities of reaching B_1 are

$$\Pr\left\{0 \to^* B_1\right\} = \frac{3}{4},\ \Pr\left\{1 \to^* B_1\right\} = 1,\ \Pr\left\{2 \to^* B_1\right\} = 0,\ \Pr\left\{3 \to^* B_1\right\} = 0,$$

and the probabilities of reaching B_2 are

$$\Pr\left\{0 \to^* B_2\right\} = \frac{1}{4},\ \Pr\left\{1 \to^* B_2\right\} = 0,\ \Pr\left\{2 \to^* B_2\right\} = 1,\ \Pr\left\{3 \to^* B_2\right\} = 1.$$

Note that the BSCCs are not reachable from each other (probability is zero). The complete matrix of steady-state probabilities is then given by

$$\mathbf{\Pi} = \begin{pmatrix} 0 & \frac{3}{4} & \frac{1}{8} & \frac{1}{8} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Each row of the matrix $\mathbf{\Pi}$ is the steady-state distribution for a single starting state. ◊

A detailed description of the numerous solution algorithms for systems of linear equations in general and the steady-state distribution of Markov chains in particular can be found in [100].

3.4 The joint distribution of X_t and Y_t

In Section 2.7 we introduced Y_t, the accumulated reward up to time t. The stochastic process Y_t is fully determined by the CTMC X_t and the reward structure ρ, because

$$Y_t = \int_0^t \rho(X_t)\, dx.$$

In contrast to the CTMC X_t, the stochastic process Y_t is non-Markovian and has state space $\mathbb{R}_{\geq 0}$. Whereas the distribution of X_t is discrete, the distribution of Y_t is continuous. Unfortunately, the distribution of Y_t depends on all X_τ for $\tau \in [0,t]$ and is therefore not easy to compute. For CSRL model checking (cf. Chapter 4) we are concerned with the *joint distribution* of the state process X_t and the accumulated reward Y_t. Define $\boldsymbol{\Upsilon}(t,y)$ to be the matrix of the joint distribution of state and reward with entries

$$\Upsilon_{ss'}(t,y) = \Pr\left\{X_t = s', Y_t \leq y \mid X_0 = s\right\}.$$

Similarly to the derivation of the ODE for the transient distribution of X_t, we consider $\Upsilon_{ss'}(\Delta t + t, y)$ by observing what can happen in the small time interval $[0,\Delta t)$, that is, we consider the beginning of the interval $[0, \Delta t + t)$ as shown in Figure 3.3. We always assume that the MRM is in state s at time 0, which leads to a backward equation.

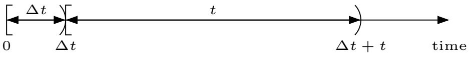


Figure 3.3: Time interval considered to derive the backward equation for $\Upsilon_{ss'}(t,y)$

i) The MRM is in state s at time Δt and no transition has occurred before. This implies, that the accumulated reward between Δt and $\Delta t + t$ is at most $y - \rho_s \Delta t$. The probability for this scenario is

$$(1 - \sum_{z \neq s} Q_{sz}\Delta t) \cdot \Upsilon_{ss'}(t, y - \rho_s\Delta t) = (1 + Q_{ss}\Delta t) \cdot \Upsilon_{ss'}(t, y - \rho_s \Delta t).$$

ii) The MRM is in a state $z \neq s$ at time Δt and exactly one transition has occurred before. We denote the probability for this case with $g_z(\Delta t)$. We do not know the exact amount of reward accumulated in the interval $[0, \Delta t)$, since the time instant of the transition from s to z is not known. However, we can bound $g_z(\Delta t)$ from below and above by observing, that the reward accumulated in the interval of length Δt is at most $\rho_{\max}\Delta t$ and at least $\rho_{\min}\Delta t$ (where $\rho_{\max} = \max_{s\in S}\{\rho_s\}$ and $\rho_{\min} = \min_{s\in S}\{\rho_s\}$):

$$Q_{sz}\Delta t \cdot \Upsilon_{zs'}(t, y - \rho_{\max}\Delta t) \quad \leq \quad g_z(\Delta t) \quad \leq \quad Q_{sz}\Delta t \cdot \Upsilon_{zs'}(t, y - \rho_{\min}\Delta t).$$

Note, that if we divide both bounds by Δt we obtain the same limit for $\Delta t \to 0$, thus

$$\lim_{\Delta t\to 0} \frac{g_z(\Delta t)}{\Delta t} = Q_{sz} \cdot \Upsilon_{zs'}(t, y).$$

iii) The probability that more than one transition occurred in the interval $[0, \Delta t)$ is negligible $(o(\Delta t))$.

The joint distribution is then given by

$$\Upsilon_{ss'}(\Delta t + t, y) \quad = \quad (1 + Q_{ss}\Delta t) \cdot \Upsilon_{ss'}(t, y - \rho_s\Delta t) \, + \sum_{z\neq s} g_z(\Delta t) \, + \, o(\Delta t). \qquad (3.8)$$

Subtracting $\Upsilon_{ss'}(t, y)$ on both sides, dividing by Δt and taking the limit for $\Delta t \to 0$ we obtain the partial derivative with respect to t on the left hand side:

$$\begin{aligned}
\lim_{\Delta t\to 0} \frac{\Upsilon_{ss'}(t+\Delta t, y) - \Upsilon_{ss'}(t, y)}{\Delta t} \quad &= \quad \lim_{\Delta t\to 0} \frac{\Upsilon_{ss'}(t, y-\rho_s\Delta t) - \Upsilon_{ss'}(t, y)}{\Delta t} \\
&+ \quad \lim_{\Delta t\to 0} \frac{Q_{ss}\Delta t \cdot \Upsilon_{ss'}(t, y-\rho_s\Delta t)}{\Delta t} \\
&+ \quad \sum_{z\neq s'} \lim_{\Delta t\to 0} \frac{g_z(\Delta t)}{\Delta t} \\
&+ \quad \lim_{\Delta t\to 0} \frac{o(\Delta t)}{\Delta t} \\
\Rightarrow \quad \frac{\partial\,\Upsilon_{ss'}(t, y)}{\partial t} \quad &= \quad \lim_{\Delta t\to 0} \frac{\Upsilon_{ss'}(t, y-\rho_s\Delta t) - \Upsilon_{ss'}(t, y)}{\Delta t} \\
&+ \quad Q_{ss} \cdot \Upsilon_{ss'}(t, y) \, + \sum_{z\neq s} Q_{sz}\Upsilon_{zs'}(t, y) \\
&= \quad \lim_{\Delta t\to 0} \frac{\Upsilon_{ss'}(t, y-\rho_s\Delta t) - \Upsilon_{ss'}(t, y)}{\Delta t} \\
&+ \quad \sum_{z\in S} Q_{sz}\Upsilon_{zs'}(t, y).
\end{aligned}$$

Define now $\Delta y = \rho_s\Delta t$ and $x = y - \Delta y$. Note that $\Delta y \to 0$ if $\Delta t \to 0$. Replacing $\rho_s\Delta t$ by Δy and y by $x + \Delta y$ we obtain a partial differential equation for $\Upsilon_{ss'}(t, y)$:

$$\begin{aligned}
\frac{\partial\,\Upsilon_{ss'}(t, y)}{\partial t} \quad &= \quad -\rho_s \cdot \lim_{\Delta t\to 0} \frac{\Upsilon_{ss'}(t, x+\Delta y) - \Upsilon_{ss'}(t, x)}{\Delta y} \quad + \quad \sum_{z\in S} Q_{sz} \cdot \Upsilon_{zs'}(t, y) \\
&= \quad -\rho_s \cdot \frac{\partial\,\Upsilon_{ss'}(t, y)}{\partial y} \quad + \quad \sum_{z\in S} Q_{sz} \cdot \Upsilon_{zs'}(t, y)
\end{aligned}$$

$$\Rightarrow \qquad \frac{\partial\, \Upsilon_{ss'}(t,y)}{\partial t} + \rho_s \cdot \frac{\partial\, \Upsilon_{ss'}(t,y)}{\partial y} \;=\; \sum_{z\in S} Q_{sz}\cdot \Upsilon_{zs'}(t,y) \tag{3.9}$$

In matrix notation this becomes

$$\frac{\partial\, \boldsymbol{\Upsilon}(t,y)}{\partial t} + \mathbf{D} \cdot \frac{\partial\, \boldsymbol{\Upsilon}(t,y)}{\partial y} \;=\; \mathbf{Q}\cdot \boldsymbol{\Upsilon}(t,y), \tag{3.10}$$

where $\mathbf{D}$ is a diagonal matrix with $D_{ss} = \rho_s$.

At time 0, only the starting state itself is reachable. Thus, the initial values for the system of PDEs are given by

$$\Upsilon_{ss'}(0,y) = \begin{cases} 1, & s = s' \text{ and } y \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

Sericola [95] derives a similar system of PDEs for the complementary distribution $\overline{\Upsilon}_{ss'}(t,y) = \Pr\{X_t = s', Y_t > y \mid X_0 = s\}$. The system of PDEs (3.10) is of first order, linear and *hyperbolic*, since the diagonal matrix $\mathbf{D}$ is symmetric with real eigenvalues ρ_s.

One possibility to find a solution for this so-called hyperbolic system of PDEs (3.9) is the *method of characteristics* as proposed by Pattipati et al. [82]. The intention is to find so-called characteristic curves $y(t)$ on which the PDEs reduce to ODEs. The knowledge about the solution of ODEs then helps to find a solution for the PDEs.

Let $y(t)$ be an arbitrary curve with time parameter t. We confine the joint distribution to this curve, that is, we consider $\Upsilon_{ss'}(t, y(t))$. Differentiating with respect to t we obtain, using the chain rule,

$$\frac{d\Upsilon_{ss'}(t,y(t))}{dt} = \frac{\partial\Upsilon_{ss'}(t,y(t))}{\partial t}\frac{dt}{dt} \quad + \quad \frac{\partial\Upsilon_{ss'}(t,y(t))}{\partial y}\frac{dy(t)}{dt}. \tag{3.11}$$

We consider now those curves $y(t)$ where

$$\frac{dy(t)}{dt} = \rho_s, \tag{3.12}$$

for which the right-hand side of (3.11) equals the left-hand side of (3.9) (note that $\frac{dt}{dt} = 1$). As long as we move along such a curve, the PDE (3.9) reduces to the ODE

$$\frac{d\Upsilon_{ss'}(t,y(t))}{dt} \;=\; \sum_{z\in S} Q_{sz}\cdot \Upsilon_{zs'}(t,y(t)). \tag{3.13}$$

with initial values $\Upsilon_{ss'}(0, y(0))$. The solution of this ODE is:

$$\Upsilon_{ss'}(t,y(t)) = e^{Q_{ss}t}\left[\int_0^t e^{-Q_{ss}x}\sum_{z\neq s} Q_{sz}\Upsilon_{zs'}(x,y(x))dx \quad + \quad \Upsilon_{ss'}(0,y(0))\right]. \tag{3.14}$$

The curve $y(t)$ is characterised by the ODE (3.12). The general solution for this ODE is

$$y(t) = \rho_s t + C.$$

For a given pair t^* and y^*, we compute the constant C:

$$C = y^* - \rho_s t^*.$$

Thus, for finding the solution of PDE (3.9) at given t^* and y^*, we solve ODE (3.13) on the curve that is given by $y(t) = \rho_s t + y^* - \rho_s t^* = y^* - \rho_s(t^* - t)$, i.e., following (3.15):

$$\Upsilon_{ss'}(t^*, y^*) = e^{Q_{ss}t^*}\Upsilon_{ss'}(0, y^* - \rho_s t^*) \ + \ \int_0^t \sum_{z \neq s} e^{Q_{ss}x} Q_{sz} \Upsilon_{zs'}(t^* - x, y^* - \rho_s x)\, dx \quad (3.15)$$

Note that we replaced x by $t^* - x$ under the integral without altering the value of the integral.

3.5 Summary

In this chapter we characterised the most basic measures for MRMs. For the transient distribution we also described in detail how they can be efficiently computed. For the steady-state distribution we referred to the literature [100]. The joint distribution of state and accumulated reward is characterised by a hyperbolic system of PDEs. Their solution via the method of characteristics leads to a system of integral equations.

We need the joint distribution of state and accumulated reward $\boldsymbol{\Upsilon}(t, y)$ for model checking the CSRL until operator (Section 4.6). No algorithm for computing $\boldsymbol{\Upsilon}(t, y)$ exists that is always the best choice, in contrast to uniformisation for the transient distribution. We postpone the presentation of several solution algorithms to Chapter 5, where we also discuss their specific properties in the context of CSRL model checking.

Chapter 4

Continuous stochastic reward logic – CSRL

The logic CSRL [6] is the formal language we are going to use to specify properties of MRMs. It is the reward extension of CSL [3, 8], which in turn is an extension of the branching time temporal logic CTL [21, 22].

We formally introduce syntax and semantics of the logic CSRL in Section 4.1 and 4.2, respectively. Section 4.3 treats the general model checking procedure for arbitrary CSRL formulas. The computation of the measure needed for checking a next formula is the topic of Section 4.5. The universal way for the computation of the measure for formulas involving the until operator is presented in Section 4.6. Special cases of the until operator are discussed in Section 4.7.

4.1 Syntax

In this section we introduce the syntax of CSRL formulas. They are defined over a set of atomic propositions AP. We distinguish between *state* and *path* formulas. Several building blocks are needed for CSRL formulas: atomic propositions $ap \in AP$, time intervals $I \subseteq \mathbb{R}_{\geq 0}$, reward intervals $J \subseteq \mathbb{R}_{\geq 0}$, a comparison operator $\bowtie \in \{<, \leq, =, \geq, >\}$ and a probability bound $p \in [0, 1]$. State formulas Φ and path formulas φ are then inductively defined:

$$\begin{aligned} \Phi &::= \mathtt{true} \mid ap \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathcal{S}_{\bowtie p}(\Phi) \mid \mathcal{P}_{\bowtie p}(\varphi)\,, \\ \varphi &::= \mathcal{X}^I_J \Phi \mid \Phi \,\mathcal{U}^I_J \Phi. \end{aligned}$$

Note that the definitions of state and path formulas are intertwined and cannot be separated from each other. In the following we denominate the different formulas and describe them in a more precise manner.

The simplest state formula is the constant `true`. Each atomic proposition $ap \in AP$ is a further constant CSRL state formula. If Φ is a CSRL state formula, the application of the Boolean *negation* operator results in the new CSRL formula $\neg\Phi$. If Φ_1 and Φ_2

are CSRL state formulas, the Boolean *conjunction* $\Phi_1 \wedge \Phi_2$ is also a CSRL state formula. Further Boolean connections like disjunction or implication are derived in the obvious way. The *steady-state operator* $\mathcal{S}_{\bowtie p}(\cdot)$ with its comparison operator $\bowtie$ and the probability p is wrapped around a state formula in order to obtain a new state formula. Similarly, the *probability operator* $\mathcal{P}_{\bowtie p}(\cdot)$ can be wrapped around path formulas. Path formulas are build using either the *next operator* $\mathcal{X}$ or the *until operator* $\mathcal{U}$. Both operators are equipped with a *time interval* I (superscript) and a *reward interval* J (subscript). The next operator refers to a single state formula, the structure of the until operator requires two state formulas.

The unary operator $\neg$ has priority over the binary operator $\wedge$. Whenever necessary or convenient we use parentheses to clarify the meaning of the formula. If the time interval I or the reward interval J equals $\mathbb{R}_{\geq 0}$, we can omit it in a CSRL formula.

Example 17. In Example 10 we introduced a set *AP* of atomic propositions which are used as labels for the illustrating MRM. We make use of this set of atomic propositions to build some CSRL example formulas. Thus, `idle`, `active` or `intact` are examples for valid CSRL state formulas. The negation

$$\neg\texttt{intact}$$

is another state formula and so is the nested Boolean formula

$$\texttt{active} \vee \neg\texttt{intact},$$

where for state formulas Φ_1 and Φ_2 the disjunction $\Phi_1 \vee \Phi_2$ is derived as $\neg(\neg\Phi_1 \wedge \neg\Phi_2)$. An example for a steady-state formula is

$$\mathcal{S}_{\leq 0.5}(\texttt{active} \vee \neg\texttt{intact}).$$

A valid path formula built from the next operator is $\mathcal{X}^{[0,0.5]}_{(5,15]}\texttt{idle}$. Wrapped into the probability operator, we obtain the state formula

$$\mathcal{P}_{>0.25}\left(\mathcal{X}^{[0,0.5]}_{(5,15]}\texttt{idle}\right).$$

Finally, an example for an until formula within the probability operator is

$$\mathcal{P}_{<0.2}\left(\texttt{empty}\,\mathcal{U}^{[1,5]}_{(20,50]}(\neg\texttt{intact})\right). \qquad \diamondsuit$$

Sub-logics of CSRL

CSRL originates from CTL [21, 22] which was designed for non-stochastic models (Kripke structures) and therefore has neither a notion of probability nor of real time or even rewards. However, CTL path formulas can be simulated by CSRL path formulas where both the time and the reward interval equal $\mathbb{R}_{\geq 0}$ (and can be omitted). The CTL path quantifier E ("for some paths") can be imitated using the probability operator as

$$\mathsf{E}\varphi \equiv \mathcal{P}_{>0}(\varphi).$$

The CTL path quantifier A ("for all paths") is expressable as

$$\mathsf{A}\varphi \equiv \mathcal{P}_{=1}(\varphi)\,.$$

If we use arbitrary probabilities and comparison operators within the probability operator, we distinguish several sub-logics of CSRL [6] depending on the nature of the time interval I and the reward interval J attached to the path formulas. If both the time and the reward interval impose no actual restriction, that is, if $I = J = \mathbb{R}_{\geq 0} = [0, \infty)$, we obtain a formula of the *stochastic logic* (SL). We omit the intervals in the formula.

If only the reward interval is without influence, that is, if $J = \mathbb{R}_{\geq 0}$, the formula is a *continuous stochastic logic* (CSL) formula. The *dual* logic to CSL is the *continuous reward logic* CRL. Here only the reward interval is specified and the time interval $I = \mathbb{R}_{\geq 0}$ is omitted.

Example 18. The SL version of the next path formula of Example 17 is

$$\mathcal{X}\,\texttt{idle}.$$

Omitting the reward interval in the until formula of Example 17, we obtain a CSL until formula:

$$\texttt{empty}\,\mathcal{U}^{[1,5]}(\neg\texttt{intact}).$$

If we omit the time interval, a CRL formula results:

$$\texttt{empty}\,\mathcal{U}_{(20,50]}(\neg\texttt{intact}).$$ ◊

4.2 Semantics

The syntax of CSRL provides us with a recipe for the *construction* of state and path formulas. With the semantics we introduce the *meaning* of the different operators. CSRL state formulas are interpreted over the states of an MRM and CSRL path formulas are interpreted over the paths of an MRM.

The semantics of CSRL is formally defined by a *satisfaction relation* $\models$. We have actually two relations, one between an MRM, one of its states s, and the CSRL state formulas and one between an MRM, a path ω in this MRM and the CSRL path formulas. We say that a state s *satisfies* the state formula Φ, is a Φ-state or that Φ *holds in* s if $\mathcal{M}, s \models \Phi$. Similarly, a path satisfies a path formula φ or is a φ-path if $\mathcal{M}, \omega \models \varphi$. Whenever it is clear from the context, we omit the explicit indication of $\mathcal{M}$.

State formulas

All states of an MRM satisfy the formula `true`:

$$s \models \texttt{true} \text{ for all } s \in S.$$

A state s satisfies an atomic proposition $ap \in AP$ if ap is contained in the labelling of s:

$$s \models ap \iff ap \in L(s).$$

The negation of a state formula Φ holds in a state s if Φ is not satisfied by s:

$$s \models \neg\Phi \iff s \not\models \Phi.$$

The conjunction of two state formulas Φ_1 and Φ_2 is satisfied by s if s is both a Φ_1-state and a Φ_2-state:

$$s \models \Phi_1 \wedge \Phi_2 \iff s \models \Phi_1 \text{ and } s \models \Phi_2.$$

A steady-state formula $\mathcal{S}_{\bowtie p}(\Phi)$ holds in a state s if – having started in state s – the cumulative steady-state probability (Section 3.3) of all Φ-states meets the bound $\bowtie p$:

$$s \models \mathcal{S}_{\bowtie p}(\Phi) \iff \sum_{s' \in \mathsf{Sat}(\Phi)} \Pi_{ss'} \bowtie p.$$

In case of the probability operator, all paths (cf. Section 2.5) starting in state s that satisfy the involved path formula φ (see below) have to be considered. Let

$$\mathit{Prob}_s(\varphi) = P_s\{\omega \in \Omega \mid \omega \models \varphi\}$$

be the probability measure of all these paths (Section 2.5). The measure $\mathit{Prob}_s(\varphi)$ has to meet the bound $\bowtie p$ in order for s to satisfy $\mathcal{P}_{\bowtie p}(\varphi)$:

$$s \models \mathcal{P}_{\bowtie p}(\varphi) \iff \mathit{Prob}_s(\varphi) \bowtie p.$$

Path formulas

CSRL path formulas are interpreted over the finite and infinite paths of an MRM. A next formula refers to the state the path occupies after the first transition. In order to satisfy a next formula $\mathcal{X}^I_J\Phi$, a path ω must have at least length 1. The residence time $t_0(\omega)$ of the first state has to fall in the interval I, the reward $y_0(\omega)$ accumulated in the first state in the interval J. The next state $s_1(\omega)$ must be a Φ-state:

$$\omega \models \mathcal{X}^I_J\Phi \iff |\omega| \geq 1 \ \wedge\ t_0(\omega) \in I \ \wedge\ y_0(\omega) \in J \ \wedge\ s_1(\omega) \models \Phi.$$

The most involved formula is the until formula. It refers to the future evolution of the path without making any restriction on the number of steps. A path ω satisfies $\Phi\,\mathcal{U}^I_J\Psi$ if there is a time $t \in I$ such that

- the state of the path at time t satisfies Ψ: $X_t(\omega) \models \Psi$;
- the state of the path at all times t' before t satisfies Φ: $\forall t' \in [0,t).X_{t'}(\omega) \models \Phi$;

- the accumulated reward at time t falls in J: $Y_t(\omega) \in J$.

$$\omega \models \Phi \mathcal{U}^I_J \Psi \quad \iff \quad \exists t \in I.(X_t(\omega) \models \Psi \ \wedge\ (\forall t' \in [0,t).X_{t'}(\omega) \models \Phi) \ \wedge\ Y_t(\omega) \in J)$$

Example 19. Recall the example MRM summarised in Section 2.9. Any path in this example of the form

$$\omega_2 = 2 \xrightarrow{0.1} 0 \cdots$$

satisfies the next formula $\mathcal{X}^{[0,0.5]}_{(5,15]}\texttt{idle}$ because the residence time of the first state is in the time interval $[0, 0.5]$, the accumulated reward in the first state is $100 \cdot 0.1 = 10$ which falls in the reward interval $(5, 15]$ and the second state 0 is an `idle`-state.

Example paths that satisfy the until formula $\texttt{empty}\,\mathcal{U}^{[1,5]}_{(20,50]}(\neg\texttt{intact})$ start with the following sequence:

$$\omega_3 = 0 \xrightarrow{0.2} 1 \xrightarrow{0.6} 0 \xrightarrow{0.3} 3 \cdots$$

Consider time 1.1. The state of the path at time 1.1 satisfies $(\neg\texttt{intact})$ because $X_{1.1}(\omega_3) = 3$. The accumulated reward is $Y_{1.1}(\omega_3) = 37 \in (20, 50]$ and before time 1.1 only the `empty`-states 0 and 1 occur on the path. ♢

4.3 The model checking procedure

Normally we are not only concerned whether one single state satisfies a CSRL state formula but we want to check the formula for *all* states. All Φ-states of an MRM $\mathcal{M}$ are subsumed in the *satisfaction set*

$$\mathsf{Sat}^{\mathcal{M}}(\Phi) = \{s \in S \mid \mathcal{M}, s \models \Phi\}.$$

We omit the superscript $\mathcal{M}$ if it is clear from the context.

Model checking a CSRL state formula Φ now encompasses the determination of the satisfaction set $\mathsf{Sat}(\Phi)$. We present the procedure that determines the satisfaction set of any CSRL state formula. The approach is inherited from the non-stochastic branching time logic CTL: we determine the satisfaction set according to the recursive structure of the formula, as follows.

All states satisfy `true`, hence,

$$\mathsf{Sat}(\texttt{true}) = S.$$

For the satisfaction set of an atomic proposition we have to consider the labelling of all states:

$$\mathsf{Sat}(ap) = \{s \in S \mid ap \in L(s)\}.$$

The satisfaction set of a negated formula $\neg\Phi$ is given by the state space without those states that satisfy Φ:

$$\mathsf{Sat}(\neg\Phi) = S \setminus \mathsf{Sat}(\Phi).$$

For the conjunction of two state formulas Φ_1 and Φ_2, the satisfaction set is

$$\mathsf{Sat}(\Phi_1 \wedge \Phi_2) = \mathsf{Sat}(\Phi_1) \cap \mathsf{Sat}(\Phi_2),$$

for a disjunction it is

$$\mathsf{Sat}(\Phi_1 \vee \Phi_2) = \mathsf{Sat}(\Phi_1) \cup \mathsf{Sat}(\Phi_2).$$

For the steady-state operator we first have to determine the steady-state probabilities $\mathbf{\Pi}$. Then we sum over all Φ-states and compare according to $\bowtie p$ (following closely the semantics of the steady-state operator):

$$\mathsf{Sat}\left(\mathcal{S}_{\bowtie p}(\Phi)\right) = \left\{ s \in S \;\middle|\; \sum_{s' \in \mathsf{Sat}(\Phi)} \Pi_{ss'} \;\bowtie\; p \right\}.$$

Finally we consider the probability operator $\mathcal{P}_{\bowtie p}\left(\varphi\right)$ in combination with a path formula φ. First we have to determine the vector $\mathit{Prob}(\varphi)$. In Section 4.5 we discuss, how this is done for next formulas, Section 4.6 is dedicated to until formulas. After the calculation of $\mathit{Prob}(\varphi)$ we compare its entries according to $\bowtie p$ in order to obtain the satisfaction set:

$$\mathsf{Sat}\left(\mathcal{P}_{\bowtie p}\left(\varphi\right)\right) = \left\{ s \in S \mid \mathit{Prob}_s(\varphi) \bowtie p \right\}.$$

Example 20. We interpret the CSRL formulas of Example 17 in the running example MRM (cf. Section 2.9). The satisfaction set of the atomic proposition `intact` consists of all states labelled accordingly, thus

$$\mathsf{Sat}(\texttt{intact}) = \{0, 1, 2\}.$$

For its negation it is

$$\mathsf{Sat}(\neg\texttt{intact}) = S \setminus \{0, 1, 2\} = \{3\}.$$

The satisfaction set of $(\texttt{active} \vee \texttt{idle})$ is derived as

$$\mathsf{Sat}(\texttt{active} \vee \texttt{idle}) \quad = \quad \mathsf{Sat}(\texttt{active}) \cup \mathsf{Sat}(\texttt{idle}) \quad = \quad \{2\} \cup \{0\} = \{0, 2\}.$$

The probability measures for the state formula involving the next operator are (cf. Example 22)

$$\mathit{Prob}(\mathcal{X}^{[0,0.5]}_{[5,15]}\texttt{idle}) = (0, 0.2387, 0.3420, 0),$$

which results in the following satisfaction set for the state formula $\mathcal{P}_{>0.25}\left(\mathcal{X}^{[0,0.5]}_{[5,15]}\texttt{idle}\right)$:

$$\mathsf{Sat}(\mathcal{P}_{>0.25}\left(\mathcal{X}^{[0,0.5]}_{[5,15]}\texttt{idle}\right)) = \{2\}.$$

In Section 5.6.3 we compute the probability measure for the until formula as

$$\mathit{Prob}(\texttt{empty}\,\mathcal{U}^{[1,5]}_{[20,50]}(\neg\texttt{intact})) = (0.0165, 0.0472, 0, 1)$$

and the satisfaction set for the corresponding state formula is

$$\mathsf{Sat}(\mathcal{P}_{<0.2}\left(\texttt{empty}\,\mathcal{U}^{[1,5]}_{[20,50]}(\neg\texttt{intact})\right)) = \{0, 1, 2\}.$$

◊

4.4 Duality

This section deals with the *duality* of time and accumulated reward. We show that the roles of time and reward can be interchanged in both MRMs and CSRL formulas. This result facilitates the model checking of formulas involving path formulas, especially when the until operator is involved.

MRMs have two continuous ingredients: the continuous time parameter and the continuous accumulation of reward. Progress in time induces reward accumulation (at a rate specified by the current state) and reward accumulation demands the time to go by. In a path ω, the state sequence together with the residence times precisely determines the accumulated reward at any time t. Vice versa, we can deduct the current time for a given path, if we know the accumulated reward. This brings us to the conjecture that time and accumulated reward are somehow interchangeable.

Recall that $X_t(\omega)$ is the state occupied by path ω at time t and $Y_t(\omega)$ is the accumulated reward by time t. Now define $X_y^{-1} : \Omega \to S$ to give the state occupied by a path for a given accumulated reward y:

$$X_y^{-1}(\omega) = X_t(\omega) \qquad \Longleftrightarrow \qquad Y_t(\omega) = y.$$

Following the same concept we define the time at which a given reward y is accumulated as Y_y^{-1}:

$$Y_y^{-1}(\omega) = t \qquad \Longleftrightarrow \qquad Y_t(\omega) = y.$$

Note that we only obtain a proper definition of the stochastic processes X_y^{-1} and Y_y^{-1} if there are no two points in time with the same accumulated reward y. In order to accomplish this restriction, the reward rates of all states must be positive, such that the accumulated reward is strictly monotonously increasing.

Instead of defining new random variables X_y^{-1} and Y_y^{-1} on the probability space defined by an MRM $\mathcal{M} = (AP, S, \mathbf{R}, L, \rho)$ we can also derive a *dual* MRM $\mathcal{M}^{-1}$ where the role of time and accumulated reward are interchanged. By the above argumentation this is only possible if $\rho_s > 0$ for all $s \in S$. The dual MRM $\mathcal{M}^{-1} = ((S, \mathbf{R}^{-1}), AP, L, \rho^{-1})$ results from $\mathcal{M}$ as follows.

i) Rescale the transition rates by the reward rate of their originating state:

$$R^{-1}_{ss'} = \frac{R_{ss'}}{\rho_s}.$$

ii) Invert the entries of the reward vector:

$$\rho'_s = \frac{1}{\rho_s}.$$

If the MRM $\mathcal{M}$ spends t time units in a state s, it accumulates $\rho_s \cdot t$ reward. Conversely, if the dual MRM M^{-1} spends $\rho_s \cdot t$ time units in state s, it accumulates $1/\rho_s \cdot \rho_s \cdot t = t$

reward. Thus, the roles of time and reward are interchanged. The one-step probability matrix $\mathbf{P}^{\mathcal{M}^{-1}} = \mathbf{P}^{\mathcal{M}}$ remains unchanged because all transitions starting from a state s are scaled by the same factor $1/\rho_s$.

Example 21. The dual MRM for the example MRM has rate matrix

$$\mathbf{R}^{-1} = \begin{pmatrix} 0 & 0.06 & 0.12 & 0.02 \\ 0.1 & 0 & 0 & 0 \\ 0.08 & 0 & 0.12 & 0 \\ 0.2 & 0 & 0 & 0 \end{pmatrix}$$

and reward vector

$$\rho^{-1} = (0.02, 0.05, 0.01, 0.2). \qquad \Diamond$$

CSRL next and until formulas impose restrictions on time and accumulated reward by the intervals I and J specified in the formulas. These intervals can be interchanged when switching from MRM $\mathcal{M}$ to MRM $\mathcal{M}^{-1}$ without changing the probability measure of the corresponding paths.

Lemma 4 ([6, Lemma 1]). Let $\mathcal{M}$ be an MRM with $\rho_s > 0$ for all $s \in S$. Let Φ and Ψ be CSRL state formulas. Then for every state s,

i) $Prob^{\mathcal{M}}(s, \mathcal{X}^I_J \Phi) = Prob^{\mathcal{M}^{-1}}(s, \mathcal{X}^J_I \Phi)$ and

ii) $Prob^{\mathcal{M}}(s, \Phi \mathcal{U}^I_J \Psi) = Prob^{\mathcal{M}^{-1}}(s, \Phi \mathcal{U}^J_I \Psi)$.

For a CSRL formula Φ the dual formula Φ^{-1} is defined by swapping the time and reward intervals in any next or until sub-formula of Φ. The satisfaction set $\mathsf{Sat}^{\mathcal{M}}(\Phi)$ for an MRM $\mathcal{M}$ can be determined in $\mathcal{M}^{-1}$: $\mathsf{Sat}^{\mathcal{M}}(\Phi) = \mathsf{Sat}^{\mathcal{M}^{-1}}(\Phi^{-1})$.

Theorem 2 ([6, Theorem 1]). Let $\mathcal{M}$ be an MRM with $\rho_s > 0$ for all $s \in S$ and let Φ be a CSRL state formula. Then

$$\mathsf{Sat}^{\mathcal{M}}(\Phi) = \mathsf{Sat}^{\mathcal{M}^{-1}}(\Phi^{-1}).$$

In Section 4.6 we classify until formulas according to the type of the intervals I and J. The duality result then comes in conveniently to derive model checking procedures for some of the formulas.

4.5 The probability measure for next formulas

The probability operator is wrapped around path formulas. The model checking procedure requires the computation of the probability measure of the involved next or until formula.

For the computation of $Prob_s(\mathcal{X}^I_J \Phi)$ we have to determine the probability measure of all paths satisfying the next formula. By the semantics, these are all paths of at least length 1, starting in s, with a residence time in I of the first state, an accumulated reward in J before the first transition, and a second state that satisfies Φ. If for a state s the probability measure $Prob_s(\mathcal{X}_J^I \Phi) \bowtie p$, the formula holds in state s. Define I_s^J to be the subinterval of I containing only those residence times in I that result in an accumulated reward in J when residing exclusively in state s:

$$I_s^J = \{t \in I \mid \rho_s \cdot t \in J\}.$$

Then the set of paths starting in a given state s and satisfying the next formula $\mathcal{X}_J^I \Phi$ consists of all paths where the residence time of the first state s falls into the interval I_s^J and the second state satisfies Φ. This set of paths is a union of cylinder sets (cf. Section 2.5.1):

$$\bigcup_{s' \in \mathsf{Sat}(\Phi)} C\left((s, s'), (I_s^J, \mathbb{R}_{\geq 0})\right),$$

and is therefore a measurable set. The probability measure of the next formula in state s is consequently given by (cf. Section 2.5.2)

$$Prob_s(\mathcal{X}_J^I \Phi) = P_s\left(\bigcup_{s' \in \mathsf{Sat}(\Phi)} C\left((s, s'), (I_s^J, \mathbb{R}_{\geq 0})\right)\right) = \Pr\left\{\delta_s \in I_s^J\right\} \cdot \sum_{s' \in \mathsf{Sat}(\Phi)} \mathbf{P}_{ss'}, \tag{4.1}$$

that is, the probability of spending an appropriate amount of time in state s multiplied with the probability of reaching a Φ-state in one step. We can use (4.1) directly to compute $Prob_s(\mathcal{X}_J^I \Phi)$.

The satisfaction set $\mathsf{Sat}\left(\mathcal{P}_{\bowtie p}\left(\mathcal{X}_J^I \Phi\right)\right)$ emerges from comparing the entries of $Prob(\mathcal{X}_J^I \Phi)$ with p using the comparison operator $\bowtie$.

Example 22. The example next formula $\mathcal{X}_{(5,15]}^{[0,0.5]}\texttt{idle}$ has intervals $I = [0, 0.5]$ and $J = (5, 15]$ which gives us the following tailored time intervals

$$I_0^J = [0.1, 0.3],\ I_1^J = [0.25, 0.5],\ I_2^J = [0.05, 0.15],\ I_3^J = \varnothing.$$

The probabilities of the appropriate residence times are

$$\begin{array}{lcl} \Pr\left\{\delta_0 \in I_0^J\right\} & = & e^{-10\cdot 0.1} - e^{-10\cdot 0.3} & = & 0.3181, \\ \Pr\left\{\delta_1 \in I_0^J\right\} & = & e^{-2\cdot 0.25} - e^{-2\cdot 0.5} & = & 0.2387, \\ \Pr\left\{\delta_2 \in I_0^J\right\} & = & e^{-20\cdot 0.05} - e^{-20\cdot 0.15} & = & 0.8551, \\ \Pr\left\{\delta_3 \in I_0^J\right\} & = & 0. \end{array}$$

The resulting probability measures for the next formula then equal

$$Prob_0(\mathcal{X}_{(5,15]}^{[0,0.5]}\texttt{idle}) = \Pr\left\{\delta_0 \in I_0^J\right\} \cdot \mathbf{P}_{00} \qquad = 0.3181 \cdot 0 = 0, \tag{4.2}$$

$$Prob_1(\mathcal{X}_{(5,15]}^{[0,0.5]}\texttt{idle}) = \Pr\left\{\delta_1 \in I_1^J\right\} \cdot \mathbf{P}_{10} \qquad = 0.2387 \cdot 1 = 0.2387, \tag{4.3}$$

$$Prob_2(\mathcal{X}_{(5,15]}^{[0,0.5]}\texttt{idle}) = \Pr\left\{\delta_2 \in I_2^J\right\} \cdot \mathbf{P}_{20} \qquad = 0.8551 \cdot 0.4 = 0.3420, \tag{4.4}$$

$$Prob_3(\mathcal{X}_{(5,15]}^{[0,0.5]}\texttt{idle}) = \Pr\left\{\delta_3 \in I_3^J\right\} \cdot \mathbf{P}_{30} \qquad = 0 \cdot 1 = 0. \tag{4.5}$$

Comparing the entries of the vector

$$Prob(\mathcal{X}^{[0,0.5]}_{(5,15]}\,\texttt{idle}) = (0, 0.2387, 0.3420, 0)$$

with 0.25, we obtain the satisfaction set

$$\mathsf{Sat}(\mathcal{P}_{>0.25}\left(\mathcal{X}^{[0,0.5]}_{(5,15]}\,\texttt{idle}\right)) = \{2\}. \qquad \lozenge$$

4.6 The probability measure for until formulas

This section deals with the computation of the probability measure for until formulas. We derive recursive equations that characterise the measure. Unfortunately, the characterisation of the probability measure of until formulas does not give us a straightforward recipe for the computation as it was the case for the next operator. However, the problem can be mapped to the problem of computing the joint distribution of time and accumulated reward $\boldsymbol{\Upsilon}(t,y)$ in a transformed MRM which is inhomogeneous with respect to time and accumulated reward. In many cases (depending on the time and reward intervals), this MRM actually reduces to a homogeneous one. In this section we only characterise the probability measure; actual algorithms for the computation of the joint distribution of homogeneous and inhomogeneous MRMs are the topic of Chapter 5.

We start with defining equations for the probability measure of an until formula in Section 4.6.1. In Section 4.6.2 we show how the computation of the probability measure is reduced to the computation of the joint distribution of state and accumulated reward in a time- and reward-inhomogeneous MRM.

4.6.1 Characterisation of $\boldsymbol{Prob}_s\big(\Phi\,\mathcal{U}^I_J\Psi\big)$

In the following we categorise states according to the formulas they satisfy. For all states s we give an intuitively derived characterisation of $Prob_s\big(\Phi\,\mathcal{U}^I_J\Psi\big)$. Recall that

$$Prob_s\left(\Phi\,\mathcal{U}^I_J\Psi\right) = P_s\left\{\omega\in\Omega \mid \omega\models\Phi\,\mathcal{U}^I_J\Psi\right\}$$

and the satisfaction relation between paths and until formulas is given by

$$\omega\models\Phi\,\mathcal{U}^I_J\Psi \quad\iff\quad \exists t\in I.(X_t(\omega)\models\Psi\ \wedge\ (\forall t'\in[0,t).X_{t'}(\omega)\models\Phi)\ \wedge\ Y_t(\omega)\in J).$$

For an interval $K\subseteq\mathbb{R}$ and $x\in\mathbb{R}_{\geq 0}$, define

$$K\ominus x = \{t-x \mid t\in K\}$$

to be the interval resulting when shifting the interval K by x units to the left. Note that in the following equations there appear reward intervals with elements that are below zero. While this is not allowed by the syntax of until formulas, it does not introduce any difference in the probability measure: since the accumulated reward in an MRM is always nonnegative, the probability measure for the negative part is zero.

Depending on the formulas a state s satisfies, and on the properties of the intervals I and J, we distinguish five cases.

i) If

$$s \models \Psi \text{ and } \inf I = \inf J = 0, \text{ that is, } \inf I_s^J = 0,$$

all paths starting in s satisfy the until formula for $t = 0$ and so we have

$$\mathit{Prob}_s\big(\Phi\, \mathcal{U}_J^I \Psi\big) = 1.$$

ii) For

$$s \models (\neg\Phi \wedge \neg\Psi)$$

no path starting in s satisfies $\Phi\, \mathcal{U}_J^I \Psi$. Consequently,

$$\mathit{Prob}_s\big(\Phi\, \mathcal{U}_J^I \Psi\big) = 0.$$

iii) For

$$s \models \neg\Phi \wedge \Psi \text{ and } \inf I > 0 \text{ or } \inf J > 0, \text{ that is, } \inf I_s^J > 0,$$

all paths starting in s do not satisfy Φ in the first state and hence do not satisfy $\Phi\, \mathcal{U}_J^I \Psi$. It makes no difference that $s \models \Psi$, because a Ψ-state should be reached at $\inf I$ at the earliest. Hence, the probability measure is also

$$\mathit{Prob}_s\big(\Phi\, \mathcal{U}_J^I \Psi\big) = 0.$$

iv) If

$$s \models (\Phi \wedge \neg\Psi),$$

a path starting in s might satisfy the until formula. To do so, it must leave s before $\sup I_s^J$ time has passed because otherwise either the upper time bound or the upper reward bound is exceeded. After it has left s it has to reach a Ψ-state (passing only along Φ-states). We integrate over the density of residing a given time in state s, moving to one of the other states and satisfying the until formula with shorter time and reward intervals starting in these states, i.e.,

$$\mathit{Prob}_s\big(\Phi\, \mathcal{U}_J^I \Psi\big) = \int_0^{\sup I_s^J} \sum_{z \neq s} e^{Q_{ss}\cdot x} Q_{sz} \cdot \mathit{Prob}_z\big(\Phi\, \mathcal{U}_{J\ominus\rho_s\cdot x}^{I\ominus x} \Psi\big)\, dx.$$

Note that $e^{Q_{ss}x} Q_{sz} = Q_{ss} e^{Q_{ss}x} P_{sz}$ is the density of residing x time units in state s and the moving on to state s'.

v) The case

$$s \models (\Phi \wedge \Psi),$$

is similar to iv), but a path satisfies also the until formula if the MRM remains in state s for at least $\inf I_s^J$ time units. Furthermore, the integral only has to be evaluated up to time $\inf I_s^J$ because at a later point in time it is not necessary to move on to another state, because the until formula is already satisfied anyway.

$$Prob_s\big(\Phi\,\mathcal{U}_J^I \Psi\big) = e^{Q_{ss} \inf I_s^J} + \int_0^{\inf I_s^J} \sum_{z \neq s} e^{Q_{ss}\cdot x} Q_{sz} \cdot Prob_z\big(\Phi\,\mathcal{U}_{J\ominus\rho_s\cdot x}^{I\ominus x}\Psi\big)\; dx,$$

where $e^{Q_{ss} \inf I_s^J} = \Pr\left\{\delta_s \geq \inf I_s^J\right\}$.

The integrals in the equations imply that it suffices to consider *closed* time intervals [7]. For the accumulated reward, things are different, we have to observe carefully whether the interval is left- or right-open or -closed, because there are discontinuities ("jumps") in the distribution of the accumulated reward. For the sake of simplicity only the following types of reward intervals are considered: no restriction ($J = [0, \infty)$), fixed amount of reward ($J = [y, y]$), below or equal to a given threshold ($J = [0, y]$), left-open and right-closed ($J = (y_1, y_2]$), above a given threshold ($J = (y, \infty)$).

Example 23. We distinguish the different cases for the states of the running example MRM and the until formula $\mathtt{empty}\,\mathcal{U}_{(20,50]}^{[1,5]}(\neg\mathtt{intact})$. State 0 (`idle`) is an `empty`-state and also an `intact`-state and so case iv) has to be applied. The same is true for state 1 (`sleeping`). State 2 is not an `empty`-state but an `intact`-state. This is case ii). Finally, state 3 satisfies only ($\neg$`intact`): case iii) . ◊

4.6.2 $\Upsilon(t, y)$ in time- and reward-inhomogeneous MRM

CSRL formulas are interpreted over the states and paths of MRMs as defined in Chapter 2. These MRMs are homogeneous with respect to time and reward, that is, the transition rates and the reward rates remain unchanged, whatever the current time or accumulated reward is. However, in this section we show that in the context of model checking CSRL formulas involving the until operator it is useful to consider so-called time- and reward-inhomogeneous MRMs, in which transition and reward rates might depend on the current time and accumulated reward. The computation of $Prob_s(\Phi\,\mathcal{U}_J^I\Psi)$ can then be reduced to the computation of the joint distribution of state and accumulated reward $\Upsilon(t, y)$ of an inhomogeneous MRM derived from the original homogeneous MRM. The basic idea is to change the behaviour of the MRM when both, the given lower time *and* reward bound, are exceeded. The evolution of the MRM proceeds in two phases: the first phase lasts until both lower bounds are exceeded, the second phase follows.

We compute the probability of residing in a Ψ-state at the end of the second phase having accumulated an amount of reward that lies inside the reward interval. During the first phase only Φ-states are allowed, all others are made absorbing. In the second phase, the $\neg\Phi$-states remain absorbing but additionally also the Ψ-states are made absorbing.

Unfortunately, one problem arises: if a path is in a state not satisfying Φ but satisfying Ψ at the end of phase 1 and stays there until phase 2 has started, its probability mass is wrongly included into the probability measure. To bypass this problem, we duplicate these $(\neg\Phi \wedge \Psi)$-states. In the first phase we use only the duplicates. In the second phase, the original states are employed. Only if a path is in an original Ψ-state at the end of the second phase, the path satisfies the until formula.

For an homogeneous MRM $\mathcal{M} = (AP, S, \mathbf{R}, L, \rho)$ and the until formula $\Phi\,\mathcal{U}^I_J\Psi$ we define the inhomogeneous MRM $\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\Psi\rangle = (\varnothing, S^*, \mathbf{R}(t,y), L^*, \rho(t,r))$ which encodes the until formula into the model.

- There are no atomic propositions,
- Let
  $$\overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi)} = \{\bar{z} \mid z \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi)\}$$
  be a set of duplicates of $(\neg\Phi\wedge\Psi)$-states. Then the state space $S^* = S \uplus \overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi)}$ consists of the original state space plus duplicates for all $(\neg\Phi \wedge \Psi)$-states.
- The rate matrix $\mathbf{R}(t,y)$ is time- and reward-dependent. The values of t and y determine which version to use:
  $$\mathbf{R}(t,y) = \begin{cases} \mathbf{R}^{(1)}, & t < \inf I \text{ or } y < \inf J \\ \mathbf{R}^{(2)}, & t \geq \inf I \text{ and } y \geq \inf J. \end{cases}$$
  In phase 1 ($t < \inf I$ or $y < \inf J$) $\mathbf{R}^{(1)}$ is used. All rates emerging from $\neg\Phi$-states are set to zero, thus these states are made absorbing. If a transition leads to a $(\neg\Phi \wedge \Psi)$-state, it is redirected to the corresponding duplicated state. The original $(\neg\Phi \wedge \Psi)$-states are not reachable in the first phase. Thus, we have:
  $$R^{(1)}_{ss'} = \begin{cases} 0, & s \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi), \\ 0, & s' \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi), \\ 0, & s \in \overline{\mathsf{Sat}^{M}(\neg\Phi \wedge \Psi)}, \\ R_{sz}, & s' = \bar{z} \in \overline{\mathsf{Sat}^{M}(\neg\Phi \wedge \Psi)}, \\ R_{ss'}, & \text{otherwise.} \end{cases}$$
  In phase 2, $\neg\Phi$-states remain absorbing and all Ψ-states are additionally made absorbing. The transitions to the duplicates are redirected to their original destinations. The duplicates are not connected anymore. Hence, we have:
  $$R^{(2)}_{ss'} = \begin{cases} 0, & s \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi), \\ 0, & s \in \mathsf{Sat}^{\mathcal{M}}(\Psi), \\ 0, & s \text{ or } s' \in \overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi)}, \\ R_{ss'}, & \text{otherwise.} \end{cases}$$

- The labelling function is trivial since there are no atomic propositions.
- The reward vector $\rho(t, y)$ contains the original rewards for the states except for the ones that are made absorbing or are duplicates. Consequently, there are also two versions of the reward vector with
$$\rho(t,r) = \begin{cases} \rho^{(1)}, & t < \inf I \text{ or } y < \inf J, \\ \rho^{(2)}, & t \geq \inf I \text{ and } y \geq \inf J, \end{cases}$$
with
$$\rho_s^{(1)} = \begin{cases} 0, & s \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi), \\ 0, & s \in \overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi)}, \\ \rho_s, & \text{otherwise}, \end{cases}$$
and
$$\rho_s^{(2)} = \begin{cases} 0, & s \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi), \\ 0, & s \in \mathsf{Sat}^{\mathcal{M}}(\Psi), \\ 0, & s \in \overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi)}, \\ \rho_s, & \text{otherwise}. \end{cases}$$

The transition and reward rates of $\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle$ do not change continuously. In the course of a path ω there is exactly one point in time at which the behaviour changes (when both the time parameter and the accumulated reward exceed their lower bounds).

In accordance with the two versions of the rate matrix and reward vector we will denote the two MRMs constituting the inhomogeneous MRM $\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle$ as $\mathcal{M}^{(1)}$ and $\mathcal{M}^{(2)}$. There are also two versions of the generator matrix, denoted $\mathbf{Q}^{(1)}$ and $\mathbf{Q}^{(2)}$.

Example 24. We derive the inhomogeneous MRM originating from the running example MRM and the until formula $\varphi = \texttt{empty}\,\mathcal{U}_{[2,5]}^{[0.1,0.5]}(\neg\texttt{intact})$. Figure 4.1 shows the graphs of the two phases of the resulting MRM. State 3 is a $(\neg\texttt{empty} \wedge (\neg\texttt{intact}))$-state and has to be duplicated. The reward vectors are
$$\rho^{(1)} = (50, 20, 0, 5, 0),\ \rho^{(2)} = (50, 20, 0, 0, 0),$$
where the fifth entry is for duplicated state $\bar{3}$. ◊

The following theorem allows us to express the probability measure of an until formula $\mathit{Prob}_s^{\mathcal{M}}(\Phi\,\mathcal{U}_J^I\Psi)$ in terms of the joint distribution in the corresponding inhomogeneous MRM.

Theorem 3. Let $\mathcal{M} = (\mathit{AP}, S, R, L, \rho)$ be a homogeneous MRM and let $\Phi\,\mathcal{U}_J^I\Psi$ be a CSRL until formula. For $s \in S$, define
$$\mathit{initial}(s) = \begin{cases} \bar{s}, & s \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi), \\ s, & \text{otherwise}. \end{cases}$$
Then
$$\mathit{Prob}_s^{\mathcal{M}}(\Phi\,\mathcal{U}_J^I\Psi) = \sum_{s' \in \mathsf{Sat}(\Psi)} \Upsilon_{\text{initial}(s)s'}^{\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle}(\sup I, \sup J)$$

The proof is straightforward but lengthy, we therefore shift it to Appendix A.

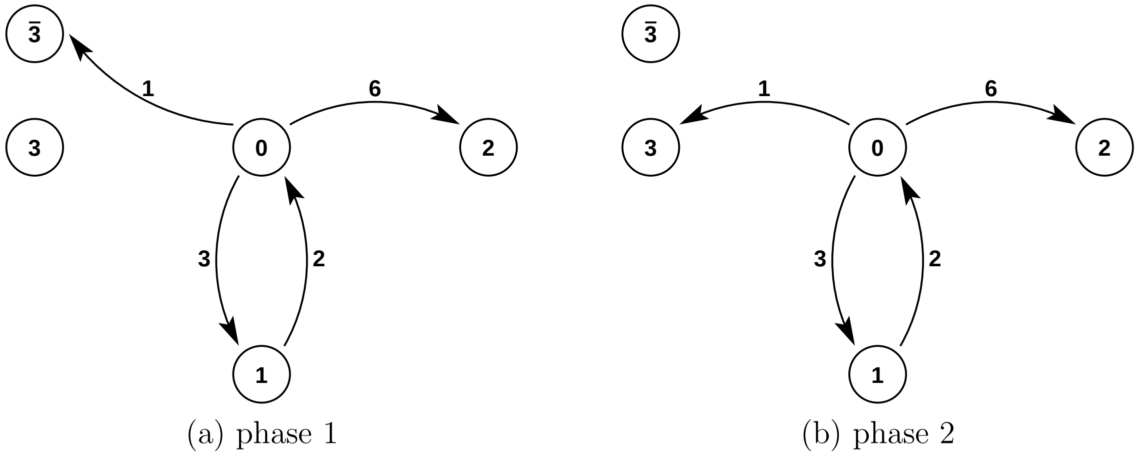


Figure 4.1: The two phases of the inhomogeneous MRM $\mathcal{M}\langle\texttt{empty}\,\mathcal{U}^{[1,5]}_{[20,50]}(\neg\texttt{intact})\rangle$

4.7 Overview of until formulas

Not every combination of a time interval I and a reward interval J within an until formula $\Phi\,\mathcal{U}^I_J\Psi$ actually leads to a inhomogeneous MRM. Often only one phase of the MRM $\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\Psi\rangle$ is used. This is tantamount to a computation in a homogeneous MRM. However, we always have to adapt the original MRM in order to support the calculation of the probability measure for the until formula using Theorem 3. For this purpose we introduce some convenient notation.

For a CSRL state formula ϕ, the MRM $\mathcal{M}[\phi]$ arises from an MRM $\mathcal{M} = (AP, S, \mathbf{R}, L, \rho)$ by making all ϕ-states absorbing and setting their reward rate to 0. Formally, the rate matrix $\mathbf{R}'$ and the reward vector ρ' of $\mathcal{M}[\phi] = (AP, S, \mathbf{R}', L, \rho')$ are defined by

$$R'_{ss'} = \begin{cases} 0, & s \in \mathsf{Sat}^{\mathcal{M}}(\phi), \\ R_{ss'}, & \text{otherwise,} \end{cases}$$

and

$$\rho'_s = \begin{cases} 0, & s \in \mathsf{Sat}^{\mathcal{M}}(\phi), \\ \rho_s, & \text{otherwise.} \end{cases}$$

If only one phase of the MRM is actually used in the computation, it is not necessary to duplicate states as explained in Section 4.6.2. For until formula $\Phi\,\mathcal{U}^I_J\Psi$ and an MRM $\mathcal{M}$, the first phase is actually $\mathcal{M}[\neg\Phi]$ and the second phase is $\mathcal{M}[\neg\Phi \vee \Psi]$ [7].

Table 4.1 gives an overview over the different CSRL until formulas according to the structure of the intervals I and J. Remember that it suffices to consider only closed time intervals and that we do not consider all possible reward intervals (cf. Section 4.6.1). Not all of the until formulas require the computation of the joint distribution of state and reward in a time- and reward-inhomogeneous MRM; often the joint distribution reduces to a simpler measure or one only needs a homogeneous MRM. Many measures can be computed using the duality result of Section 4.4, as long as the reward rates are appropriate. In the table we state which algorithm to choose, which MRM to employ and in which section to find this particular case. If the duality result (Theorem 2) should be applied, we indicate the interchanged time and reward interval. The duality theorem is only properly applicable if all reward rates are strictly positive. The MRM $\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\,\Psi\rangle$ has always states with reward rates 0, namely the states that are explicitely made absorbing. However, there are no transitions that leave these states. Since the structure is preserved in the dual MRM, we can safely set $R'_{ss'}(t,y) = R_{ss'}(t,y) = 0$ and $\rho'_s(t,y) = \rho_s(t,y) = 0$ for a state s which is absorbing for time t and accumulated reward y, i.e., for which $R_{ss'}(t,y) = 0$ for all s'. That is, if $\rho_s(t,y) = 0$ if and only if $R_{ss'}(t,y) = 0$ for all $s' \in S$, we can still apply the duality result.

4.7.1 SL ($I = [0,\infty)$ and $J = [0,\infty)$)

$\varphi = \Phi\,\mathcal{U}\,\Psi$. When deriving the equations for $\Upsilon_{ss'}(t,y)$ we implicitely assumed that $\sup I < \infty$ and $\sup J < \infty$. This is not the case for an SL formula where $I = J = \mathbb{R}_{\geq 0}$. The reward interval $J = [0,\infty)$ does not imply any restriction on the accumulated reward, thus, the joint probabilities of state and accumulated reward reduce to the transient probabilities:

$$\lim_{y \to \sup J = \infty} \Upsilon_{ss'}(t,y) = \Pi_{ss'}(t).$$

Furthermore, the transient probabilities for $t \to \sup I = \infty$ correspond to the steady-state probabilities:

$$\lim_{t \to \sup I = \infty} \Pi_{ss'}(t) = \Pi_{ss'}.$$

Consequently, the probability measure for an SL until formula is given by

$$\begin{aligned} \mathit{Prob}^{\mathcal{M}}(s, \Phi\,\mathcal{U}\,\Psi) &= \sum_{s' \in \mathsf{Sat}(\Psi)} \Pi_{ss'}^{\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\,\Psi\rangle} \\ &= \sum_{s' \in \mathsf{Sat}(\Psi)} \Pi_{ss'}^{\mathcal{M}[\neg\Phi\vee\Psi]}. \end{aligned}$$

The equality follows because $\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\,\Psi\rangle$ is actually not time- and reward inhomogeneous but behaves like $\mathcal{M}[\neg\Phi \vee \Psi]$ for all $t, y \geq 0$ because $\inf I = \inf J = 0$.

Hence, the computation of $\mathit{Prob}^{M}(s, \Phi\,\mathcal{U}\,\Psi)$ reduces to the computation of the steady-state probabilities in $\mathcal{M}[\neg\Phi \vee \Psi]$.

4.7.2 CSL ($J = [0,\infty)$)

The reward interval $J = [0,\infty)$ imposes no restriction on the accumulated reward. Thus, in order to compute the probability measure of the until formula we can replace the joint probabilities of state and accumulated reward by the transient probabilities:

$$\lim_{y \to \sup J = \infty} \Upsilon_{ss'}(\sup I, y) = \Pi_{ss'}(\sup I).$$

	$I = [0, \infty)$	$I = [t, t]$	$I = [0, t]$	$I = [t_1, t_2]$	$I = [t, \infty)$
$J = [0, \infty)$	steady-state $\mathcal{M}[\neg\Phi \vee \Psi]$ Sec. 4.7.1	transient $\mathcal{M}[\neg\Phi]$ Sec. 4.7.2 i)	transient $\mathcal{M}[\neg\Phi \vee \Psi]$ Sec. 4.7.2 ii)	double transient $\mathcal{M}[\neg\Phi]$ & $\mathcal{M}[\neg\Phi \vee \Psi]$ Sec. 4.7.2 iii)	transient & steady-state $\mathcal{M}[\neg\Phi]$ & $\mathcal{M}[\neg\Phi \vee \Psi]$ Sec. 4.7.2 iv)
$J = [y, y]$	duality $I = [y, y]$ $J = [0, \infty)$	transient $\mathcal{M}[\neg\Phi(t, y)]$ Sec. 4.7.3 i)	duality $I = [y, y]$ $J = [0, t]$	duality $I = [y, y]$ $J = (t_1, t_2]$	duality $I = [y, y]$ $J = (t, \infty)$
$J = [0, y]$	duality $I = [0, y]$ $J = [0, \infty)$	algorithms Chap. 5 $\mathcal{M}[\neg\Phi]$ Sec. 4.7.3 ii)	algorithms Chap. 5 $\mathcal{M}[\neg\Phi \vee \Psi]$ Sec. 4.7.3 v)	duality $I = [0, y]$ $J = (t_1, t_2]$	duality $I = [0, y]$ $J = (t, \infty)$
$J = (y_1, y_2]$	duality $I = [y_1, y_2]$ $J = [0, \infty)$	algorithms Chap. 5 (twice) $\mathcal{M}[\neg\Phi]$ Sec. 4.7.3 iv)	discretisation, Markovian $\mathcal{M}\langle \Phi\, \mathcal{U}^I_J\, \Psi \rangle$ Sec. 4.7.3 vi)	discretisation, Markovian $\mathcal{M}\langle \Phi\, \mathcal{U}^I_J\, \Psi \rangle$ Sec. 4.7.3 viii)	duality $I = [y_1, y_2]$ $J = (t, \infty)$
$J = (y, \infty)$	duality $I = [y, \infty)$ $J = [0, \infty)$	algorithms Chap. 5 $\mathcal{M}[\neg\Phi]$ Sec. 4.7.3 iii)	$I = [0, t]$ $J = (y, \rho_{\max} \cdot t]$ Sec. 4.7.3 vii)	$I = [t_1, t_2]$ $J = (y, \rho_{\max} \cdot t]$ Sec. 4.7.3 ix)	?

Table 4.1: Overview of the algorithms needed for model checking CSRL until formulas

Applying Theorem 3 we obtain the following formula for the probability measure of a CSL until formula:

$$\mathit{Prob}^{\mathcal{M}}(s, \Phi\,\mathcal{U}^I\Psi) = \sum_{s' \in \mathsf{Sat}(\Psi)} \Pi_{ss'}^{\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle}(\sup I_s^J).$$

We distinguish four types of intervals I:

i) $\varphi = \Phi\,\mathcal{U}^{[t,t]}\Psi$. If $t = 0$, the probability measure is 1 for Ψ-states and 0 for all other states:

$$\mathit{Prob}_s(\Phi\,\mathcal{U}^{[0,0]}\Psi) = \begin{cases} 1, & s \in \mathsf{Sat}^{\mathcal{M}}(\Psi), \\ 0, & \text{otherwise.} \end{cases}$$

For $t > 0$, only the first phase of $\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle$ is used in the computation, the second phase starts exactly at the instant of time at which the joint distribution should be evaluated. The probability of entering a Ψ-state exactly at time t is 0, therefore a Ψ-state must be reached already before t. This is only allowed if it is simultaneously a Φ-state. We rephrase the until formula as $\Phi\,\mathcal{U}_J^I(\Phi \wedge \Psi)$. The duplication of $(\neg\Phi \wedge \Psi)$-states is then not necessary because there are no longer goal states. The complete computation of the transient probabilities is performed in $\mathcal{M}[\neg\Phi]$:

$$\mathit{Prob}_s(\Phi\,\mathcal{U}^{[t,t]}\Psi) = \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Phi\wedge\Psi)} \Pi_{ss'}^{\mathcal{M}[\neg\Phi]}(t).$$

Using uniformisation (cf. Section 3.2), the transient probabilities are efficiently obtainable.

ii) $\varphi = \Phi\,\mathcal{U}^{[0,t]}\Psi$. We assume $t > 0$. The first phase of the MRM $\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle$ is actually nonexistent because $\inf I = \inf J = 0$. Thus, only the second phase of the MRM $\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle$ is used. It is not necessary to duplicate states.

$$\mathit{Prob}^{\mathcal{M}}(s, \Phi\,\mathcal{U}^{[0,t]}\Psi) \;=\; \sum_{s' \in \mathsf{Sat}(\Psi)} \Pi_{ss'}^{\mathcal{M}[\neg\Phi\vee\Psi]}(t).$$

Like in i), uniformisation is the method of choice to compute the transient probabilities.

iii) $\varphi = \Phi\,\mathcal{U}^{[t_1,t_2]}\Psi$. We have to compute transient probabilities in a time-inhomogeneous Markov chain. Uniformisation as presented in Section 3.2 deals only with homogeneous Markov chains. However, if – as is the case here – the inhomogeneity is only present at a single point in time, uniformisation can be applied twice in order to compute the transient probabilities [45]. First the probabilities of reaching all other states by time $\inf I = t_1$ are computed using $\mathbf{Q}^{(1)}$. These probabilities are multiplied by the transient probability of reaching the goal state by time $t_2 - t_1$ using $\mathbf{Q}^{(2)}$.

$$\begin{aligned}\sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \Pi^{\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle}(\sup I)_{ss'} &= \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \Pi^{\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle}(t_2)_{ss'} \\ &= \sum_{z \in S \cup \{\bar{z} | z \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)\}} \Pi_{sz}^{\mathcal{M}^{(1)}}(t_1) \cdot \Pi_{zs'}^{\mathcal{M}^{(2)}}(t_2 - t_1).\end{aligned}$$

We can avoid the duplication of $(\neg\Phi \wedge \Psi)$-states by observing that $\Pi^{\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\Psi\rangle}_{\bar{z}s'}(t) = 0$ for $z \neq s'$ (all duplicated states reachable in the first phase are absorbing). We can exclude these states explicitely from the summation and use the two homogeneous MRMs $\mathcal{M}[\neg\Phi]$ and $\mathcal{M}[\neg\Phi \vee \Psi]$:

$$\sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \Pi^{\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\Psi\rangle}(\sup I)_{ss'} = \sum_{z \in \mathsf{Sat}^{\mathcal{M}}(\Phi)} \Pi^{\mathcal{M}[\neg\Phi]}_{sz}(t_1) \cdot \Pi^{\mathcal{M}[\neg\Phi\vee\Psi]}_{zs'}(t_2 - t_1).$$

One can also come to this formula applying renewal arguments [7].

iv) $\varphi = \Phi\,\mathcal{U}^{[t,\infty)}\Psi$. Using similar arguments as for SL until formulas and applying the idea of a double computation like in case iii), the steady-state probabilities in the inhomogeneous MRM can be a computed as follows.

$$\Pi^{\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\Psi\rangle}_{ss'} = \sum_{z \in \mathsf{Sat}^{\mathcal{M}}(\Phi)} \Pi^{\mathcal{M}[\neg\Phi]}_{sz}(t) \cdot \Pi^{\mathcal{M}[\neg\Phi\vee\Psi]}_{zs'}.$$

First the probabilities of reaching any Φ-state by time $\inf I = t$ are computed in $\mathcal{M}[\neg\Phi]$. These probabilities are multiplied by the steady-state probability of reaching the goal state in $\mathcal{M}[\neg\Phi \vee \Psi]$.

4.7.3 CSRL

i) $\varphi = \Phi\,\mathcal{U}^{[t,t]}_{[y,y]}\Psi$.

At a given time t, the reward can only equal y, if $y = \rho_{s^*} \cdot t$ for at least one state s^* and the MRM has been in states with reward rate ρ^*_s all the time. We can therefore code the reward restriction into the structural formula. Introduce a new atomic proposition $\Phi(t, y)$ which is assigned to the states that satisfy Φ and have reward rate ρ_{s^*}. In the until formula, the state formula Φ is replaced by $\Phi(t, r)$ which makes the reward restriction dispensable. Thus, only a CSL formula has to be checked (cf. CSL i)):

$$\begin{aligned} \mathit{Prob}^{\mathcal{M}}_s(\Phi\,\mathcal{U}^{[t,t]}_{[y,y]}\Psi) &= \mathit{Prob}^{\mathcal{M}}_s(\Phi(t,y)\,\mathcal{U}^{[t,t]}\Psi) \\ &= \sum_{s' \in \mathsf{Sat}(\Psi)} \Pi^{\mathcal{M}[\neg\Phi(t,y)]}_{ss'}(t). \end{aligned}$$

ii) $\varphi = \Phi\,\mathcal{U}^{[t,t]}_{[0,y]}\Psi$. Following the same line of argumentation as in CSL i), only the first phase of the MRM $\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\Psi\rangle$ is used. The joint distribution must consequently only be evaluated in a homogeneous MRM. Different algorithms for the computation of the joint probabilities $\Upsilon^{\mathcal{M}[\neg\Phi]}_{ss'}(t, y)$ are presented in Chapter 5.

iii) $\varphi = \Phi\,\mathcal{U}^{[t,t]}_{(y,\infty)}\Psi$. This is almost the same case as ii). The only difference is that the accumulated reward must be larger than y instead of at most y. Thus, the joint probability of being in a Ψ-state while having accumulated a reward greater than y is employed.

$$Prob(s, \Phi\,\mathcal{U}^{[t,t]}_{(y,\infty)}\Psi) = \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \overline{\Upsilon}^{\mathcal{M}[\neg\Phi]}_{ss'}(t, y)$$

Some of the algorithms in Chapter 5 are directly suitable for the computation of $\overline{\Upsilon}^{\mathcal{M}[\neg\Phi]}_{ss'}(t, y)$, the other algorithms can be applied by observing that

$$\overline{\mathbf{\Upsilon}}^{\mathcal{M}}_{ss'} = \mathbf{\Pi}^{\mathcal{M}}_{ss'} - \mathbf{\Upsilon}^{\mathcal{M}}_{ss'}.$$

iv) $\varphi = \Phi\,\mathcal{U}^{[t,t]}_{(y_1,y_2]}\Psi$. Observe that

$$\begin{aligned} &\mathrm{Pr}_{\mathcal{M}\langle\Phi\mathcal{U}^I_J\Psi\rangle}\left\{X_t = s', Y_t \in (y_1, y_2] \mid X_0 = s\right\} \\ = \mathrm{Pr}_{\mathcal{M}\langle\Phi\mathcal{U}^I_J\Psi\rangle}&\left\{X_t = s', Y_t \le y_2 \mid X_0 = s\right\} - \mathrm{Pr}_{\mathcal{M}\langle\Phi\mathcal{U}^I_J\Psi\rangle}\left\{X_t = s', Y_t \le y_1 \mid X_0 = s\right\} \\ &= \Upsilon^{\mathcal{M}[\neg\Phi]}_{ss'}(t, y_1) - \Upsilon^{\mathcal{M}[\neg\Phi]}_{ss'}(t, y_2). \end{aligned}$$

The probability measure is then

$$Prob_s(\Phi\,\mathcal{U}^{[t,t]}_{(y_1,y_2]}\Psi) = \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \Upsilon^{\mathcal{M}[\neg\Phi]}_{ss'}(t, y_1) - \Upsilon^{\mathcal{M}[\neg\Phi]}_{ss'}(t, y_2).$$

v) $\varphi = \Phi\,\mathcal{U}^{[0,t]}_{[0,y]}\Psi$. For this CSRL until formula, only the second phase of the MRM $\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\Psi\rangle$ is of importance (cf. CSL ii)). The joint probability of being in a Ψ-state and having accumulated not more than y reward at time t is then computed by one of the algorithms in Chapter 5.

vi) $\Phi\,\mathcal{U}^{[0,t]}_{(y_1,y_2]}\Psi$. For this type of CSRL until formulas, the MRM $\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\Psi\rangle$ is actually inhomogeneous, but not with respect to time but only with respect to reward. Suitable algorithms for the computation of $\mathbf{\Upsilon}^{\mathcal{M}\langle\Phi\mathcal{U}^I_J\Psi\rangle}(t, y_2)$ are the discretisation algorithm (Section 5.4) and the Markovian approximation (Section 5.5).

vii) $\Phi\,\mathcal{U}^{[0,t]}_{(y,\infty)}\Psi$. Let $\rho_{\max} = \max_{s\in S}\{\rho_s\}$. The maximum reward that can be accumulated in the interval $[0, t]$ is $\rho_{\max}\cdot t$. It is consequently not necessary to consider the complete reward interval (y, ∞) but it can be replaced by $(y, \rho_{\max} \cdot t]$:

$$Prob^{\mathcal{M}}(s, \Phi\,\mathcal{U}^{[0,t]}_{(y,\infty)}\Psi) = Prob^{\mathcal{M}}(s, \Phi\,\mathcal{U}^{[0,t]}_{(y,\rho_{\max}\cdot t]}\Psi).$$

The calculation then proceeds as in vi).

viii) $\Phi\,\mathcal{U}^{[t_1,t_2]}_{(y_1,y_2]}\Psi$. This formula leads to an MRM $\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\Psi\rangle$ that is both time and reward inhomogeneous. Algorithms for calculating $\mathbf{\Upsilon}^{\mathcal{M}\langle\Phi\mathcal{U}^I_J\Psi\rangle}(t_2, y_2)$ are the discretisation algorithm (Section 5.4) and the Markovian approximation (Section 5.5).

ix) $\Phi\,\mathcal{U}^{[t_1,t_2]}_{(y,\infty)}\Psi$. The accumulated reward by time t_2 is at most $\rho_{\max} \cdot t_2$. We can therefore restrict the computation to the reward interval $(y, \rho_{\max} \cdot t_2]$ and proceed like in viii).

$$Prob^{\mathcal{M}}(s, \Phi\,\mathcal{U}^{[t_1,t_2]}_{(y,\infty)}\Psi) = Prob^{\mathcal{M}}(s, \Phi\,\mathcal{U}^{[t_1,t_2]}_{(y,\rho_{\max}\cdot t_2]}\Psi).$$

4.8 Summary

In this chapter we recapitulated the logic CSRL [6] which allows us to express properties of stochastic models involving time and reward. The difficulties in model checking formulas of this logic are in the computation of the probability measure of paths satisfying a given until formula. We showed how to construct a time- and reward-inhomogeneous MRM that supports this measure. The measure can be determined by calculating the joint distribution of state and accumulated reward in this inhomogeneous MRM. For many types of time and reward intervals, computations in a homogeneous MRM are sufficient, or only the transient or steady-state probabilities are of interest (without regarding the rewards). Numerical algorithms for the joint distribution are the topic of the next chapter.

Chapter 5

Computation of the joint distribution of state and accumulated reward

In Chapter 3 we derived a set of PDEs (3.10) that describes

$$\Upsilon_{ss'}(t, y) = \Pr\{X_t = s', Y_t \leq y \mid X_0 = s\}, \tag{5.1}$$

i.e., the joint distribution of state and accumulated reward (performability). A "solution" for this set of PDEs is a set of integral equations that, however, is not really easier to solve numerically than the PDEs. In Chapter 4 we showed that we need this joint distribution (5.1) for model checking the until operator of the logic CSRL. Many algorithms for the computation of the performability [73, 74]

$$\Pr\{Y_t \leq y\}$$

can be found in the literature; for an overview see [86, 104]. Most of them can be adapted for the joint distribution (5.1) of state and accumulated reward. However, only a few of these algorithms are generical enough to be applicable in the CSRL model checking context. Early work was restricted to *acyclic* MRMs [17, 37, 41, 43]. Some algorithms for possibly cyclic MRMs are based on Laplace transforms and are therefore restricted to MRMs with very small state spaces [64, 98]. Others consider only availability models with two different reward classes [31, 91, 94]. In this chapter we present five algorithms for the unrestricted actual numerical computation of $\boldsymbol{\Upsilon}(t, y)$.

Section 5.1 shows how the integral equations are numerically solved using Picard's method as proposed by Pattipati et al. [81, 82, 83]. In Section 5.2 we present an analytical uniformisation-based solution algorithm by Sericola [95]. In Section 5.3 we adapt the algorithm of Qureshi & Sanders [86] (based on work of de Souza e Silva and Gail [32]) for our purposes. It has already been used for CSRL model checking in [28]. The discretisation approach of Tijms & Veldman is the topic of Section 5.4. The last algorithm is the Markovian approximation by Haverkort et al., first published in [50]. The uniformisation algorithm, discretisation and Markovian approximation were already exploited for the joint distribution in homogeneous MRMs in [49]. Finally, we compare the results and the performance of the algorithms in Section 5.6.

5.1 Successive approximation (Picard's method)

The integral equation

$$\Upsilon_{ss'}(t,y) = e^{Q_{ss}t}\Upsilon_{ss'}(0, y-\rho_s t) \quad + \quad \int_0^t \sum_{z\neq s} e^{Q_{ss}x} Q_{sz} \Upsilon_{zs'}(t-x, y-\rho_s x)dx$$

is the analytical solution of PDE (3.9) for the joint distribution of state and reward $\Upsilon_{ss'}(t,y)$ in a homogeneous MRM. Since $\mathbf{\Upsilon}(t,y)$ appears on both sides of the equation, its actual values have to be determined by a fixed point computation. One numerical algorithm (used by Pattipati et al. [82] in this context) for fixed point computations is "Picard's method": it generates a sequence of approximations $\mathbf{\Upsilon}^{(n)}(t,y)$ that converges to the correct solution:

$$\lim_{n\to\infty} \mathbf{\Upsilon}^{(n)}(t,y) = \mathbf{\Upsilon}(t,y).$$

The first approximation is given by

$$\Upsilon^{(0)}_{ss'}(t,y) = e^{Q_{ss}t}\Upsilon_{ss'}(0, y-\rho_s t).$$

Note that $\Upsilon_{ss'}(0, y-\rho_s t)$ is known as initial value for the PDE. The subsequent approximations are computed using the integral equation:

$$\Upsilon^{(n+1)}_{ss'}(t,y) = e^{Q_{ss}t}\Upsilon^{(n)}_{ss'}(0, y-\rho_s t) \quad + \quad \int_0^t \sum_{z\neq s} e^{Q_{ss}x} Q_{sz} \Upsilon^{(n)}_{zs'}(t-x, y-\rho_s x)dx.$$

The iteration is terminated after a predefined number of steps N or if the change between two subsequent iterations drops below a given accuracy threshold.

Each iteration step involves the evaluation of $|S|-1$ integrals over the previous approximation of the joint distribution. The integration can only be performed numerically. Different integration schemes are possible, for the sake of simplicity we restrict the description to the simple trapezoidal rule. The integration interval $[0,t]$ is divided into subintervals of size Δt. The integrals are then approximated by

$$\begin{aligned}\int_0^t e^{Q_{ss}x} Q_{sz} \Upsilon^{(n)}_{zs'}(t-x, y-\rho_s x)dx = Q_{sz}\cdot &\left(\frac{1}{2}\Upsilon^{(n)}_{zs'}(t,y)\right.\\ &+ \sum_{i=1}^{\frac{t}{\Delta t}-1} e^{Q_{ss}i\Delta t}\Upsilon^{(n)}_{zs'}(t-i\Delta t, y-\rho_s i\Delta t)\\ &\left.+\frac{1}{2}e^{Q_{ss}t}\Upsilon^{(n)}_{zs'}(0, y-\rho_s t)\right).\end{aligned}$$

The integration scheme shows that it does not suffice just to compute $\Upsilon_{ss'}(t,y)$ in each iteration step, we need sample points $\Upsilon_{ss'}(i\Delta t, j\Delta t)$, for $i = 0, \cdots, \frac{t}{\Delta t}$, and $j = 0, \cdots, \frac{y}{\Delta t}$. Actually we might also need sample points for negative values of j, but then the distribution is zero anyway and we do not need to compute/store these values.

The multitude of numerical integrations plus the approximate nature of the outer iteration make it impossible to indicate an estimation of the resulting numerical error for this method.

Space and time complexity

In each iteration step we have to consider $|S| \cdot (\frac{t}{\Delta t}+1) \cdot (\frac{y}{\Delta t}+1)$ sampling points. Two successive versions of the sampling grid have to be stored, so the space requirement is two times $\mathcal{O}(|S| \cdot t \cdot y \cdot \Delta t^{-2})$. The smaller Δt is, that is, the finer the grid, the more space is needed.

Each sampling point is the result of a sum of $|S|-1$ numerically computed integrals and one additional term; on the average we need $\frac{1}{2}(\frac{t}{\Delta t}+1)$ sampling points of the previous iteration for the evaluation of one of the integrals. Thus a new sampling point is computed with time complexity $\mathcal{O}(|S| \cdot t \cdot \Delta t^{-1})$. For N iterations, the overall time complexity is then $\mathcal{O}(N \cdot |S|^2 \cdot t^2 \cdot y \cdot \Delta t^{-3})$. The algorithm always computes a complete column of the matrix $\mathbf{\Upsilon}(t,y)$. If we need all entries of the matrix, the time complexity becomes cubic in the number of states.

5.2 Analytical solution by uniformisation

Sericola [95] derives a uniformisation-based solution of the system of partial differential equations that describes the complementary joint distribution of state and accumulated reward $\overline{\Upsilon}_{ss'}(t,y) = \Pr\{X_t = s', Y_t > r \mid X_0 = s\}$ for an MRM. His approach is suitable for *homogeneous* MRMs. When using uniformisation for the computation of the transient distribution (cf. Section 3.2) we condition on the number of steps in the uniformised Markov chain. Sericola does not only condition $\overline{\Upsilon}_{ss'}(t,y)$ on the number of steps n, but also on the number k of transitions that happen before a certain threshold y_h (see below) and the reward bound y, as follows:

$$\overline{\Upsilon}_{ss'}(t,y) \;=\; \sum_{n=0}^{\infty} PP(\lambda t, n) \sum_{k=0}^{n} \binom{n}{k} y_h^k (1-y_h)^{n-k} C_{ss'}^{(h)}(n,k), \qquad (5.2)$$

where $y_h = \frac{y - r_{h-1}t}{r_h t - r_{h-1}t}$, for $y \in [r_{h-1}t, r_h t)$, and $\binom{n}{k} y_h^k (1-y_h)^{n-k}$ is the probability that exactly k of the n transitions have happened by time $\frac{y-r_{h-1}t}{r_h}$.

The value of $C_{ss'}^{(h)}(n,k)$ is then the complementary distribution $\overline{\Upsilon}_{ss'}(t,y)$ conditioned on n and k. We describe the recursive computation of $C_{ss'}^{(h)}(n,k)$ in the following.

Let $r_0, \cdots, r_K$ be the $K+1$ different values that appear in the reward vector ρ of the MRM, ordered in such a way that $r_0 < r_1 < \cdots < r_{K-1} < r_K$. Without loss of generality, we assume that $r_0 = 0$. If $r_0 > 0$, we can decrease all reward rates by r_0 and compute $\overline{\Upsilon}_{ss'}(t, y - r_0 t)$ instead. The initial values for the recursion are given by

$$C_{ss'}^{(1)}(n,0) = (\mathbf{U}^n)_{ss'} \text{ , for } n \geq 0 \text{ and } \rho_{s'} \geq r_1 \qquad (5.3)$$

and

$$C_{ss'}^{(K)}(n,n) = 0 \text{ , for } n \geq 0 \text{ and } \rho_{s'} < r_K. \qquad (5.4)$$

For the recursion, there are two pairs of equations:

$$\begin{aligned} C^{(i)}_{ss'}(n,0) &= C^{(i-1)}_{ss'}(n,n) \text{ , for } 1 < i \leq K \text{ and } \rho_{s'} \geq r_i, & (5.5)\\ C^{(i)}_{ss'}(n,n) &= C^{(i+1)}_{ss'}(n,0) \text{ , for } 0 \leq i < K \text{ and } \rho_{s'} < r_i. & (5.6)\end{aligned}$$

and

$$\begin{aligned} C^{(i)}_{ss'}(n,k) &= \frac{\rho_{s'} - r_i}{\rho_{s'} - r_{i-1}} \cdot C^{(i)}_{ss'}(n,k-1) + \frac{r_i - r_{i-1}}{\rho_{s'} - r_{i-1}} \cdot \sum_{z \in S} C^{(i)}_{sz}(n-1,k-1) U_{zs'},\\ &\quad \text{for } 1 \leq k \leq n, 1 < i \leq K \text{ and } \rho_{s'} \geq r_i, & (5.7)\\ C^{(i)}_{ss'}(n,k) &= \frac{r_{i-1} - \rho_{s'}}{r_i - \rho_{s'}} \cdot C^{(i)}_{ss'}(n,k+1) + \frac{r_i - r_{i-1}}{r_i - \rho_{s'}} \cdot \sum_{z \in S} C^{(i)}_{sz}(n-1,k) U_{zs'},\\ &\quad \text{for } 0 \leq k < n, 1 < i \leq K \text{ and } \rho_{s'} < r_i. & (5.8)\end{aligned}$$

All coefficients in the recurrence equations are real numbers between 0 and 1; the coefficients in a single equation always add up to 1. Consequently, the recursion only deals with sums and products of real numbers between 0 and 1 and is therefore numerically stable.

Using uniformisation, it is not possible to evaluate the complete infinite sum (5.2). It has to be cut at some $N \in \mathbb{N}$. The error induced by this truncation is bounded by $\varepsilon = 1 - \sum_{n=0}^{N} I\!P(\lambda t, n)$, exactly as it was the case for the computation of the transient distribution.

Example 25. Consider the CSRL path formula $\texttt{empty}\,\mathcal{U}^{[0,0.5]}_{[0,5]}(\neg\texttt{intact})$ and the example MRM (Section 2.9). For the computation of he probability measure for the path formula we have to compute the joint probability in the transformed MRM $\mathcal{M}[(\neg\texttt{empty}) \vee (\neg\texttt{intact})]$ where states 2 and 3 are made absorbing. This MRM has $K+1 = 3$ distinct rewards. They are ordered such that

$$\begin{aligned} r_0 &= \rho_2 = \rho_3 = 0\\ < r_1 &= \rho_1 = 20\\ < r_2 &= \rho_0 = 50\end{aligned}$$

form the three reward classes. The states are reordered accordingly. For the formula $\texttt{empty}\,\mathcal{U}^{[0,0.5]}_{[0,5]}(\neg\texttt{intact})$ we have that $y \in [r_0 \cdot 0.5, r_1 \cdot 0.5)$, and $y_h = \frac{5 - 0 \cdot 0.5}{20 \cdot 0.5 - 0 \cdot 0.5} = \frac{1}{2}$. Figure 5.1 shows graphically how the recursion for the $C^{(i)}_{ss'}(n,k)$ proceeds for $n = 0, \ldots, 4$, and a fixed starting state s. The superscript i ranges from 1 to $K = 2$. Each cube corresponds to one $C^{(i)}_{ss'}(n,k)$-value. For each i and n there are $n \cdot |S|$ values, hence, the graphical representation resembles a staircase. The block of $C^{(i)}_{ss'}(n,k)$ is depicted twice (one above the other) for each i, from different perspectives, in order to better illustrate the recursion.

The grey cubes are initialised using (5.3) (for $i = 1$) and (5.4) (for $i = K = 2$). Arrows *between* the blocks for different i show the relations given by (5.5) and (5.6). The arrows have to be duplicated for those cubes in a block that are not shaded black. Finally, arrows *within* a block show how the recursions given in (5.7) and (5.8) proceed. Any arrow with a tick at its end (↓ or ↳) indicates that all cubes behind are also involved in the computation, because we have to sum over all states. Again, the arrows have to be duplicated for the non-black cubes. ◊

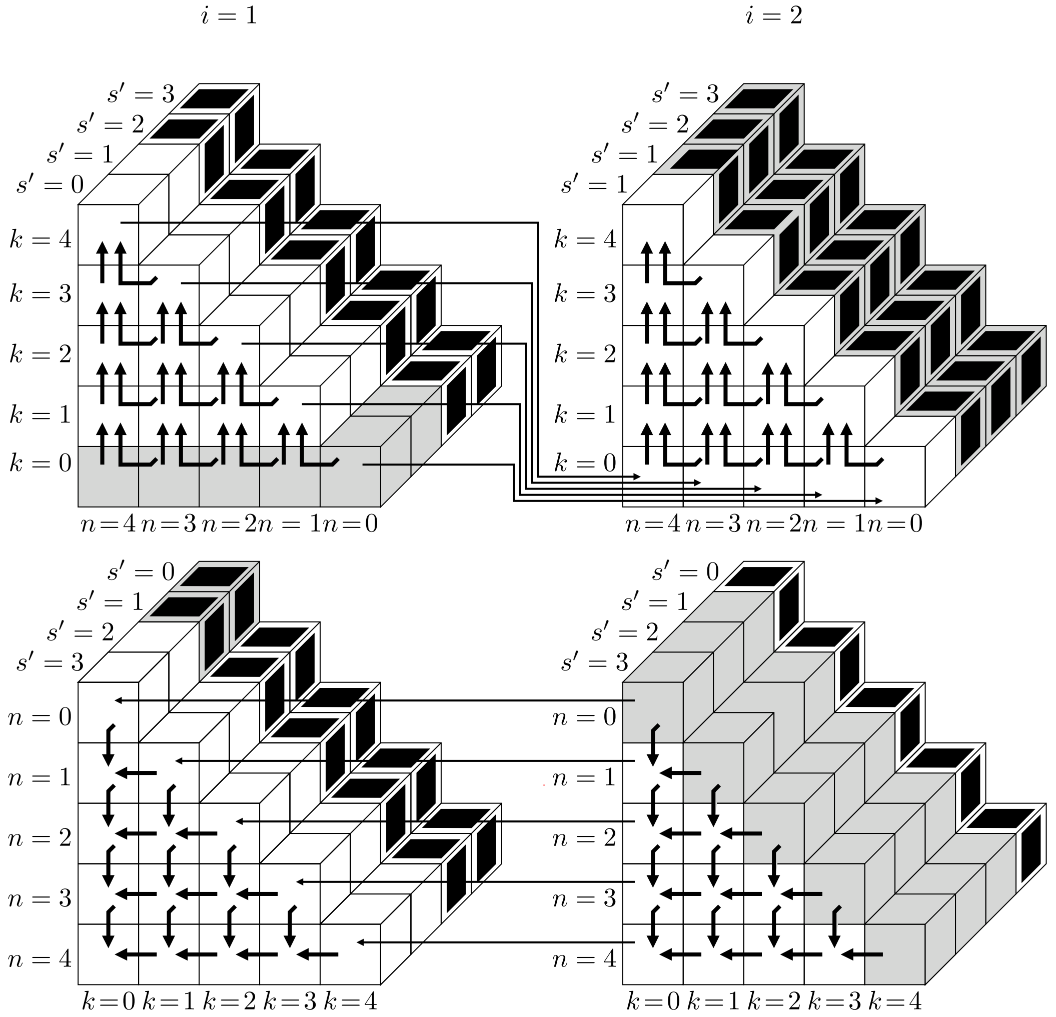


Figure 5.1: Graphical representation of the recursion for $C_{ss'}^{(i)}(n,k)$ for the computation of $\Upsilon_{ss'}(0.5,5)$ for the illustrating example

Space and time complexity

The matrices $\mathbf{C}^{(i)}(n,k)$ have to be stored for two subsequent iterations, that is, for $n-1$ and n. The superscript i ranges from 1 to K, k ranges from 0 to n. Hence, in each iteration we have $(n+1)\cdot K$ matrices of size $|S|^2$. The total number of iterations is in $\mathcal{O}(\lambda t)$ and so the space complexity is $\mathcal{O}(\lambda t \cdot K \cdot |S|^2)$. Note that in the worst case the number of reward classes is $K+1=|S|$.

In each iteration the matrices $\mathbf{C}^{(i)}(n,k)$ are computed via (5.7) and (5.8), and for each entry we need $|S|+1$ multiplications. This gives us an overall time complexity of $\mathcal{O}(\lambda t \cdot |S|^3)$ for computing the complete matrix $\boldsymbol{\Upsilon}(t,y)$.

5.3 Path exploration

In this section we present a second uniformisation-based method, also only suitable for homogeneous MRMs. It is based on the work of Qureshi and Sanders [85, 86] and has already been used in the CSRL model checking context in [62, 26]. The joint distribution $\boldsymbol{\Upsilon}(t,y)$ is not conditioned on the number of steps taken until time t but on the precise path taken up to this time.

Recall from Section 3.2 that $\sigma = (s_0, \cdots, s_n) \in uPath$ is a so-called uniformised path of length n. Its probability is given by $P(\sigma) = U_{s_0 s_1} \cdot \cdots \cdot U_{s_{n-1} s_n}$. In Section 3.2 one way to derive the transient probabilities was to condition on all possible uniformised paths. The joint probability of state and accumulated reward $\Upsilon_{ss'}(t,y)$ can also be computed conditioned on paths:

$$\Upsilon_{ss'}(t,y) = \sum_{n=0}^{\infty} PP(\lambda t, n) \cdot \sum_{\substack{\sigma \in uPath \\ |\sigma| = n}} P(\sigma \mid Z_0 = s) \cdot \Pr\{Z_n = s', Y_t \leq y \mid \sigma\} \tag{5.9}$$

$$= \sum_{n=0}^{\infty} PP(\lambda t, n) \cdot \sum_{\substack{\sigma \in uPath \\ |\sigma| = n \\ first(\sigma) = s \\ last(\sigma) = s'}} P(\sigma) \cdot \Pr\{Y_t \leq y \mid \sigma\}. \tag{5.10}$$

Following this expression, we consider the uniformised paths that start in s and end in s', calculate the reward distribution conditioned on each of these paths and compute the weighted sum of all conditioned probabilities. Two questions arise with this approach:

i) How do we calculate the conditional reward distribution?

ii) Is it possible to consider all relevant uniformised paths?

The two issues are addressed in the following two sections.

5.3.1 The conditional reward distribution

The state space of the MRM $\mathcal{M}$ can be divided into $K+1$ distinct reward classes. Without loss of generality, the reward classes are ordered such that

$$r_0 > r_2 > \ldots > r_K \geq 0.$$

Note that in contrast to the approach by Sericola presented in the previous section, the reward rates are ordered in descending order. For the computation of $\Pr\{Y_t \leq y \mid \sigma\}$ it is not necessary to consider the complete information contained in the path σ but one only has to know how many epochs of the uniformised path have been spent in each of the reward classes. A vector $\mathbf{k} = (k_1, \ldots, k_K)$ recording these visit counts is called a *colouring.* The value k_i indicates the number of epochs the MRM has spent in states of reward class i. The term colouring stems from the idea of assigning the same colour to states with identical reward rate [34]. We denote the colouring deduced directly from a uniformised path σ as $\mathbf{k}(\sigma)$ and have

$$\Pr\{Y_t \leq y \mid \sigma\} = \Pr\{Y_t \leq y \mid \mathbf{k}(\sigma)\}.$$

It has been shown [90, 33] that

$$\Pr\{Y_t \leq y \mid \mathbf{k}\} = \Pr\left\{\sum_{i=1}^{K-1}(r_i - r_{i+1}) \cdot V_{(k_1+\ldots+k_i)} \leq y'\right\}$$

where $V_{(1)}, \ldots, V_{(\sum_i k_i)}$ are the order statistics from a set of independent and identically distributed random variables, uniform on $[0,1]$, and $y' = \frac{y}{t} - r_K$.

Thus, the computation of the distribution of Y_t given a colouring $\mathbf{k}$ boils down to the computation of the distribution of a linear combination of uniform order statistics.

We are aware of 3 methods for the calculation of this type of distribution. The approaches of Weisberg [107] and Matsunawa [71] use involved computations and tend to be numerically unstable. The recently proposed method of Diniz et al. [36] is based on a very simple recursion scheme and is numerically stable. It is therefore the only one presented here. The probability under consideration is abbreviated as

$$\Upsilon(y' \mid \mathbf{k}) = \Pr\{Y_t \leq y \mid \mathbf{k}\} = \Pr\left\{\sum_{i=1}^{K-1}(r_i - r_{i+1}) \cdot V_{(k_1+\ldots+k_i)} \leq y'\right\}.$$

Let $G = \max_i\{r_i > \frac{y}{t}\}$ be the highest index of a reward class with a rate greater than $\frac{y}{t}$. The colouring $\mathbf{k} = (k_1, \ldots, k_K)$ is divided into two sub-vectors $\mathbf{k}_G = (k_1, \ldots, k_G)$ and $\mathbf{k}_L = (k_{G+1}, \ldots, k_K)$.

Initial values for the recursion are defined if either all entries of $\mathbf{k}_G$ are zero ($\|\mathbf{k}_G\| = 0$) *or* all entries of $\mathbf{k}_L$ are zero ($\|\mathbf{k}_L\| = 0$), but not both. If $\|\mathbf{k}_G\| = 0$, the Markov chain has only visited states with a reward rate at most $\frac{y}{t}$, so the probability for having accumulated at most r reward at time t is 1. By a similar argument, the probability is 0 if the Markov chain has only visited states with a reward rate greater than $\frac{y}{t}$. Thus, we have

$$\Upsilon(y' \mid \mathbf{k}) = \begin{cases} 1, & \|\mathbf{k}_G\| = 0 \text{ and } \|\mathbf{k}_L\| > 0, \\ 0, & \|\mathbf{k}_G\| > 0 \text{ and } \|\mathbf{k}_L\| = 0. \end{cases}$$

In case $\|\mathbf{k}_G\| > 0$ and $\|\mathbf{k}_L\| > 0$, we can choose any $i \in \{1, \ldots, G\}$ with $\mathbf{k}_i > 0$ and $j \in \{G+1, \ldots, K\}$ with $\mathbf{k}_j > 0$ and find the recursion

$$\Upsilon(y' \mid \mathbf{k}) = \left(\frac{r_i - y'}{r_i - r_j}\right) \cdot \Upsilon(y' \mid \mathbf{k} - \mathbf{1}_{[j]}) + \left(\frac{y' - r_j}{r_i - r_j}\right) \cdot \Upsilon(y' \mid \mathbf{k} - \mathbf{1}_{[i]}) \tag{5.11}$$

where $\mathbf{1}_{[i]}$ is a vector of length K with value 1 at position i and zeros at all other positions. An important property of (5.11) is that the coefficients satisfy

$$0 \leq \frac{r_i - y'}{r_i - r_j} \leq 1, \quad 0 \leq \frac{y' - r_j}{r_i - r_j} \leq 1, \quad \frac{r_i - y'}{r_i - r_j} + \frac{y' - r_j}{r_i - r_j} = 1,$$

because $r_i > y \geq r_j$. Hence, the recursion only deals with sums and products of real numbers between 0 and 1 which makes it numerically stable.

The recursion (5.11) is also suitable to compute the complementary distribution $\Pr\{Y_t > r \mid \mathbf{k}\}$ by simply interchanging the initial conditions.

Example 26. Consider the transformed MRM, $\mathcal{M}[(\neg\texttt{empty}) \vee (\neg\texttt{intact})]$ which has $K+1 = 3$ reward classes. We have $r_1 = 50$ (state 0), $r_2 = 20$ (state 1), $r_3 = 0$ (states 2 and 3). For the uniformised path

$$\sigma = 0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow 3$$

the colouring is $\mathbf{k}(\sigma) = (3, 2, 1)$. Using the algorithm of Diniz et al. for the computation yields

$$\Pr\{Y_{0.25} \leq 10 \mid \mathbf{k}(\sigma)\} = \Upsilon\left(\frac{10}{0.25} - 0 \mid \mathbf{k}(\sigma)\right) = \Upsilon(40 \mid (3,2,1)).$$

Table 5.1 shows how the computation proceeds. Each cell contains a colouring $\mathbf{k}$ split into entries for rewards greater and lower or equal to 40, and the corresponding $\Upsilon(40 \mid \mathbf{k})$. The values of the first row and column are fixed by the initial conditions. The arrows indicate which cells are needed to compute the next value recursively. Following an arrow downwards, an entry of a reward class with reward rate greater than 40 is added to the colouring. Following an arrow to the right, an entry of a reward class with a reward rate lower or equal to 40 is added.

The rightmost cell in the last row contains the colouring $\mathbf{k}(\sigma) = ((3)(2,1))$ and its corresponding $\Upsilon(40 \mid \mathbf{k}(\sigma))$. Note that this is not the only possibility of constructing the recursion for $\Upsilon(40 \mid \mathbf{k}(\sigma))$. It resulted by always increasing the smallest possible class index. Other schemes apply as well.

Complexity and implementation issues

Following the graphical representation of the recursion of Example 26, one can easily see that is suffices to store two rows of the recursion grid. Each row has $\|\mathbf{k}_L\|+1$ entries, thus, the space requirement is $\mathcal{O}(\|\mathbf{k}_L\|)$. The computation of each inner entry in the grid requires two multiplications. Since there are $\|\mathbf{k}_G\|+1$ rows we have an overall time complexity of $\mathcal{O}(\|\mathbf{k}_G\| \cdot \|\mathbf{k}_L\|)$. Note that $\|\mathbf{k}_G\|, \|\mathbf{k}_L\| \leq |\sigma|+1 \leq N+1$. A coarser term for the time complexity is therefore $\mathcal{O}(N^2)$.

If $\|\mathbf{k}_G\| < \|\mathbf{k}_L\|$ one can also proceed columnwise in order to reduce the space requirement. The coefficients in the recursion do not depend on the colouring, so we precompute them if we want to compute the reward distribution for several colourings.

$$\begin{tikzcd}
 & ((0)(1,0)) \arrow[r] \arrow[d] & ((0)(2,0)) \arrow[r] \arrow[d] & ((0)(2,1)) \arrow[d] \\
((1)(0,0)) \arrow[r] & ((1)(1,0)) \arrow[r] \arrow[d] & ((1)(2,0)) \arrow[r] \arrow[d] & ((1)(2,1)) \arrow[d] \\
((2)(0,0)) \arrow[r] & ((2)(1,0)) \arrow[r] \arrow[d] & ((2)(2,0)) \arrow[r] \arrow[d] & ((2)(2,1)) \arrow[d] \\
((3)(0,0)) \arrow[r] & ((3)(1,0)) \arrow[r] & ((3)(2,0)) \arrow[r] & ((3)(2,1))
\end{tikzcd}$$

$\Diamond$

Table 5.1: The Diniz recursion for the computation of $\Upsilon(40|(3,2,1))$

5.3.2 Which paths to explore?

We now concentrate on the answer to the second question: can we consider all relevant uniformised paths?

The set of all uniformised paths starting in s and ending in s' in (5.9) is partitioned according to the number of steps (the length) within a path. For a fixed starting state s there is exactly one uniformised path with 0 steps, namely s itself. The starting state has up to $|S|$ successor states, so there are $\mathcal{O}(|S|)$ uniformised paths of length 1. Repeating this argument, there are $\mathcal{O}(|S|^n)$ uniformised paths of length n. The total number of paths is of course infinite, but even if we take into account only paths up to a given length N, the number grows exponentially with N. Hence, the consideration of all paths in (5.9) is infeasible.

The probability $P(\sigma)$ of a uniformised path is used in (5.9) as a weight for the conditional reward distribution. Qureshi and Sanders [86] introduce a threshold $w \in (0,1)$ for $P(\sigma)$: only if $P(\sigma) > w$, the path σ is included in the summation. Additionally, a maximum length N is fixed for the uniformised paths. Define the set of uniformised paths of length n that are actually considered for the computation as

$$\mathit{Considered}(s,s',w,n) := \{\sigma \in \mathit{uPath} \mid \mathit{first}(\sigma) = s, \mathit{last}(\sigma) = s', P(\sigma) > w \text{ and } |\sigma| = n\}$$

This leads to the following approximation for the reward distribution:

$$\Upsilon_{ss'}(t,y) \approx \sum_{n=0}^{N} P\!\!P(\lambda t, n) \cdot \sum_{\sigma \in \mathit{Considered}(s,s',w,n)} P(\sigma) \cdot \Pr\{Y_t \leq y \mid \mathbf{k}(\sigma)\}. \tag{5.12}$$

For the calculation of (5.12), all paths contained in one of the sets $\mathit{Considered}(s,s',w,n)$ for $n = 0,\ldots,N$, have to be generated one by one. Algorithm 1 shows a simple way to perform this computation. The set `Paths` contains the uniformised paths that have to be explored further. As long as there are such paths, one is chosen and we check whether it belongs to the set of relevant paths $\mathit{Considered}(s,s',w,n)$. If this is the case, the accumulated reward weighted by the probability of the path is added to the probability of interest (Prob). Even if the path itself is not to be considered, it is prolonged by one state as long as its probability is above the threshold and it is not too long. The resulting paths are included in the set of paths.

Algorithm 1 Compute $\texttt{Prob} = \Upsilon_{ss'}(t, y)$ via path exploration

```
Prob $= 0$
Error $= 0$
Paths $= \{s\}$
while Paths $\neq \emptyset$ do
  choose $\sigma \in$ Paths
  Paths $:=$ Paths$\setminus\{\sigma\}$
  if $(P(\sigma) > w)$ and $(|\sigma| \leq N)$ then
    if $(\mathit{last}(\sigma) = s')$ then
      Prob += $\mathit{PP}(\lambda t, |\sigma|) \cdot P(\sigma) \cdot \Pr\{Y_t \leq y \mid \mathbf{k}(\sigma)\}$
    end if
    for all $(s' \in S)$ do
      insert $(\sigma \circ s')$ into Paths
    end for
  else
    Error += $\left(1 - \sum_{n=0}^{|\sigma|-1} \mathit{PP}(\lambda t, n)\right) \cdot P(\sigma)$
  end if
end while
```

Error bound

Algorithm 1 also illustrates how an error bound Error for the approximation is determined. Whenever a path σ is discarded because its probability is too small ($P(\sigma) \leq w$) or it is too long ($|\sigma| > N$), the probability mass of σ and all prolongations of σ is lost for the computation. Not all of these lost paths would actually contribute to the infinite sum. Thus, the error $E(\sigma)$ introduced by discarding a uniformised path σ is bound by

$$
\begin{aligned}
E(\sigma) &\leq \sum_{n=|\sigma|}^{\infty} \mathit{PP}(\lambda t, n) \sum_{\substack{\sigma' \,\in\, \mathit{uPath} \\ |\sigma'| = n \\ \sigma \text{ is prefix of } \sigma'}} P(\sigma') \cdot \underbrace{\Pr\{Y_t \leq r \mid \sigma\}}_{\in[0,1]} \\
&\leq \sum_{n=|\sigma|}^{\infty} \mathit{PP}(\lambda t, n) \cdot P(\sigma) \\
&= \left(1 - \sum_{n=0}^{|\sigma|-1} \mathit{PP}(\lambda t, n)\right) \cdot P(\sigma)
\end{aligned}
$$

The second line follows because the probability mass of all possible prolongations of the same length of a path σ is exactly $P(\sigma)$ (no probability is "lost" when making further steps). The sum of this expression over all paths discarded in the course of the algorithm then gives a valid bound for the error. It can not be computed a priori but only simultaneously with the path generation.

Implementation issues

Different paths might lead to the same colouring and, hence, to the same conditional distribution of state and reward. In an implementation of Algorithm 1 it is, consequently, sensible to generate all relevant paths first and to store only the corresponding colourings together with the accumulated probability of all uniformised path with the same colouring. In the course of the path generation it is not necessary to keep track of the complete path but only of the last state and the probability measure. The generation of paths with probability 0 can easily be avoided. The organisation of the data structure representing the set of paths determines whether paths are generated in a breadth-first or depth-first manner.

For CSRL model checking, we often deal with the case that we have a single goal state and want to know the distribution for all possible starting states. This is most efficiently done by generating the path backwards, starting at the goal state.

It is not reasonable to indicate the complexity of the overall algorithm. The number of paths explored is potentially exponential, but the exploration is restricted by the weight w and the maximal path length N. The complexity of the Diniz algorithm was explained in Section 5.3.1.

5.4 Discretisation

Many general purpose numerical solution methods for ODEs and PDEs are based on the idea of *discretising* the continuous parameters. Tijms and Veldman published an approximate discretisation algorithm for the computation of $\boldsymbol{\Upsilon}(t, y)$ that uses the same step size Δ for both time and accumulated reward [102]. In [51] this algorithm was extended to deal with time-inhomogeneous MRMs. Here we apply it to time- and reward-inhomogeneous MRMs. Like any distribution, $\Upsilon_{ss'}(t, y)$ is a definite integral over the corresponding density:

$$\Upsilon_{ss'}(t, y) = \int_0^y \upsilon_{ss'}(t, x)dx. \tag{5.13}$$

For fixed $t > 0$ and step size Δ we can use the rectangular approximation:

$$\Upsilon_{ss'}(t, y) \approx \sum_{j=1}^{\frac{y}{\Delta}} \upsilon_{ss'}(t, j \cdot \Delta)\Delta. \tag{5.14}$$

For $\Delta \to 0$ we obtain again (5.13). Other approximation schemes, e.g. trapezoid, are possible.

We discretise the time up to t and the accumulated reward up to y in steps of size Δ and consider the density at times $0, \Delta, \cdots, \frac{t}{\Delta}\Delta$, and for accumulated rewards $0, \Delta, \cdots, \frac{y}{\Delta}\Delta$. The densities $\upsilon_{ss'}(\tau, x)$ are also not determined exactly but approximated by $\upsilon^{\Delta}_{ss'}(\tau, x)$ assuming that at most one transition has occurred in a time interval of length Δ. The possibility that two or more transitions occur is neglected. This is a reasonable assumption if Δ is small. The initial values for $\tau = 0$ are given by

$$\upsilon^{\Delta}_{ss'}(0, x) = \begin{cases} \frac{1}{\Delta}, & s = s' \text{ and } x = 0, \\ 0, & \text{otherwise.} \end{cases}$$

Only if $s = s'$ and $x = 0$ the approximate density can be positive at time 0. Since it is a derivative we have to fit it to the step size Δ.

By assuming that either no transition or exactly one transition has occurred in the time interval $[\tau, \tau + \Delta)$, the quantity $\upsilon^{\Delta}_{ss'}(\tau + \Delta, x)$ can be recursively calculated as follows.

$$\begin{aligned} \upsilon^{\Delta}_{ss'}(\tau + \Delta, x) &= \upsilon^{\Delta}_{ss'}(\tau, x - \rho_{s'}(\tau, x)\Delta) \cdot (1 + Q_{s's'}(\tau, x - \rho_{s'}(\tau, x)\Delta) \cdot \Delta) \\ &+ \sum_{z \neq s'} \upsilon^{\Delta}_{sz}(\tau, x - \rho_{s'}(\tau, x)\Delta) \cdot Q_{zs'}(\tau, x - \rho_{s'}(\tau, x)\Delta) \cdot \Delta. \end{aligned} \tag{5.15}$$

Note that this is the *forward* version of (3.8) adapted for inhomogeneous MRMs where the negligible term $o(\Delta)$ for the probability of more than one transition is omitted. We also have to approximate the parameters of the generator and the reward vector. We use $\rho(\tau, x)$ (even though the accumulated reward at time τ is actually smaller, but we do not known by what amount) and instantiate the generator with the resulting reward: $\mathbf{Q}(\tau, x - \rho_{s'}(\tau, x))$. For the homogeneous case, (5.15) reduces to

$$\upsilon^{\Delta}_{ss'}(\tau + \Delta, x) = \upsilon^{\Delta}_{ss'}(\tau, x - \rho_{s'}\Delta) \cdot (1 + Q_{s's'}\Delta) + \sum_{z \neq s'} \upsilon^{\Delta}_{sz}(\tau, x - \rho_{s'}\Delta) \cdot Q_{zs'}\Delta.$$

With the above recursion we automatically have to compute a complete row of the matrix $\mathbf{\Upsilon}(t, y)$. Using the backward equation for *homogeneous* MRMs (3.8), we obtain a complete column of $\mathbf{\Upsilon}(t, y)$:

$$\upsilon^{\Delta}_{ss'}(\tau + \Delta, x) = (1 + Q_{ss}\Delta) \cdot \upsilon^{\Delta}_{ss'}(\tau, x - \rho_s\Delta) + \sum_{z \neq s} Q_{sz}\Delta \upsilon^{\Delta}_{zs'}(\tau, x - \rho_s\Delta). \tag{5.16}$$

The recursions only work if $(1 + Q_{ss}(\tau, x)\Delta)$ and $(Q_{sz}(\tau, x)\Delta)$ are indeed probabilities, that is, if Δ is small enough. This is the case for any state s' if $\Delta \le -\frac{1}{Q_{ss}(\tau, x)}$ for all $s \in S$ and all τ and x [100, p. 31]. No error bound is known for this method.

Implementation issues

In an implementation of (5.15) we replace τ and x by integers. The discrete time $i \in \{0, 1, \ldots, \frac{t}{\Delta}\}$ corresponds to real time $i\Delta$. The discrete reward $j \in \{0, 1, \ldots, \frac{y}{\Delta}\}$ corresponds to true accumulated reward $j\Delta$. To remain inside the discrete time and reward grid when applying (5.15), the reward rates have to be integers as well. As long as the reward rates of the MRM are rational this can be achieved by scaling reward rates and reward bound y by the same number, hence, it is not really a restriction.

We iterate over the discretised time steps, in each iteration the values $v^{\Delta}_{zs'}(i, j)$ for all states z and reward bounds j are computed using (5.15). Finally, the $v^{\Delta}_{ss'}(\frac{t}{\Delta}, j)$ are added according to (5.14) to obtain $\Upsilon_{ss'}(t, y)$.

Space and time complexity

Each vector $v^{\Delta}(i, j)$ consists of $|S|$ real numbers and for a fixed i there are $\frac{y}{\Delta} + 1$ such vectors. It is not necessary to store the vectors $v^{\Delta}(i, j)$ for all i; for each iteration only the previous version (for $i - 1$) is needed. Thus, the space requirement is two times $\mathcal{O}(|S|\frac{y}{\Delta})$. In total there are $\frac{t}{\Delta}$ iterations for the time parameter i, whereas there are $\frac{y}{\Delta} + 1$ inner iterations for the reward parameter j. For each of the $|S|$ entries in the vector $v^{\Delta}(i, j)$, $\mathcal{O}(|S|)$ multiplications and summations have to be performed. In total this leads to a time complexity of $\mathcal{O}(|S|^2 \cdot t \cdot y \cdot \Delta^{-2})$ for the computation of one column of the matrix $\boldsymbol{\Upsilon}(t, y)$. The calculation of the complete matrix is cubic in the number of states.

The accuracy of this discretisation algorithm increases with decreasing Δ. Unfortunately, time complexity grows by a power of 2 with decreasing Δ, leading quickly to infeasible run times. The time complexity derived here does not agree with the time complexity reported in [102]. However, it reflects the way we implemented the algorithm.

5.5 Fully Markovian approximation

The last algorithm we present in this chapter is an approximation for the joint distribution of state and accumulated reward that is based on the transient solution of a derived time-inhomogeneous CTMC, i.e., no rewards are involved. It was first published in [50].

The joint distribution of state and accumulated reward,

$$\Upsilon_{ss'}(t, y) \quad = \quad \Pr\{X_t = s', Y_t \leq y \mid X_0 = s\} \quad = \quad \Pr\{X_t = s', Y_t \in [0, y] \mid X_0 = s\}$$

can be rewritten by summing over evenly-sized subintervals of the reward interval $[0, y]$:

$$\Upsilon_{ss'}(t, y) \quad = \quad \Pr\{X_t = s', Y_t \in [0, \Delta y]\} + \sum_{j=1}^{\frac{y}{\Delta y}-1} \Pr\{X_t = s', Y_t \in (j\Delta y, (j+1)\Delta y]\}.$$

We want to approximate the terms $\Pr\{X_t = s', Y_t \in (j\Delta y, (j+1)\Delta y]\}$ in such a way that the computation is done for a pure CTMC (without rewards). Recall that an MRM $\mathcal{M}$ can be seen as having an infinite and uncountable state space $S \times \mathbb{R}_{\geq 0}$. A joint state then (s, y) indicates that the CTMC part of the MRM is in state s and that the accumulated reward is y. For our approximation, we break down the uncountable state space to an infinite but countable one. Let $\mathcal{C}^{\infty}(t) = (S \times \mathbb{N}, \mathbf{R}^{\infty}(t))$ be a time-inhomogeneous CTMC with infinite state space $S \times \mathbb{N}$. The probability of being in state (s, j) is then the desired approximation:

$$\Pr^{\mathcal{M}}\{X_t = s', Y_t \in (j\Delta y, (j+1)\Delta y] \mid X_0 = s\} \approx \Pi^{\mathcal{C}^\infty(t)}_{(s,0)(s',j)}(t).$$

The rate matrix $\mathbf{R}^\infty(t)$ must contain transitions from one reward level $j\Delta y$ to the next reward level $(j+1)\Delta y$. The rate at which reward is accumulated in a state s in the original MRM is $\rho_s(t, y)$. The natural choice for the rate of accumulating Δy reward, that is, reaching the next reward level, is $\frac{\rho_s(t, j \cdot \Delta y)}{\Delta y}$. Transitions between states at the same reward levels are transfered from the original MRM.

$$R^\infty_{(s,i)(s',j)}(t) = \begin{cases} R_{s,s'}(t, i \cdot \Delta y), & i = j, \\ \frac{\rho_s(t, i \cdot \Delta y)}{\Delta y}, & s = s' \text{ and } j = i + 1, \\ 0, & \text{otherwise.} \end{cases}$$

The rate matrix $\mathbf{R}^\infty(t)$ has a block structure consisting of the original rate matrix $\mathbf{R}(t, y)$ and the matrix $\mathbf{D}(t, y)/\Delta y$, where $\mathbf{D}$ is the diagonal matrix arising from the reward vector $\rho(t, y)$.

$$\mathbf{R}^\infty(t) = \begin{pmatrix} \mathbf{R}(t,0) & \mathbf{D}(t,0)/\Delta y & 0 & \cdots & \cdots \\ 0 & \mathbf{R}(t,\Delta y) & \mathbf{D}(t,\Delta y)/\Delta y & 0 & \cdots \\ 0 & 0 & \mathbf{R}(t, 2\Delta y) & \mathbf{D}(t, 2\Delta y)/\Delta y & \cdots \\ 0 & 0 & 0 & \mathbf{R}(t, 3\Delta y) & \cdots \\ 0 & 0 & 0 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix}$$

The infinite CTMC is a time-inhomogeneous quasi-birth process (a subclass of quasi-birth-death processes [79]). Transitions are only possible within a level (matrix $\mathbf{R}(t, i \cdot \Delta y)$ or to the next higher level (matrix $\mathbf{D}(t, i \cdot \Delta y)/\Delta y$).

Following the above procedure, we constructed a fully Markovian model in order to approximate the joint distribution of state and reward in an MRM. The time it takes to jump from one reward level i to the next is exponentially distributed with rate $\rho_s(t, i\Delta y)/\Delta y$. Thus, the rate at which the reward level changes is $\frac{\rho_s(t, i\Delta y)}{\Delta y} \cdot \Delta y = \rho_s(t, i\Delta y)$, independent of Δy. By letting $\Delta y \to 0$, the fully Markovian model crossfades into the original MRM. For reasonably small Δy it should give us a decent approximation.

The transient probabilities $\Pi^{\mathcal{C}^\infty(t)}_{(s,0)(s',j)}(t)$ needed for the approximation can efficiently be computed using uniformisation even though the CTMC $\mathcal{C}^\infty$ has infinite state space [89, 88] if the QBD is homogeneous. As long as the time-inhomogeneity is restricted to a single point in time, double uniformisation can be employed to obtain the transient probabilities (cf. Section 4.7.2).

We are not able to indicate the error introduced by this approximation. Of course, the approximation will get more accurate for smaller Δy. The accuracy is also influenced by the error bound used for computing the transient probabilities using uniformisation.

Example 27. The generator matrix $\mathbf{R}^{\infty(2)}$ for the second phase of the example MRM

$$\mathcal{M}\left\langle \texttt{empty}\,\mathcal{U}^{[1,5]}_{[20,50]}(\neg\texttt{intact})\right\rangle$$

(cf. Figure 4.1(b)) for the Markovian approximation with a step size $\Delta y = 0.1$ is composed of the rate matrix

$$\mathbf{R}^{(2)} = \begin{pmatrix} 0 & 3 & 6 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

and the diagonal matrix

$$\mathbf{D}^{(2)} = \begin{pmatrix} 500 & 0 & 0 & 0 & 0 \\ 0 & 200 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

$\Diamond$

Space and time complexity

The submatrices of the uniformised generator $\mathbf{U}^{\infty}(t)$ are stored independently in a sparse way, thus the space requirement here is $\mathcal{O}(|S|^2)$. If we want to compute $\boldsymbol{\Upsilon}(t,y)$, the entries $\Pi_{(s,0)(s',j)}$, for $j = 0,\ldots,\frac{y}{\Delta y}$, are of interest, so only fractions of $\boldsymbol{\Pi}$ and $\mathbf{U}^n$ of size $|S|\cdot(\frac{y}{\Delta y}+1)|S|$ have to be stored. The space requirement $\mathcal{O}(|S|^2\cdot y\Delta y^{-1})$ grows for smaller Δy. Each row or column in the uniformised generator has at most $|S|+1$ entries, hence, the matrix/matrix multiplication in each step requires at most $|S|^2\frac{y}{\Delta y}\cdot|S| = |S|^3\cdot\frac{y}{\Delta y}$ multiplications. The number of iterations needed depends on the uniformisation constant and the time t. The uniformisation constant in turn is a linear function of Δy^{-1}. The overall time complexity is then $\mathcal{O}(\Delta y^{-1}t\cdot|S|^3\cdot y\cdot\Delta y^{-1}) = \mathcal{O}(t\cdot|S|^3\cdot y\cdot\Delta y^{-2})$. If we only want to compute a single entry (or a column) of the matrix $\boldsymbol{\Upsilon}(t,y)$, the time complexity is quadratic in the number of states.

5.6 Discussion and Comparison

Table 5.2 gives an overview of the space and time complexity of the different algorithms. We omit the path exploration because its complexity is theoretically exponential in the number of transitions allowed in the paths. For fair comparison, we stated the complexity for the computation of all entries of the matrix $\boldsymbol{\Upsilon}(t,y)$ and not only for a single row or column. As one can easily see, all algorithms have a space complexity that is quadratic and a time complexity that is cubic in the number of states. Note, that the discretisation

	space	time
Picard's method	$\mathcal{O}(\|S\|^2 \cdot t \cdot y \cdot \Delta t^{-2})$	$\mathcal{O}(\|S\|^3 \cdot N \cdot t^2 \cdot y \cdot \Delta t^{-3})$
Uniformisation	$\mathcal{O}(\|S\|^2 \cdot \lambda \cdot t \cdot K)$	$\mathcal{O}(\|S\|^3 \cdot (\lambda t)^2)$
Discretisation	$\mathcal{O}(\|S\|^2 \cdot y \cdot \Delta^{-1})$	$\mathcal{O}(\|S\|^3 \cdot t \cdot y \cdot \Delta^{-2})$
Markovian approximation	$\mathcal{O}(\|S\|^2 \cdot y \cdot \Delta y^{-1})$	$\mathcal{O}(\|S\|^3 \cdot t \cdot y \cdot \Delta y^{-2})$

Table 5.2: Space and time complexity of the algorithms when computing the complete matrix $\mathbf{\Upsilon}(t, y)$

algorithm and the Markovian approximation have the same complexity with respect to their discretisation parameters.

We apply the presented algorithm to the computation of the probability measure for until formulas $\texttt{empty}\,\mathcal{U}^I_J(\neg\texttt{intact})$ in the illustrating example. We have to calculate the joint distribution in the transformed (possibly inhomogeneous) MRM $\mathcal{M}\left\langle \texttt{empty}\,\mathcal{U}^I_J(\neg\texttt{intact})\right\rangle$.

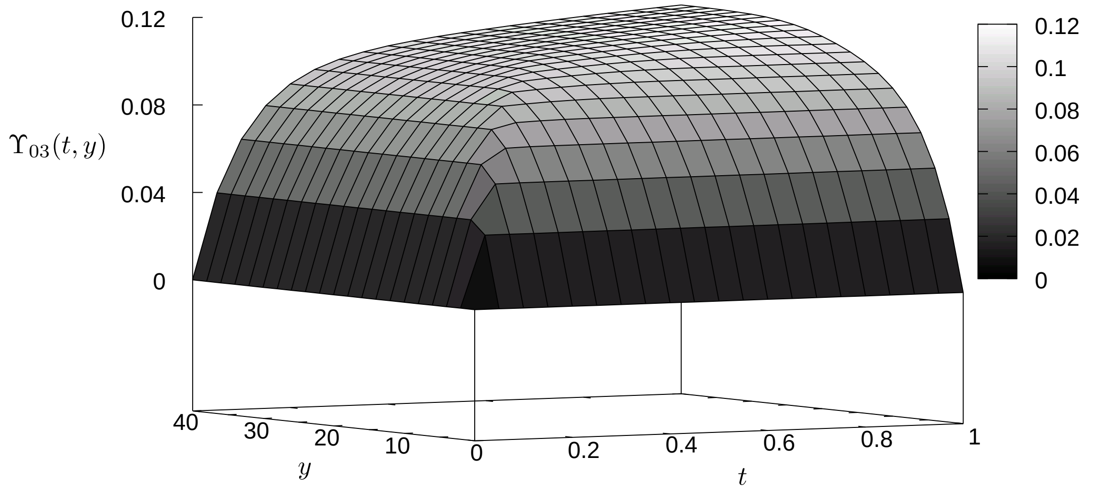
In Section 5.6.1 we visualise the joint probability for different $I = [0, t]$ and $[0, y]$ (Section 5.6.1). In Section 5.6.2 we then compare the results of the five algorithms for four selected pairs of intervals $I = [0, t]$ and $J = [0, y]$, that is, we compute the joint distribution in the *homogeneous* MRM $\mathcal{M}[(\neg\texttt{empty}) \vee (\neg\texttt{intact})]$. We restrict ourselves to starting states 0 and 1, since the joint probability of reaching absorbing state 3 from itself (before time t and reward y) is always one and state 3 is never reachable from absorbing state 2.

Subsequently, the scaling of the algorithms with the size of the state space is discussed in Section 5.6.4. The inhomogeneous case is treated in Section 5.6.3, where we compare the discretisation algorithm and the Markovian approximation.

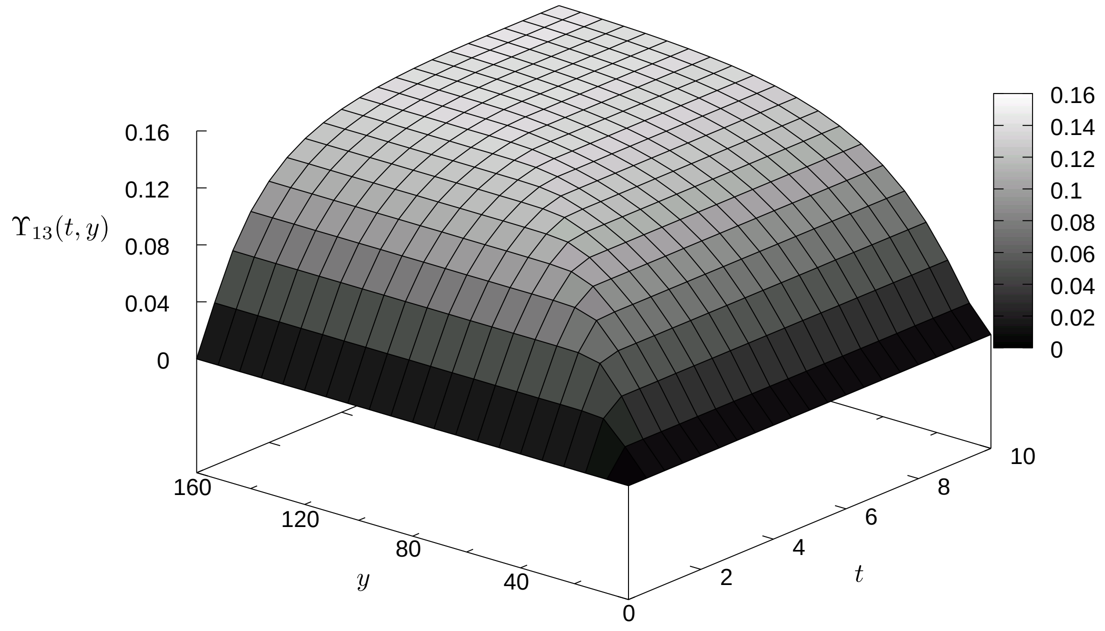
5.6.1 Visualisation of the joint distribution

Figure 5.2(a) shows the probability $\Upsilon_{03}(t, y)$ of being in the goal state 3 at time t with an accumulated reward of at most y, having started in state 0 of MRM $\mathcal{M}[(\neg\texttt{empty}) \vee (\neg\texttt{intact})]$, as computed with the uniformisation algorithm by Sericola (cf. Section 5.2). This probability equals $Prob^{\mathcal{M}}_0(\texttt{empty}\,\mathcal{U}^{[0,t]}_{[0,y]}(\neg\texttt{intact}))$.

For $t = 0$, the joint probability is zero because it is not possible to reach another state. The same is true for $y = 0$, since after a positive amount of time spent in state 0, the accumulated reward is greater than 0 because state 0 has a positive reward rate. The more time is available, the larger is the probability of reaching another state. Therefore the joint probability grows with t. It also grows with y, since it is more probable to stay below a higher bound y. The joint probability for y is bound by the transient probability $\Pi_{03}(t)$. Figure 5.2(b) shows $\Upsilon_{13}(t, y)$ The same explanation applies as for $\Upsilon_{03}(t, y)$. Note that we considered different ranges for time and reward.



(a) starting from state 0



(b) starting from state 1

Figure 5.2: The joint probability of being in state 3 at time t, having accumulated at most y reward

5.6.2 Homogeneous MRMs

In the following we concentrate on four time- and reward-bound pairs: $(0.5, 5)$, $(0.5, 50)$, $(10, 5)$, and $(10, 50)$. They are chosen such that we have all combination of a small and a big time bound and a small and big reward bound. We apply the different algorithms and compute $\Upsilon_{03}(t, y)$ and $\Upsilon_{13}(t, y)$ for each of these pairs in the MRM $\mathcal{M}[(\neg\texttt{empty}) \vee (\neg\texttt{intact})]$. We study the behaviour of the algorithms with respect to accuracy and run time.

For all algorithms we used prototype C++ implementations that operate on the same representation of the MRM. All experiments described in the following were performed on a PC with a Pentium 4 CPU at 3 GHz, 1 GB of main memory running Linux.

If no error bound can be computed for the results of an algorithm, the relative error with respect to the result of the uniformisation-based algorithm (cf. Section 5.2) is indicated. We start with the two algorithms where an error bound can be computed: uniformisation and path generation.

Uniformisation (Section 5.2)

For the uniformisation algorithm we can fix an error bound a priori. We performed the computations for an accuracy ε of 10^{-2}, 10^{-5} and 10^{-10}. In Table 5.3 we present the results; it has four parts, one for each time/reward pair. In each line there is the predefined accuracy ε, the number of steps required to reach this accuracy N, the numerical results of the algorithm and the run time (user time) in seconds. The accuracy and the time bound influence N (see also Section 3.2), the number of iterations needed. The reward bound plays no role when determining N; the number of iterations is the same if the time bound remains unchanged. We underline the digits that are known to be correct, for an accuracy of 10^{-2} this is only the first digit after the decimal point, for 10^{-5} the first four digits are correct, and so on. The run time for this small example is reasonable, it stays far below one second.

Path exploration (Section 5.3)

With path exploration, an error bound can only be determined a posteriori. In Table 5.4 we list the result of this algorithm. For each time/reward pair we fixed different path weights w and accuracy parameters ε for the involved Poisson probabilities. The table shows the number of colourings $|K|$ that are considered, the numerical results, the computed error bound `Error` and the run time in seconds. The reward bound has no influence on the number of colourings, which is proportional to t, w^{-1} and ε^{-1}. The correct digits are underlined. We see that a reasonable error bound `Error` requires a small w and leads to run times that are higher than the ones needed by the uniformisation algorithm. For $t = 10$ the error bound is quite coarse but comparing the results with the ones of the uniformisation algorithm shows that the results are actually quite accurate. A more acceptable error bound could not be obtained within reasonable time.

ε	N	$\Upsilon_{03}(0.5, 5)$	$\Upsilon_{13}(0.5, 5)$	run time
10^{-2}	11	$0.\underline{0}634002274$	$0.\underline{0}082920545$	0.001
10^{-5}	17	$0.\underline{0639}407262$	$0.\underline{0084}704560$	0.002
10^{-10}	25	$0.\underline{063941316}7$	$0.\underline{008470732}5$	0.004

ε	N	$\Upsilon_{03}(0.5, 50)$	$\Upsilon_{13}(0.5, 50)$	run time
10^{-2}	11	$0.\underline{1}065953160$	$0.\underline{0}335501655$	0.000
10^{-5}	17	$0.\underline{1072}623268$	$0.\underline{0339}827871$	0.001
10^{-10}	25	$0.\underline{107263026}9$	$0.\underline{033983332}3$	0.003

ε	N	$\Upsilon_{03}(10, 5)$	$\Upsilon_{13}(10, 5)$	run time
10^{-2}	124	$0.\underline{0}632945352$	$0.\underline{0}083605116$	0.096
10^{-5}	145	$0.\underline{0639}405532$	$0.\underline{0084}705817$	0.133
10^{-10}	170	$0.\underline{063941316}7$	$0.\underline{008470732}5$	0.181

ε	N	$\Upsilon_{03}(10, 50)$	$\Upsilon_{13}(10, 50)$	run time
10^{-2}	124	$0.\underline{1}296541394$	$0.\underline{1}081752880$	0.096
10^{-5}	145	$0.\underline{1308}401437$	$0.\underline{1092}426524$	0.132
10^{-10}	170	$0.\underline{130841467}7$	$0.\underline{109243881}9$	0.181

Table 5.3: Results for the uniformisation algorithm by Sericola

w	ε	$\lvert K\rvert$	$\Upsilon_{03}(0.5,5)$	$\Upsilon_{13}(0.5,5)$	`Error`	run time
10^{-5}	10^{-2}	151	$0.\underline{0}634002249$	$0.\underline{0}082919730$	0.0164576091	0.014
	10^{-5}	382	$0.\underline{06}39407215$	$0.\underline{008}4703406$	0.0000755217	0.019
	10^{-10}	858	$0.\underline{0639}413120$	$0.\underline{008}4706168$	0.0000511579	0.036
10^{-10}	10^{-2}	173	$0.\underline{0}634002274$	$0.\underline{0}082920545$	0.0164164853	0.014
	10^{-5}	521	$0.\underline{0639}407262$	$0.\underline{0084}704560$	0.0000244115	0.048
	10^{-10}	1321	$0.\underline{063941316}7$	$0.\underline{008470732}5$	0.0000000002	1.438

w	ε	$\lvert K\rvert$	$\Upsilon_{03}(0.5,50)$	$\Upsilon_{13}(0.5,50)$	`Error`	run time
10^{-5}	10^{-2}	151	$0.\underline{1}065921777$	$0.\underline{0}335279648$	0.0164576091	0.013
	10^{-5}	382	$0.\underline{107}2584800$	$0.\underline{03}39573909$	0.0000755217	0.018
	10^{-10}	858	$0.\underline{107}2591776$	$0.\underline{03}39579258$	0.0000511579	0.035
10^{-10}	10^{-2}	173	$0.\underline{1}065953160$	$0.\underline{0}335501655$	0.0164164853	0.014
	10^{-5}	521	$0.\underline{1072}623268$	$0.\underline{03}39827871$	0.0000244115	0.042
	10^{-10}	1321	$0.\underline{10726302}69$	$0.\underline{033983332}3$	0.0000000002	1.444

w	ε	$\lvert K\rvert$	$\Upsilon_{03}(10,5)$	$\Upsilon_{13}(10,5)$	`Error`	run time
10^{-5}	10^{-2}	13757	$0.\underline{063}2945307$	$0.\underline{008}3603996$	0.2250183464	5.231
	10^{-5}	16781	$0.\underline{06394}05485$	$0.\underline{008470}4660$	0.2109555759	7.704
	10^{-10}	20381	$0.\underline{06394}05485$	$0.\underline{008470}4660$	0.2109376686	11.268

w	ε	$\lvert K\rvert$	$\Upsilon_{03}(10,50)$	$\Upsilon_{13}(10,50)$	`Error`	run time
10^{-5}	10^{-2}	13757	$0.\underline{1}289257945$	$0.\underline{10}56186394$	0.2250183464	5.266
	10^{-5}	16781	$0.\underline{130}0958127$	$0.\underline{10}66454514$	0.2109555759	7.601
	10^{-10}	20381	$0.\underline{130}0971095$	$0.\underline{10}66466198$	0.2109376686	11.380

Table 5.4: Results for path exploration

Picard's method (Section 5.1)

Table 5.5 shows the results for Picard's method of successive approximations. For different step sizes Δt, we show N, the number of iterations needed, the resulting probabilities $\Upsilon_{03}(t, y)$ and $\Upsilon_{13}(t, y)$, the corresponding relative errors E_0 and E_1, and the run time. Correct digits of the resulting values are again underlined. In all computations the iteration is stopped when the first 10 digits after the decimal points remain unchanged. The relative error is determined with respect to the results of the uniformisation algorithm.

The results computed by Picard's method are not very accurate. The relative errors are in general larger than 1%, even for small step sizes. The run time grows with t and y since they influence the number of sample points. The algorithm is cubic in the inverse of the step size, Δt^{-1}. The number of iterations only decreases marginally for decreasing Δ, hence, the run time grows roughly with a factor $2^3 = 8$ when halving the step size Δt. To sum up, Picard's method is slow and gives inaccurate results.

Δt	N	$\Upsilon_{03}(0.5, 5)$	E_0	$\Upsilon_{13}(0.5, 5)$	E_1	run time
0.1	12	$0.\underline{0}516995415$	0.1914532862	$0.\underline{0}113302765$	0.3375793042	0.012
0.05	11	$0.\underline{0}562477531$	0.1203222577	$0.\underline{008}6931856$	0.0262613820	0.068
0.025	11	$0.\underline{0}596981944$	0.0663596326	$0.\underline{008}3576872$	0.0133453959	0.455
0.0125	10	$0.\underline{06}17198406$	0.0347424193	$0.\underline{008}3563562$	0.0135025256	3.656
0.00625	10	$0.\underline{06}28055044$	0.0177633543	$0.\underline{008}3988701$	0.0084836096	29.432

Δt	N	$\Upsilon_{03}(0.5, 50)$	E_0	$\Upsilon_{13}(0.5, 50)$	E_1	run time
0.1	15	$0.\underline{1}144351244$	0.0668645822	$0.\underline{03}21329463$	0.0544498092	0.163
0.05	13	$0.\underline{10}83912247$	0.0105180493	$0.\underline{03}17329274$	0.0662208410	1.062
0.025	13	$0.\underline{107}1863011$	0.0007153056	$0.\underline{03}25110439$	0.0433238360	8.454
0.0125	13	$0.\underline{107}0615226$	0.0018786003	$0.\underline{033}1580871$	0.0242838202	68.364

Δt	N	$\Upsilon_{03}(10, 5)$	E_0	$\Upsilon_{13}(10, 5)$	E_1	run time
0.1	12	$0.\underline{0}516995415$	0.1914532862	$0.\underline{0}113302765$	0.3375793041	2.504
0.05	11	$0.\underline{0}562477531$	0.1203222577	$0.\underline{008}6931856$	0.0262613819	18.523
0.025	10	$0.\underline{0}596981944$	0.0663596325	$0.\underline{008}3576872$	0.0133453960	138.352

Δt	N	$\Upsilon_{03}(10, 50)$	E_0	$\Upsilon_{13}(10, 50)$	E_1	run time
0.1	20	$0.\underline{1}455457258$	0.1123822470	$0.\underline{1}212599414$	0.1099929751	49.196
0.05	19	$0.\underline{13}44429393$	0.0275254606	$0.\underline{1}121085741$	0.0262229081	378.298

Table 5.5: Results for Picard's method of successive approximations

Discretisation (Section 5.4)

Table 5.6 contains the results, the relative errors and the run time of the discretisation algorithm subject to the step size Δ. Correct digits of the results are underlined. Using this algorithm it is possible to obtain results with an relative error below 1% with reasonable run times.

One can easily see that both t and y have influence on the run time. Halving the step size Δ approximately quadruples the run time which confirms that the algorithm has a time complexity in Δ^{-2}. The values show that the discretisation algorithm provides usable results in tolerable time.

Markovian approximation (Section 5.5)

The last algorithm to evaluate is the Markovian approximation. Table 5.7 shows the results, the relative errors and the run time depending on the step size for the accumulated reward Δy. For the calculation of the transient probabilities via uniformisation we chose the error bound $\varepsilon = 10^{-10}$. The correct digits of the results are underlined.

It is possible to obtain quite accurate results with a relative error below 1% with still short run times. The parameter t influences the run time via the number of steps considered by uniformisation. As could be expected from the time complexity, the runtime depends also linearly on the reward bound y: when switching from 5 to 50, the run time grows approximately by a factor 10. The runtime also grows with Δy^{-2}: If we divide Δy by 10, the run time is multiplied with 100. Among the algorithms without known error bound, the Markovian approximation is the fastest while giving adequately accurate results.

5.6.3 Inhomogeneous MRMs

Only two algorithms are available for the computation of the joint distribution in an inhomogeneous MRM arising when model checking a CSRL formula involving the until operator with general time and reward intervals: the discretisation algorithm and the Markovian approximation.

Table 5.8 shows the results and the run time for the two algorithms when computing the joint distribution of state and accumulated reward for $t = 5$ and $y = 50$ in the inhomogeneous MRM $\mathcal{M}\left\langle \mathtt{empty}\,\mathcal{U}^{[1,5]}_{(20,50]}(\neg\mathtt{intact})\right\rangle$. Note that with the provided accuracy the results of the two algorithms only agree up to the second digit after the decimal point. The run time increases with decreasing step sizes Δ and Δy in exactly the same manner as was the case for homogeneous MRMs: halving the step size Δ in the discretisation algorithm roughly quadruples the run time, whereas dividing the reward step size Δy by 10 stretches the run time approximately by a factor 100. However, the run time is generally higher than in the homogeneous case.

We cannot give a definite recommendation which algorithm to use for time- and reward-inhomogeneous MRMs. Both algorithms give approximate results for which we do not know

Δ	$\Upsilon_{03}(0.5,5)$	E_0	$\Upsilon_{13}(0.5,5)$	E_1	run time
0.05	0.0750000000	0.1729505099	0.0048750000	0.4244889689	0.001
0.025	$0.\underline{06}87212757$	0.0747554043	$0.\underline{00}69969034$	0.1739907521	0.001
0.0125	$0.\underline{06}61882849$	0.0351411010	$0.\underline{00}77937131$	0.0799245465	0.007
0.00625	$0.\underline{06}50329335$	0.0170721667	$0.\underline{008}1453830$	0.0384086677	0.030
0.003125	$0.\underline{06}44795753$	0.0084180099	$0.\underline{008}3111549$	0.0188387048	0.121
0.001	$0.\underline{06}41119638$	0.0026688078	$0.\underline{0084}203199$	0.0059513874	1.189
0.0005	$0.\underline{06}40264552$	0.0013315098	$0.\underline{0084}456017$	0.0029667800	4.796
0.00025	$0.\underline{0639}838398$	0.0006650344	$0.\underline{0084}581859$	0.0014811717	19.797

Δ	$\Upsilon_{03}(0.5,50)$	E_0	$\Upsilon_{13}(0.5,50)$	E_1	run time
0.05	$0.\underline{10}66463496$	0.0057492074	$0.\underline{03}07782659$	0.0943128927	0.004
0.025	$0.\underline{10}69765505$	0.0026707848	$0.\underline{03}24032972$	0.0464944117	0.017
0.0125	$0.\underline{107}1232656$	0.0013029779	$0.\underline{033}1990077$	0.0230796829	0.078
0.00625	$0.\underline{107}1938269$	0.0006451436	$0.\underline{033}5925959$	0.0114978835	0.306
0.003125	$0.\underline{1072}285767$	0.0003211757	$0.\underline{033}7883205$	0.0057384517	1.234
0.001	$0.\underline{1072}520331$	0.0001024939	$0.\underline{0339}210061$	0.0018340223	12.449
0.0005	$0.\underline{1072}575335$	0.0000512149	$0.\underline{0339}521783$	0.0009167426	50.259

Δ	$\Upsilon_{03}(10,5)$	E_0	$\Upsilon_{13}(10,5)$	E_1	run time
0.05	$0.\underline{0}750000000$	0.1729505100	$0.\underline{00}48750000$	0.4244889689	0.009
0.025	$0.0.\underline{06}87212757$	0.0747554043	$0.\underline{00}69969034$	0.1739907522	0.035
0.0125	$0.0.\underline{06}61882849$	0.0351411010	$0.\underline{00}77937131$	0.0799245466	0.149
0.00625	$0.0.\underline{06}50329335$	0.0170721667	$0.\underline{008}1453830$	0.0384086678	0.605
0.003125	$0.0.\underline{06}44795753$	0.0084180099	$0.\underline{008}3111549$	0.0188387049	2.439
0.001	$0.0.\underline{06}41119638$	0.0026688079	$0.\underline{0084}203199$	0.0059513874	23.965
0.0005	$0.\underline{06}40264552$	0.0013315099	$0.\underline{0084}456017$	0.0029667801	96.587

Δ	$\Upsilon_{03}(10,50)$	E_0	$\Upsilon_{13}(10,50)$	E_1	run time
0.05	$0.\underline{13}11687319$	0.0025012271	$0.\underline{109}9822018$	0.0067584559	0.093
0.025	$0.\underline{13}10040101$	0.0012422851	$0.\underline{109}6178799$	0.0034235151	0.387
0.0125	$0.\underline{130}9227493$	0.0006212220	$0.\underline{109}4320024$	0.0017220236	1.526
0.00625	$0.\underline{1308}821415$	0.0003108636	$0.\underline{109}3382067$	0.0008634334	6.140
0.003125	$0.\underline{1308}618164$	0.0001555224	$0.\underline{1092}911083$	0.0004323027	25.167
0.001	$0.\underline{1308}479823$	0.0000497902	$0.\underline{1092}590080$	0.0001384620	254.908

Table 5.6: Results for the discretisation algorithm

Δy	$\Upsilon_{03}(0.5, 5)$	E_0	$\Upsilon_{13}(0.5, 5)$	E_1	run time
5	$0.\underline{0}513626012$	0.1967228093	$0.\underline{00}93781304$	0.1071215345	0.000
1	$0.\underline{06}07551369$	0.0498297498	$0.\underline{008}9874458$	0.0609998407	0.000
0.1	$0.\underline{063}5997638$	0.0053416621	$0.\underline{008}5317981$	0.0072090151	0.003
0.01	$0.\underline{0639}069058$	0.0005381640	$0.\underline{0084}769078$	0.0007290208	0.261
0.001	$0.\underline{0639}378730$	0.0000538567	$0.\underline{0084}713507$	0.0000729844	23.468

Δy	$\Upsilon_{03}(0.5, 50)$	E_0	$\Upsilon_{13}(0.5, 50)$	E_1	run time
5	$0.\underline{10}86759849$	0.0131728332	$0.\underline{03}76342133$	0.1074315205	0.000
1	$0.\underline{107}7516872$	0.0045557191	$0.\underline{03}52046604$	0.0359390351	0.001
0.1	$0.\underline{107}3205166$	0.0005359690	$0.\underline{03}41270179$	0.0042281220	0.031
0.01	$0.\underline{1072}688792$	0.0000545598	$0.\underline{0339}979589$	0.0004304077	2.593
0.001	$0.\underline{107263}6132$	0.0000054658	$0.\underline{03398}47976$	0.0000431183	236.109

Δy	$\Upsilon_{03}(10, 5)$	E_0	$\Upsilon_{13}(10, 5)$	E_1	run time
5	$0.\underline{0}515463917$	0.1938484468	$0.\underline{0}103092783$	0.2170468539	0.000
1	$0.\underline{06}07582307$	0.0497813635	$0.\underline{00}90193561$	0.0647669641	0.000
0.1	$0.\underline{063}5997638$	0.0053416620	$0.\underline{008}5317981$	0.0072090151	0.052
0.01	$0.\underline{0639}069058$	0.0005381639	$0.\underline{0084}769078$	0.0007290207	4.556
0.001	$0.\underline{0639}378730$	0.0000538567	$0.\underline{0084}713507$	0.0000729843	444.076

Δy	$\Upsilon_{03}(10, 50)$	E_0	$\Upsilon_{13}(10, 50)$	E_1	run time
5	$0.\underline{1}293036476$	0.0117533077	$0.\underline{10}50985308$	0.0379458417	0.001
1	$0.\underline{130}5312240$	0.0023711414	$0.\underline{10}83845730$	0.0078659683	0.007
0.1	$0.\underline{1308}102979$	0.0002382254	$0.\underline{109}1572657$	0.0007928699	0.501
0.01	$0.\underline{1308}383491$	0.0000238349	$0.\underline{1092}352134$	0.0000793498	45.816

Table 5.7: Results for the Markovian approximation

the error bound. For model checking the CSRL until operator it might be wise to apply both, and then decide carefully whether the probabilities meet the given bound or not.

Δ	$\Upsilon_{03}(10,50)$	$\Upsilon_{13}(10,50)$	run time
0.05	0.0179025467	0.0512607935	4.31334
0.025	0.0170903133	0.0489142692	17.3564
0.0125	0.0166817295	0.0477566947	73.4648
0.00625	0.0164771336	0.0471818209	293.524

(a) Results for the discretisation algorithm

Δy	$\Upsilon_{03}(10,50)$	$\Upsilon_{13}(10,50)$	run time
5	0.0117716488	0.0354429180	0.006
1	0.0137651867	0.0419738119	0.056
0.1	0.0143490683	0.0443792494	3.675
0.01	0.0143968991	0.0446936737	317.564

(b) Results for the Markovian approximation

Table 5.8: The joint distribution for the inhomogeneous MRM

5.6.4 Scaling the size of the state space

The results shown so far suggest that the uniformisation algorithm is fast and the most accurate of the five algorithms. However, we have not shown the behaviour of the algorithms for different sizes of the state space. In this section we compare the run times for growing model sizes. For this purpose we modify the MRM underlying the Google file system model which is going to be presented in the next chapter (see Section 6.3 for a detailed description). We modify it such that the transition rates are independent from the number of states. As reward rate we take the number of available chunk servers if the master is up, and zero otherwise. By varying the number of chunk servers between 3 and 20 we obtain MRMs with 60 to 2406 states.

The atomic proposition `service_level_3` is defined to hold in those states of the MRM where three replicas of certain data block (a so-called chunk) are present. We chose the probability measure of the until path formula

$$\texttt{true}\,\mathcal{U}^{[0,0.5]}_{[0,1]}\texttt{service\_level\_3}$$

for our comparison and so each of the algorithms was employed to compute the joint distribution in the homogeneous second phase of the corresponding MRM.

As basis for the comparison we took the smallest MRM with 60 states and computed the results using the uniformisation algorithm with error bound $\varepsilon = 10^{-5}$. For the other

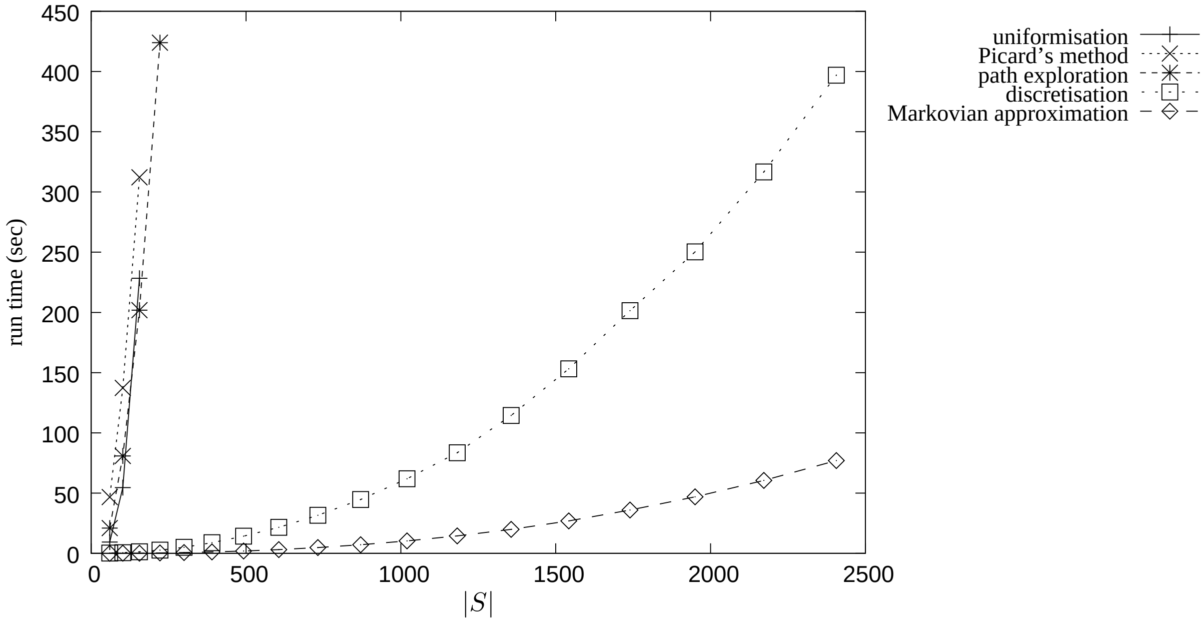


Figure 5.3: Run times for the five algorithms when scaling the size of the state space

four algorithms we chose the parameters in such a way that the average relative error with respect to uniformisation was at most 5%. We took the average because some of the algorithms would not result in all relative errors being below a small threshold within reasonable time.

This results in a path weight $w = 2 \cdot 10^{-4}$ and a bound for the error induced by the Poisson probabilities $\varepsilon = 10^{-5}$ for path exploration. For Picard's method, the time step size is $\Delta t = 2.5 \cdot 10^{-2}$ and we consider four non-changing digits after the decimal point, For discretisation we use a step size $\Delta = 1.25 \cdot 10^{-2}$. Finally, for the Markovian approximation, the reward step size is $\Delta y = 10^{-1}$. We kept the parameters unchanged when scaling the size of the state space.

Figure 5.3 shows the run times for the five algorithms. We clearly distinguish two groups of algorithms. The first group contains the two algorithms with known error bound, that is, uniformisation and path exploration, together with Picard's method. The algorithms of this group are only suitable for small state spaces, here up to 102 and 156 states, respectively. The second group encompasses discretisation and the Markovian approximation. With both algorithms we are able to compute the joint distribution of an MRM with more than 2400 states. Both algorithms exhibit run times that grow quadratically with the number of states. In case of discretisation, the constant is bigger: the run time is five to six times higher than the run time of the Markovian approximation.

Summarising, it can be stated that the only suitable algorithms for larger state spaces are the discretisation algorithm and the Markovian approximation. The latter is preferable because it is faster while having comparable accuracy. Drawback of both algorithms is that no error bound is known, in contrast to uniformisation and path exploration. The practical applicability of Picard's method is negligible.

5.7 Summary

In this chapter we presented five algorithms for the computation of the joint transient distribution of state and accumulated reward. Three of them are only applicable if the MRM under consideration is time- and reward-homogeneous (uniformisation, path exploration and Picard's method). Two can be employed with inhomogeneous MRMs as well (discretisation and Markovian approximation). We only know error bounds for uniformisation and path exploration, for the other algorithms the accuracy is unknown. Experiments show that for a small homogeneous MRM accuracy is easiest to obtain when using the uniformisation algorithm. However, it does not scale well. For larger models (several thousands of states), only discretisation or – even faster – Markovian approximation are to be considered. In the inhomogeneous case we can not recommend an algorithm as we do not know an error bound. But if we extrapolate the results respecting accuracy and run time from the homogeneous case, the Markovian approximation is faster.

Chapter 6

Model checking for survivability

Many parts of everyday life depend heavily on large-scale communication and information systems. It is therefore essential for such systems to survive catastrophic events. However, the concept of survivability is not restricted to this type of technical system. It is also known for military devices, for example, aircraft combat survivability, and even in agriculture [68].

The goal of this chapter is to show a direct and precise method for the quantitative evaluation of survivability by using a CSRL model-checking approach [26, 25]. Section 6.1 introduces an intuitive definition of survivability and gives a short overview of how survivability evaluation was tackled in the literature. In Section 6.2 we show how we can phrase survivability in terms of CSRL and give results for the illustrating example. Section 6.3 assesses the survivability of the Google file system [42]. This scalable file system is especially designed to support large data-intensive applications, primarily the Google search engines. We study its behaviour under disaster conditions.

6.1 What is survivability?

The literature is abundant with different definitions of survivability. For an overview see for example [63, 70]. Distinct definitions stress different aspects of survivability, be it the detection of faults, the defence against attacks or the recovery from various types of disasters. The definition we will use throughout this chapter focuses on the behaviour of a system after a disaster has occurred. Note that we do not introduce a new definition of survivability but state a slightly generalised version of the one in [39]; it reflects an intuitively appealing view on survivability but is therefore also quite imprecise:

> *Survivability is the ability of a system to* ***recover*** *predefined* ***service*** *levels in a* ***timely manner*** *after the occurrence of* ***disasters****.*

A disaster might be any kind of severe disturbance of the communication system, for example, a power breakdown or the complete or partial cut of communication lines. The possible causes are manifold and include purposeful attacks as well as natural disasters like earthquakes or thunderstorms.

A system is survivable if it includes mechanisms to return to normal service *in a timely manner* even though a disaster occurred. What kind of mechanisms are used to ensure survivability and how they are implemented is not part of the survivability definition. One possible mechanism to achieve survivability is fault tolerance [84].

The above definition of survivability does not give at all a precise recipe how to decide whether a system is survivable or not. To overcome this, many approaches have been followed in the literature for the quantitative determination of survivability [46, 67, 69, 70, 72, 97, 99, 108]. Most of them are model-based and suggest some measure on the system (model) behaviour and study its evolution after the occurrence of a disaster. It is thus the deliberate decision of the person performing the survivability evaluation to choose an appropriate measure.

6.2 Expressing survivability using CSRL

With CSRL we can describe survivability following the intuitive definition closely, including the quantitative aspects. In doing so, it is not necessary to rely on an arbitrary measure as an indicator for survivability. It is a model design decision whether to include the possible occurrence of disasters into a survivability model or not [108]. Out of this decision two types of models appeared in the literature. In a so-called *given-occurrence-of-disaster* (GOOD) model, there are no transitions that correspond to the incidence of a disaster. For such systems, a disaster is such a rare event that we can not include its occurrence in the model appropriately. We simply *assume* that an external disaster has happened and start our observation just after its occurrence, that is, we set the starting conditions (probabilities) such as if a disaster has just happened. On the other hand, so-called *random-occurrence-of-disaster* (ROOD) models include transitions that indicate the incidence of disasters. Both approaches are discussed below, in Sections 6.2.1 and 6.2.2, respectively. In Section 6.2.3 we generalise the concept of survivability to multiple levels of disaster and service.

6.2.1 Survivability for GOOD models

In this section we phrase the constituent ingredients of survivability as CSRL formulas. Putting the formulas together in a suitable manner, we will end up with a global CSRL formula describing survivability.

Disaster and service. First of all we design a CSRL state formula `disaster` describing those states of our system model that are disaster states, that is, states that will be occupied just after a disaster has occurred. We may choose whatever we feel is suitable in the given context. For the specification of states providing a predefined service level we specify a CSRL formula `service`.

Example 28. For the illustrating example (cf. Section 2.9), we set

$$\texttt{disaster} \equiv \texttt{broken}, \tag{6.1}$$

that is, the occurrence of a disaster has always led to a system failure. Note that the cause of the disaster does not play a role here. Only state 3 satisfies this formula. The predefined level of service is tantamount to an intact system:

$$\texttt{service} \equiv \texttt{intact}. \tag{6.2}$$

This formula is satisfied by states 0,1 and 2. ◊

Recoverability. The definition of survivability requires the system "*to recover predefined service levels in a timely manner.*" CSRL until formulas allow us to formulate the time constraint: we can specify a maximum time allowed for recovery. Additionally, we can restrict the cost (reward) allowed for recovery: the system has to reach a service state before a given time t and before the accumulated reward exceeds y. The logic CSRL forces us to wrap it into a probability constraint. For a given time bound t, a reward bound y and a probability p, the corresponding CSRL until formula reads

$$\texttt{recoverability} \equiv \mathcal{P}_{\geq p}\left(\texttt{true}\,\mathcal{U}_{[0,y]}^{[0,t]}\texttt{service}\right). \tag{6.3}$$

Hence, a state is recoverable if the probability of reaching a service state before t units of time have elapsed and y units of reward are accumulated is at least p. If we want recoverability in any possible case we set $p = 1$.

Survivability. We now have defined CSRL specifications for all the ingredients that make up survivability. Survivability, finally, requires any disaster state to be also recoverable. This, again, can easily be phrased as a CSRL state formula:

$$\texttt{survivability} \equiv \texttt{disaster} \Rightarrow \texttt{recoverability}. \tag{6.4}$$

All non-disaster states satisfy the survivability property trivially (because of the implication). If this formula is also satisfied by all disaster states of a system, we say that the system is survivable. Note that the disaster formula is not bound to describe the "worst" states of the system. We might purposely ignore really bad states, for example, because we already know they can never be recoverable.

Example 29. For the illustrating example, the survivability formula becomes

$$\texttt{survivability} \equiv \texttt{broken} \Rightarrow \mathcal{P}_{\geq p}\left(\texttt{true}\,\mathcal{U}_{[0,y]}^{[0,t]}\texttt{intact}\right). \tag{6.5}$$

All states besides state 3 trivially satisfy the formula because they are not disaster states. In the following we focus on the single disaster state.

Figure 6.1 shows in a three-dimensional plot the probability that has to be compared with the probability bound p when checking the recoverability formula for time bound t and reward bound y. The numbers were computed using the uniformisation algorithm by Sericola (cf. Section 5.2).

For triples (t, y, p) on and below the surface, the survivability formula holds in the disaster state. All points above the curve represent time bound/probability pairs for which the system is not survivable. ◊

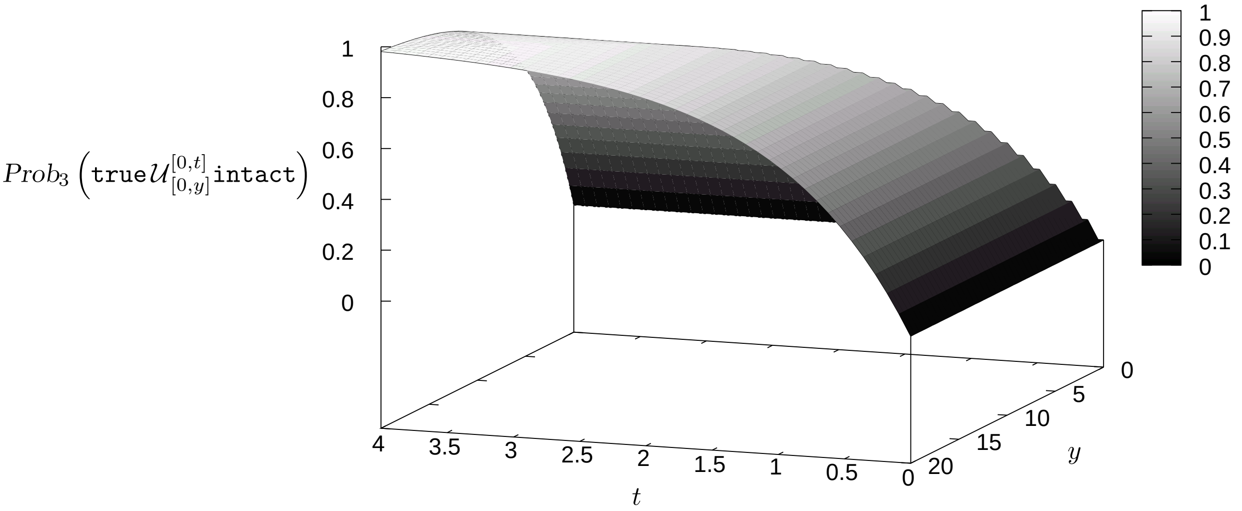


Figure 6.1: Probability of recovering from the single disaster state before time t and with an accumulated reward less than y: $Prob_3\left(\mathtt{true}\,\mathcal{U}^{[0,t]}_{[0,y]}\mathtt{intact}\right)$

6.2.2 ROOD models: long-run survivability

In a ROOD model, the occurrence of disasters is explicitly modelled by transitions leading into disaster states. Because of the stochastic nature of the model, they occur *randomly.*

We can check the survivability formula (6.4) for all states of a ROOD model as we do for GOOD models. For the case that not all states are survivable, ROOD models support a different notion of survivability. We define the *long-run survivability* as the fraction of time the system resides in survivable states when operating in steady-state. A system model complies with a level $l \in [0, 1]$ of long-run survivability if the accumulated steady-state probability of survivable states is at least l. Phrased as a CSRL formula this becomes

$$\mathtt{long\_run\_survivability} \equiv \mathcal{S}_{\geq l}(\mathtt{survivability}).$$

Note that ROOD models are a special type of availability models [108], and, as such, suitable for this purpose.

However, when following the ROOD approach a question arises how to determine an appropriate rate for the disaster transitions. Disasters are absolutely rare events, much more rare than single component failures, so it will often not be possible to accomplish measurements that provide an adequate rate. It is even questionable if it makes sense to introduce disaster transitions into the model at all. Furthermore, it is not obvious, how to introduce disaster transitions in case there are several disaster states. Even if we have found a reasonable rate for disasters, it will in general be very small in comparison to other rates in the system. This discrepancy imposes numerical difficulties when applying the model checking algorithms.

Employing ROOD models also foils the basic idea behind model checking. Model checking allows us to take an existing model of an arbitrary system, that reflects the mechanisms that should provide survivability, for example, redundancy. The survivability property is specified separately from the model, in terms of a formula in an appropriate logic. Automated tools then check the validity of the formula. We do not have to alter the model to support the measure. In contrast, a ROOD model is tailored specifically to support the evaluation of long-run survivability.

We think that the ROOD approach is not suitable for the evaluation of survivability in the presence of catastrophic disasters. Considering the difficulties in designing a suitable model we decide not to follow this approach any further and concentrate on GOOD models.

6.2.3 Different disasters & levels of service

The definition of survivability explicitly allows several predefined service levels and is also not restricted to one type of disaster.

One might easily imagine a scenario where different disasters have different impact on the system. For a "mild" disaster we want the system to return to full service in a very short period of time. For a severe disaster we could require recovery to a medium service level quickly but allow a much longer interval for full recovery. In the following we formalise a survivability specification where these aspects are taken into account.

Let *Disasters* be a set of CSRL formulas describing the different grades of disaster impact we are going to explore for a given model. The `disaster` formula (6.1) would be one element of such a set.

Let *Service_Levels* be a set of CSRL formulas describing different levels of service. Again, the `service` formula (6.2) would be one element of this set.

A *bound function* $B : \mathit{Disasters} \times \mathit{Service\_Levels} \rightarrow (\mathbb{R}_{\geq 0} \cup \{\infty\}) \times \mathbb{R}_{\geq 0} \cup \{\infty\}) \times [0, 1]$ indicates the time bound, the reward bound, and the probability bound for recovery from the different disasters to the different service levels. We might set the time allowed for recovery to 0. In this case the system must provide some service even in the presence of a disaster. Using the value ∞ (for the time/cost bound) indicates that we require the system to recover eventually but not within a specified time/cost horizon.

To complete it all, the tuple $(\mathit{Disasters}, \mathit{Service\_Levels}, B)$ comprises a *survivability specification*. Given such a survivability specification, each pair of disaster and service formula generates a CSRL survivability formula (6.4). A system is survivable according to the survivability specification if its states satisfy each of the generated survivability formulas. We show this use in the next section.

6.3 Case study: a replicated file system

To show the feasibility of our approach, we evaluate the survivability of a distributed file system with replicated data. The case study presented in this section appeared in [25]. Our model of this file system is inspired by the description of the Google file system [42]. Even though the Google search engine is not vitally important, we all appreciate the promptness of its service. A substantial part of the Google services is made possible through the underlying file system. The Google file system is a scalable distributed file system for large distributed data-intensive applications. It is especially designed to run on inexpensive commodity hardware for which failures are thought to be the norm rather than the exception. Remarkable is also its specialisation in the handling of very large (multiple GB) files.

In the following we describe the design of the file system. We construct a GSPN model (cf. Section 2.8) from which a MRM model is generated automatically. We do not define reward rates for the states of the model, the MRM is actually only a labelled CTMC. We define different disaster and service levels as CSRL formulas. Instead of checking just one survivability specification, we evaluate the conditions under which the file system survives different disasters. We thereby concentrate on the timely recovery and do not specify any cost (reward) constraints.

6.3.1 System description

Files are divided into evenly sized *chunks* of 64 MB. Several replicas (usually three) of these chunks reside on the disks of a collection of so-called *chunk servers*. A client interacts with the Google file system by first accessing the *single master* computer that keeps a hash table for the locations of all replicas of a chunk. In a second step, the client either reads one replica (usually the one residing on the closest chunk server) or initiates a chain of updates to all replicas. The data transfer is directly done between client and chunk servers, the master is not further involved.

The master monitors the chunk servers with regular *HeartBeat* messages. If it detects the failure of a chunk server, it will re-replicate the chunks that have been lost. Both, master and chunk servers, are designed to recover really fast if there is any software problem. A restart of the corresponding processes takes just a couple of seconds. In case of a severe master failure, another machine will soon take over. The chunk servers are cheap commodity PCs. The designers of the file system assumed that component failures occur frequently. A software restart is – as it was the case for the master – a matter of seconds. Replacing failed hardware is not urgent, since the file system takes care of the lost data.

6.3.2 GSPN Model

In the following we describe a GSPN modelling the Google file system. Our model focuses on the life cycle of a single chunk but also accounts for the load imposed by the entirety of all chunks. We took inspiration from the GSPN model of a replicated file system with voting in [9].

The *master* is modelled by the places and transitions in the upper part of Figure 6.2. Its places are prefixed with the uppercase letter M, its transitions with the lowercase letter m. The master is in working mode, if a token resides in place M_up. Transition m_fail indicates a failure of the master. Firing of immediate transition m_soft implies that the failure is to be repaired by a simple software restart, modelled by transition m_soft_repair. If transition m_hard fires, the hardware has failed and repair is modelled by transition m_hard_repair. If

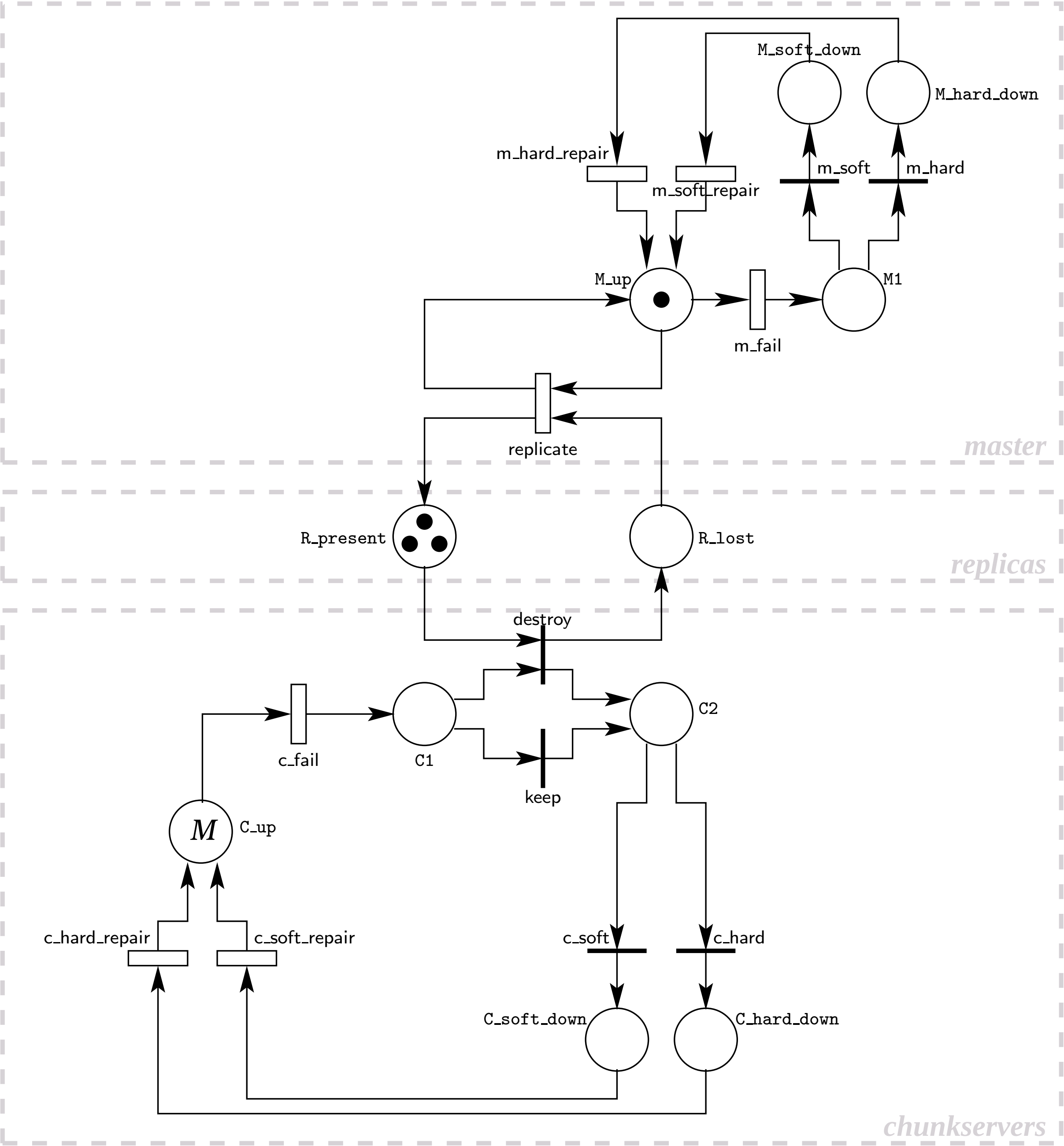


Figure 6.2: GSPN for the Google replicated file system

the master is up, transition replicate models the replication process of chunks that have previously resided on a now failed chunk server.

Tokens in the two places in the middle of Figure 6.2 represent the *replicas* of a single chunk. A token in place R_present stands for an existing replica on a working chunk server. Tokens in place R_lost are replicas that were destroyed during the failure of a chunk server. They are re-replicated by the master via the replicate transition.

transition	*rate* (min^{-1})	*probability*	*enabling condition*
m_fail	0.0005		
m_soft_repair	$\#\texttt{M\_soft\_down} \cdot 12$		
m_hard_repair	6.0		
c_fail	0.05		
c_soft_repair	$\#\texttt{C\_soft\_down} \cdot 12$		
c_hard_repair	1.0		
replicate	20.0 for $\#\texttt{R\_present} > 0$		$(\#\texttt{C\_up} > \#\texttt{R\_present})$ and
	2.0 for $\#\texttt{R\_present} = 0$		$(\#\texttt{C\_up} \cdot S \geq (\#\texttt{R\_present} + 1) \cdot N)$
m_soft		0.95	
m_hard		0.05	
c_soft		0.95	
c_hard		0.05	
destroy		$\frac{\#\texttt{R\_present}}{\#\texttt{C\_up}}$	
keep		$1 - \frac{\#\texttt{R\_present}}{\#\texttt{C\_up}}$	

Table 6.1: Rates, probabilities and enabling conditions of transitions

The lower part of the GSPN in Figure 6.2 represents the status of the *chunk servers*. The total number of chunk servers is M. Place C_up contains one token for each chunk server in working mode. Failure of chunk servers is modelled by transition c_fail. The immediate transitions destroy and keep form a probabilistic switch deciding whether the failing chunk server held one of the explicitly modelled replicas or not. The probabilities for these immediate transitions are determined by the ratio of the number of active replicas and the number of operational chunk servers:

$$p(\textsf{destroy}) = \frac{\#\texttt{R\_present}}{\#\texttt{C\_up}}, \quad p(\textsf{keep}) = 1 - \textsf{p}(\textsf{destroy}).$$

Note that the weighting probabilities are to be determined *before* transition c_fail fires because the intermediate marking (with a token in C1) is vanishing. Transitions c_soft and c_hard decide whether it has been a software or a hardware fault. Repair is modelled by transitions c_soft_repair and c_hard_repair, respectively. A software restart requires no further human interaction and is therefore performed simultaneously for all failed chunk servers (infinite server semantics). Hardware failures are solved one by one.

The restoration of replicas depends on several details. First of all, the master has to be in working mode. Having two replicas of the same chunk on one chunk server has no advantages, so the number of chunk servers must exceed the number of already existing replicas. Finally, we take into account that the explicitly modelled chunk is not the only one in the file system. We introduce two additional parameters of the system. The *storage capacity* S of a chunk server is the maximum number of chunks it can store on its disks. With N we denote the total number of chunks handled by the file system. The file system supports three replicas of every chunk if the number of unique chunk servers $M \geq 3N/S$. For the distinguished chunk we define a worst case scenario: the restoration of the kth replica is only possible, if there are already k replicas of all other chunks. Since we do not model explicitly the majority of the chunks, we approximate this behaviour by enabling the replicate transition only if

$$\#\texttt{C\_up} \cdot S \geq (\#\texttt{R\_present} + 1) \cdot N,$$

that is, if the available storage capacity is sufficient for $(\#\texttt{R\_present} + 1)$ replicas of all N chunks. Thus we avoid the explicit modelling of the status of all chunks and their replicas, which would lead to a state space of intractable size. The transition replicate has a marking-dependent rate: if there are still replicas on working servers, e.g., $\#\texttt{R\_present} > 0$, replication is done quickly. If the chunk was lost completely $(\#\texttt{R\_present} = 0)$, the chunk has to be rebuilt from external sources which takes a much longer time.

The rates and probabilities assigned to transitions (cf. Table 6.1) are all educated guesses made on the basis of [42].

The underlying labelled MRM The GSPN for the file system is described in the variant of CSPL that we briefly described in Section 2.8.

We now derive an explicit expression for the number of tangible states as a function of the number of chunk servers M. First considering only the places of the master part, there are three different markings. For the replica part, there are just four different markings. Concerning just the chunk-server places, there are $\frac{1}{2}(M+1)(M+2)$ different markings. However, not all of these comprise valid overall system states: markings where the number of working chunk servers is smaller than the number of existing replicas $(\#\texttt{C\_up} < \#\texttt{R\_present})$ are not generated. The total number of such markings is $18M+6$. Hence, the total number of states equals

$$|S| = 3 \cdot 4 \cdot \frac{1}{2}(M+1)(M+2) - (18M+6) = 6(M^2+1).$$

6.3.3 Survivability Specification

Disasters. We differentiate six levels of disaster, depending on the number of chunk servers failed and the type of failure.

- The master has failed in all disaster circumstances.

- If 25–50% of the chunk servers are affected we call it a *light* disaster, for 50–75% a *medium* and for 75–100% a *severe* disaster. The remaining chunk servers are working.
- We assume that there are either only software failures of each affected computer or only hardware failures.

We do not deal with mixed disasters, where both software and hardware failures have occurred. The event provoking a disaster is thought to affect either the hardware or the software of the file system. However, a mixed disaster is always worse than a pure hardware disaster. Results for hardware disasters consequently hold for mixed disasters as well.

Consider a “light software disaster”. The formula

$$\texttt{master\_software\_failure} \equiv (\#\texttt{M\_soft\_down} = 1)$$

holds if the master has failed because of a software problem. The formula

$$\begin{aligned}\texttt{c\_light\_software\_failure} \quad \equiv \quad & (\#\texttt{C\_soft\_down} > \frac{M}{4}) \\ & \wedge (\#\texttt{C\_soft\_down} \leq \frac{M}{2}) \\ & \wedge (\#\texttt{C\_hard\_down} = 0)\end{aligned}$$

holds in states where between 25% and 50% of the chunk servers have failed because of software reasons and the rest of the chunk servers is working properly. The corresponding disaster states are described as:

$$\texttt{light\_software\_disaster} \equiv \texttt{master\_software\_failure} \wedge \texttt{c\_light\_software\_failure}.$$

The other elements of the disaster set

$$\begin{aligned}\mathit{Disasters} \quad = \quad \{ & \texttt{light\_software\_disaster}, \texttt{medium\_software\_disaster}, \\ & \texttt{severe\_software\_disaster}, \texttt{light\_hardware\_disaster}, \\ & \texttt{medium\_hardware\_disaster}, \texttt{severe\_hardware\_disaster} \ \}\end{aligned}$$

are specified similarly.

Service levels. Concerning the single modelled chunk, we can only talk of service if there is at least one replica left. The number of existing replicas determines also the performance of the file system when handling a request. So, we define three levels of service, depending on the number of available replicas. The master has to be in working mode in either case. As CSRL formulas:

$$\begin{aligned}\texttt{master\_working} \quad & \equiv \quad (\#\texttt{M\_up} = 1) \\ \texttt{at\_least\_1\_replica} \quad & \equiv \quad (\#\texttt{R\_present} \geq 1) \\ \texttt{service\_level\_1} \quad & \equiv \quad \texttt{master\_working} \ \wedge \ \texttt{at\_least\_1\_replica}.\end{aligned}$$

The other service levels in

$$\mathit{Services} \;=\; \{\, \texttt{service\_level\_1}, \texttt{service\_level\_2}, \texttt{service\_level\_3} \,\}$$

are specified in the same manner.

6.3.4 Survivability Evaluation

In [42], a setup of the file system with 1000 chunk servers and a total of 300 TB disk space is mentioned, thus implying 300 GB storage per chunk server. We therefore choose to set the storage capacity of each chunk server to $S = 5000$ replicas, which amounts to $5000 \cdot 64$ MB/chunk $= 312.5$ GB per chunk server. The GSPN model for the file system with 1000 chunk servers generates a MRM with about 6 million states. For illustrative reasons we restrict the number of chunk servers to $M = 80$. The total number of chunks is taken to be $N = 100000$, so that in total $3 \cdot N = 300000$ replicas have to be stored. The MRM generated from the GSPN model with these parameters has 38406 states and 260885 transitions.

The six disaster types and three service levels generate a total of 18 survivability formulas as stated in (6.4). For a complete survivability specification (cf. Sec. 6.2.3), we need to identify the time, reward, and probability bounds $B(\cdot,\cdot)$ for each disaster/service pair.

We do not check a single survivability specification here, but rather evaluate in general what types of survivability specifications are satisfied. Since there are always multiple disaster states, for each disaster/service pair, we calculate the *minimum* probability over all disaster states to recover within time t (remember that we omit any reward constraints). Only if this minimum probability is above the specified probability threshold, all of the disaster states are survivable.

Figure 6.3 shows the minimum probability of recovering in time for different times t starting from a state that satisfies `disaster_soft_light`. The probabilities always stem from the same "worst" state such that the resulting curves are smooth. This need not be the case in general. Any survivability specification where the bound function B maps $(\texttt{disaster\_soft\_light}, \texttt{service\_level\_}i)$ to a time/probability pair (t, p) *below* the corresponding curve, is satisfied by the system (concerning only this disaster). If it is above the corresponding curve, the system is not survivable (concerning the considered disaster types).

Software failures of multiple chunk servers are repaired by restarting all affected computers simultaneously. The average time needed for a restart is only 5 seconds. Hence, the time to recover depends mainly on the master to return to service followed by the restoration of the replicas, and *not* on the restart of the chunk servers. As a consequence, the numerical results for `disaster_soft_medium` and `disaster_soft_severe` are almost identical to the ones for `disaster_soft_light`. We therefore omit a graph of the minimum probabilities of recovering in time for the medium and severe software disaster.

Figure 6.4 shows the minimum probability over all disaster states of recovering in time after the different hardware disasters, for the three service levels. Again, if the bound

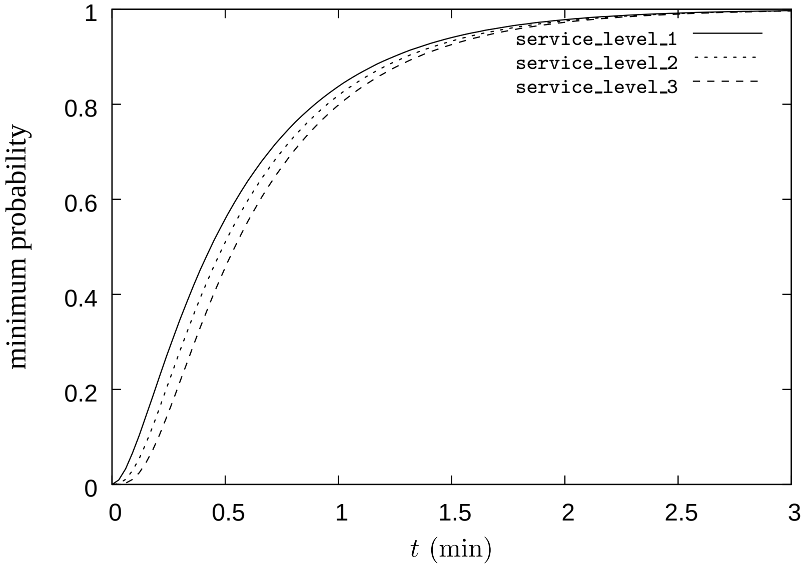


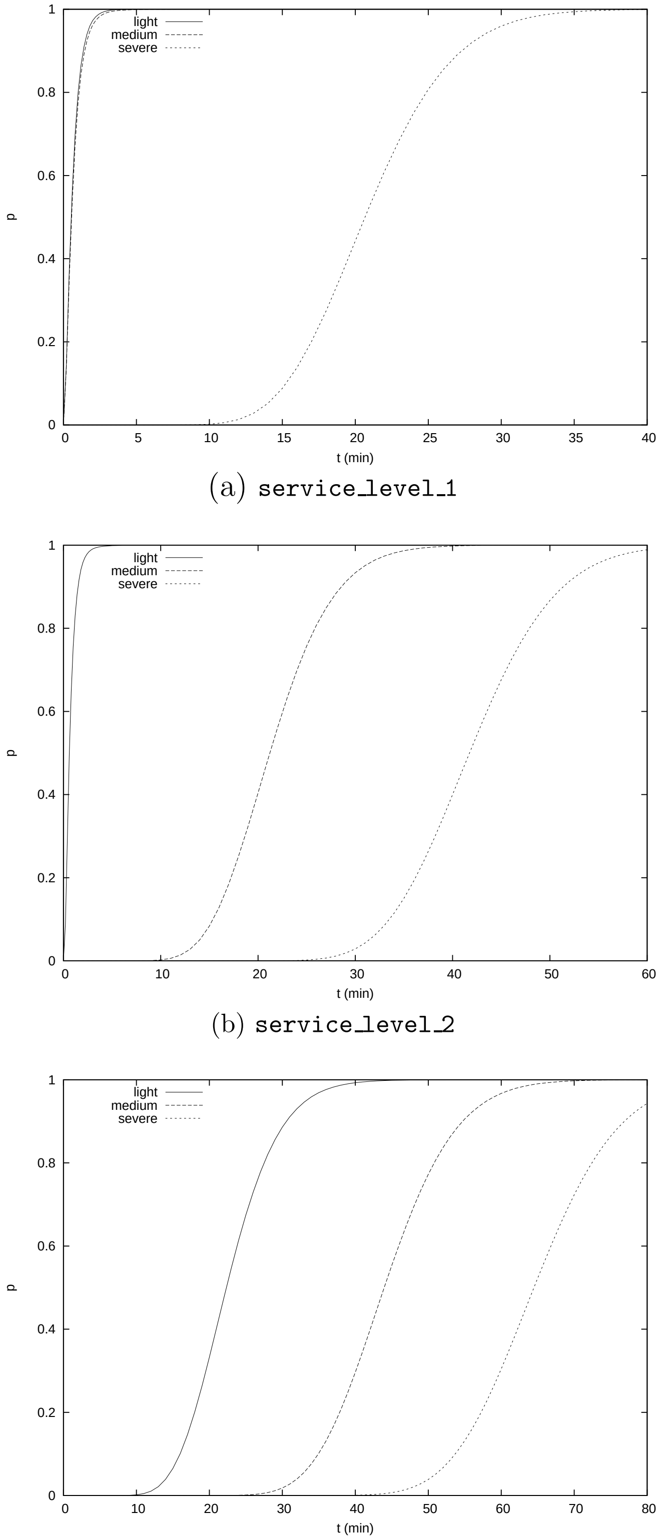
Figure 6.3: The minimum probability of recovering from `disaster_soft_light` within t minutes for the three different service levels

function B assigns values below the corresponding curve, all states satisfy this part of the survivability specification. For values above the curves, there are disaster states that do not satisfy the survivability specification, i.e., for these values the system is not survivable. Note that the t-axes in Figure 6.4 have different scales. This different scaling makes the effect of probability mass "moving to the right" when going to higher service levels (that is, going from Figure 6.4(a) to 6.4(c)) appearing less strong than it is!

An example of a bound function B for which the model of the Google file system is survivable is given in Table 6.2. Since recoverability is almost independent of the grade of a software disaster, time and probability bounds are chosen to be the same for one of the service levels and all three software disasters. All time/probability pairs are situated below the corresponding curves in Figure 6.3. For light and medium hardware disasters, recovery to service level 1 is likely to take place in a couple of minutes. For a severe hardware disaster, recovery takes substantially longer (more than 30 minutes, cf. Figure 6.4(a)). In case of service level 2 and 3, recovery from severe hardware disasters takes even more time (up to 80 minutes). The survivability specification then only requires the system to recover at all, but does not state a finite time bound.

6.4 Summary

Previously proposed intuitive definitions of survivability have focused on the ability to recover from disastrous circumstances. The logic CSRL gives us the means to formulate a precise survivability specification, thereby adhering to an intuitive definition. A system is called survivable if all of its states satisfy the survivability formula. The only work



(a) `service_level_1`

(b) `service_level_2`

(c) `service_level_3`

Figure 6.4: The minimum probability of recovering within t minutes from the different hardware disasters to the service levels 1, 2 and 3

	service_level_1	service_level_2	service_level_3
disaster_soft_light	$(1.6, \infty, 0.95)$	$(1.8, \infty, 0.95)$	$(2.0, \infty, 0.95)$
disaster_soft_medium	$(1.6, \infty, 0.95)$	$(1.8, \infty, 0.95)$	$(2.0, \infty, 0.95)$
disaster_soft_severe	$(1.6, \infty, 0.95)$	$(1.8, \infty, 0.95)$	$(2.0, \infty, 0.95)$
disaster_hard_light	$(1.2, \infty, 0.8)$	$(2.0, \infty, 0.8)$	$(30, \infty, 0.8)$
disaster_hard_medium	$(2.0, \infty, 0.8)$	$(30, \infty, 0.8)$	$(\infty, \infty, 1.0)$
disaster_hard_severe	$(25, \infty, 0.8)$	$(\infty, \infty, 1.0)$	$(\infty, \infty, 1.0)$

Table 6.2: Bound function $B : \mathit{Disasters} \times \mathit{Services} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\} \times \mathbb{R}_{\geq 0} \cup \{\infty\} \times [0, 1]$ for survivability specification

that has to be done manually is the design of a suitable model (we prefer GOOD models) and the specification of the ingredients of the survivability formula. CSRL model checking automatically accomplishes the actual survivability evaluation. We think it is worth using the CSRL framework, as it allows for an elegant, consistent, and comparable specification and evaluation of survivability.

Several other measures can be computed for MRM models. We relate a few of them to the survivability measure. Availability [103] is the ability of a system to deliver a predefined service at any instant of time. Long-run survivability is defined as the availability of survivable states. Survivable states are characterised by recoverability (return to service in a given time) after the occurrence of a disaster. Reliability [96] is the ability to deliver uninterrupted service for a specified time. This measure is more suitable to evaluate the defence mechanisms of a system against disasters. Dependability is a more general concept that includes, among others, availability and reliability [2]. One could view survivability as one possible additional constituent of dependability.

We showed the complete survivability evaluation process for a replicated file system that exhibits essentially the behaviour of the Google file system. We could check whether a given survivability specification is satisfied by the file system or not. We also showed how to indicate in detail the conditions under which the system is survivable.

Chapter 7

pathCSRL

Markov reward models are usually specified using a suitable high-level modelling formalism such as stochastic Petri nets [1], stochastic activity networks [75] or stochastic process algebra [54, 58]. When deriving the MRM from a high-level description not only the states have basic properties that can be expressed by atomic propositions but also the transitions carry a specific meaning. In stochastic process algebras, for example, the basic building blocks are the *actions* but also in stochastic Petri net-style formalisms one can attach names to transitions. It is therefore natural to consider MRMs that have action labels attached to transitions.

The logic CSRL does not include a mechanism to reason about transition labels. In this chapter we therefore introduce pathCSRL, a logic that allows both reasoning about state properties and action labels. It semantically extends the logic asCSL [4, 5] by adding constraints for the accumulated reward as CSRL does. Predecessors of asCSL were pathCSL [27] and SPDL [65] which already included the basic ingredients of asCSL. Logics that only allow for the reasoning about sequences of actions are aCSL and aCSL+ [56, 76].

The idea for the logics pathCSL and its reward-extension pathCSRL was born when comparing the power of CSRL model checking with the expressiveness of so-called path-based reward variables [80]. The logics pathCSL/pathCSRL combine the model checking approach for MRMs with the specification of path properties via finite automata.

Obal and Sanders introduced path-based reward variables as a means to specify complex system behaviours. Their approach helps to keep the model clean of information only necessary for the computation of a reward measure. Structural behaviour of a model is specified with a so-called *path automaton* which keeps track of the evolution of the model. On top of such a path automaton a (multiple) *reward structure* for the model is defined. Reward rates and impulse rewards are assigned according to the current state of the path automaton. Most often the rewards are only used to mark the occurrence of an event or the residence in a given type of state. The authors then combine the original model, the path-automaton and the automaton-based reward structure into a single model supporting the computation of the desired measure. To do so, they pair model states with automaton states in order to keep track of the evolution of the automaton. For the model checking of the logic pathCSRL we adopt the idea of combining the MRM with an automaton describing the property under study. In contrast to path-based reward variables, rewards are an integral part of MRMs. They can represent the availability of resources but also monetary costs or power consumption. With path-based reward variables they are mainly used to mark the completion of a given structural behaviour of the model. In our case such structural requirements are coded into a formula of the logic pathCSRL.

The chapter is further organised as follows. In Section 7.1 we briefly recapitulate the concept of regular expressions and finite automata which are needed for syntax and semantics of pathCSRL. Section 7.2 is dedicated to the extended model class, that is, MRMs with state and transition labels. Syntax and semantics of the logic pathCSRL are introduced in Section 7.3. Section 7.4 deals with the actual model checking of pathCSRL formulas. In Section 7.5 we evaluate some properties of the handover procedure in a cellular mobile communication network in order to illustrate the feasibility of pathCSRL model checking.

7.1 Regular expressions and finite automata [59]

Regular expressions and finite automata describe *regular languages*. A language L is a set of *words* over a finite alphabet (set) Σ. A word is a sequence of symbols of Σ, its length is given by the number of successive symbols. The *concatenation* of two languages L_1 and L_2 arises when concatenating two words from each of the two sets:

$$L_1L_2 = \{xy \mid x \in L_1, y \in L_2\}.$$

The powers of a language are given by subsequent concatenations. Let ε be the "empty" word of length 0.

$$L^0 = \{\varepsilon\},$$

$$L^{n+1} = LL^n.$$

The Kleene's star L^* of a language is the infinite union of all powers of L:

$$L^* = \bigcup_{n=0}^{\infty} L^n.$$

The set of all possible finite words over an alphabet Σ is denoted as Σ^*, each language over Σ is a subset of Σ^*. In the following, we use $a \in \Sigma$ to denote a symbol and $x, y \in L$ to denote words.

Regular expressions are used to describe a certain class of languages. The syntax of a regular expression α over an alphabet Σ is given by

$$\alpha ::= \varepsilon \mid a \in \Sigma \mid \alpha\alpha \mid \alpha + \alpha \mid \alpha^*.$$

The semantics of a regular expression α is its language $L(\alpha)$. We say that α accepts a word $x \in \Sigma^*$ if $x \in L(\alpha)$. The language of a regular expression is inductively defined as follows.

$$\begin{aligned}
x \in L(\varepsilon) &\quad\Longleftrightarrow\quad x = \varepsilon;\\
x \in L(a) &\quad\Longleftrightarrow\quad x = a \quad (a \in \Sigma);\\
x \in L(\alpha_1\alpha_2) &\quad\Longleftrightarrow\quad x \in L(\alpha_1)L(\alpha_2);\\
x \in L(\alpha_1 + \alpha_2) &\quad\Longleftrightarrow\quad x \in L(\alpha_1) \text{ or } x \in L(\alpha_2);\\
x \in L(\alpha^*) &\quad\Longleftrightarrow\quad x \in L(\alpha)^*.
\end{aligned}$$

Finite automata are another formalism that can be employed to describe regular languages. Formally a (nondeterministic) finite automaton is a tuple $\mathcal{A} = (Z, \Sigma, \delta, Z_0, Z_F)$ where

- Z is a finite set of *automaton states*,
- Σ is the alphabet,
- $\delta : Z \times \Sigma \to 2^Z$ is the *transition function*,
- Z_0 is the set of *initial states* and
- Z_F is the set of *final states*.

A finite automaton reads input symbols from its alphabet thereby changing state. The successor states are chosen nondeterministically from the set of successor states assigned by the transition function. The automaton *accepts* a word if it resides in a final state after consuming the word. The nondeterministic transition function may provide several successor states for an input symbol (or none at all). This behaviour is captured by the *extended transition function* $\hat{\delta}$ which takes a set of automaton states $Z' \subseteq Z$ and a word $x \in \Sigma^*$ as input and returns the complete set of states reachable from states in Z' when feeding x into the automaton. Formally, the empty word does not alter the set of automaton states.

$$\hat{\delta}(Z', \varepsilon) = Z'$$

For a word ya first all states z are collected that are reachable from Z' reading input y. Then the transition function δ is used to determine the reachable states when reading the additional symbol a.

$$\hat{\delta}(Z', ya) = \bigcup_{z \in \hat{\delta}(Z', y)} \delta(z, a).$$

An automaton *accepts* a word if it leaves the automaton in at least one final state, having started in the set of initial states. Whether a word x is in the language of an automaton $\mathcal{A}$ is consequently determined by

$$x \in L(\mathcal{A}) \quad\Longleftrightarrow\quad \hat{\delta}(Z_0, x) \cap Z_F \neq \varnothing.$$

Both regular expressions and finite automata recognise the class of regular languages. In particular, for each regular expression α there exists an automaton $\mathcal{A}$ that has the same language, and vice versa.

7.2 MRMs with state and transition labels

An MRM with state and transition labels has (in contrast to “normal” MRMs, cf. Chapter 2) an additional set of *actions* Act. Furthermore, the rate matrix $\mathbf{R}$ is now a function that maps action labels and pairs of states to non-negative real numbers: $\mathbf{R} : \mathsf{Act} \times S \times S \to \mathbb{R}_{\geq 0}$. We denote a transition from state s to state s' labelled with action a as $R^a_{ss'}$. It is possible to have several transitions between the same two states, as long as they are labelled with different actions.

States keep their labels drawn from the set of atomic propositions AP and assigned by the labelling function $L : S \to 2^{AP}$. A state- and transition-labelled MRM is then a tuple $\mathcal{M} = (AP, \mathsf{Act}, S, \mathbf{R}, L, \rho)$ Note that the transition labels are “hidden” in the rate matrix.

In the following — unless stated differently — we just write MRM when meaning a state- and transition-labelled MRM.

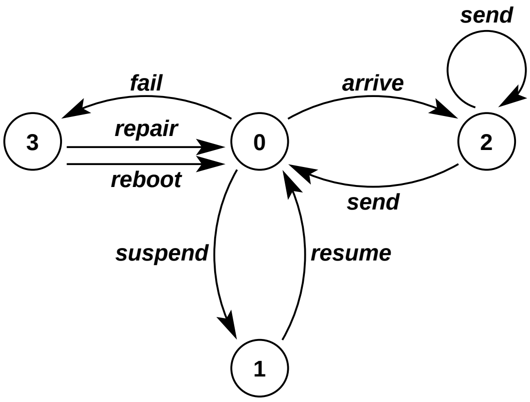


Figure 7.1: A state- and transition-labelled MRM $\mathcal{M}$ (Example 30)

Example 30. Figure 7.1 depicts a state- and transition-labelled variant of the example MRM used throughout the preceeding chapters (cf. Section 2.9). Transitions have action labels from the set $\mathsf{Act} = \{\mathsf{arrive}, \mathsf{send}, \mathsf{suspend}, \mathsf{resume}, \mathsf{fail}, \mathsf{repair}, \mathsf{reboot}\}$. It is possible to have more than one transition with the same action label; e.g., there are two transitions labelled send. Two different transitions lead from state 3 to state 0, modelling a software reboot or a hardware repair. They have different rates:

$$R^{\mathsf{reboot}}_{30} = 0.8 \qquad R^{\mathsf{repair}}_{30} = 0.2.$$

The rates of all other transitions, the reward rates and the state labelling is transferred from the state-labelled MRM defined in Section 2.9. ◊

The total exit rate E_s (cf. Section 2.1) is defined as the sum over all transition rates originating at state s, independent of the action:

$$E_s = \sum_{a \in \mathsf{Act}} \sum_{s' \in S} R^a_{ss'}.$$

The one-step probabilities $P^a_{ss'}$ are defined now per action:

$$P^a_{ss'} = \frac{R^a_{ss'}}{E_s}.$$

Note that for a single action a the one-step probability matrix $\mathbf{P}^a$ is not necessarily a probabilistic matrix.

A path in a state- and transition-labelled MRM is similar to a path in a state-labelled MRM (cf. Section 2.5). It additionally contains information about the actual transition taken between two states. Formally, an infinite path is an infinite sequence

$$\omega = (s_0, t_0, a_0)(s_1, t_1, a_1), \dots \in (S \times \mathbb{R}_{\geq 0} \times \mathsf{Act})^\omega,$$

where s_i is the ith state of the path, t_i is the residence time of this state and a_i is the label of the transition (the action) taken from state s_i to state s_{i+1}. The last state of a *finite path* must be an absorbing state. Since there is no further outgoing transition from an absorbing state, no action label can be indicated in this case. A finite path $\omega \in \Omega$ of length n is then a defined as $\omega = (s_0, t_0, a_0)(s_1, t_1, a_1) \cdots (s_n, \infty) \in (S \times \mathbb{R}_{\geq 0} \times \mathsf{Act})^n \times (S \times \{\infty\})$.

The probability measure on sets of paths in a state- and transition-labelled MRM is defined similarly as the one for ordinary MRMs. Cylinder sets are characterised by a state sequence, time intervals *and* the (action) labels of the transitions between the states. No outgoing action has to be specified for the last state. When calculating the probability measure (cf. Section 2.3) the one-step probability for the transition labelled with the indicated action is chosen.

Example 31. Consider the cylinder sets

$$C_1 = C((0,3,0), \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}, (\mathsf{fail}, \mathsf{reboot}))$$

and

$$C_2 = C((0,3,0), \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}, (\mathsf{fail}, \mathsf{repair})).$$

They both consist of all paths that start in state 0, switch to state 3 and return to state 0. But in C_1 the last transition must be labelled reboot whereas in C_2 the label has to be repair. Without transition labels they would not be distinguishable. Since $P^{\mathsf{reboot}}_{30} = 0.8 \neq 0.2 = P^{\mathsf{repair}}_{30}$ the two cylinder sets have different a probability measure:

$$P(C_1) = 0.1 \cdot 0.8 \cdot 1 \cdot 1 = 0.08 \qquad P(C_2) = 0.1 \cdot 0.2 \cdot 1 \cdot 1 = 0.02. \qquad \lozenge$$

7.3 Syntax and Semantics of pathCSRL

The syntax of pathCSRL state formulas is identical to the syntax of CSRL state formulas:

$$\Phi ::= \mathtt{true} \mid ap \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathcal{S}_{\bowtie}(\Phi) \mid \mathcal{P}_{\bowtie p}(\varphi),$$

where $ap \in AP$ is an atomic proposition, $\bowtie \in \{<, \leq, \geq, >\}$ is a comparison operator and $p \in [0,1]$ is a probability. The difference lies in the definition of the path formula φ that occurs within the probability operator. The CSRL next and until operators are replaced by a regular expression α equipped with a time interval $I \in \mathcal{I}$ and a reward interval $J \in \mathcal{I}$ and followed by a $\mathsf{pathCSRL}$ state formula Φ:

$$\varphi ::= \alpha_J^I \Phi.$$

The regular expressions allowed in $\mathsf{pathCSRL}$ path formulas are not arbitrary. Let F be a *finite* set of $\mathsf{pathCSRL}$ state formulas and recall that Act is a set of action labels. Regular expressions involved in path formulas have to be constructed over the alphabet

$$\Sigma = \mathsf{F} \times \mathsf{Act} = \{(\Phi, a) \mid \Phi \in \mathsf{F}, a \in \mathsf{Act}\}.$$

Furthermore, these regular expression are interpreted in a non-standard way. We do not simply derive the language $L(\alpha)$ but tweak the acceptance conditions such that so-called time-abstract *finite fragments* of paths in a MRM are recognised. The time-abstract finite fragment for two indices $0 \leq i \leq j$ of a possibly infinite path $\omega = (s_0, t_0, a_0)(s_1, t_1, a_1) \cdots$ with $|\omega| \geq j$, denoted as $\omega[i,j]$, consists of states i to j and the corresponding actions, but not the residence times:

$$\omega[i,j] = (s_i, a_i) \cdots (s_{j-1}, a_{j-1}) s_j \in (S \times \mathsf{Act})^{(j-i)} \times S.$$

The set of all time-abstract path fragments of an MRM $\mathcal{M}$ is denoted $\mathit{Frag}^{\mathcal{M}}$. The path fragments accepted by a regular expression α are denoted as $\mathit{Frag}^{\mathcal{M}}(\alpha)$. We omit $\mathcal{M}$ if it is clear from the context. A fragment is included in $\mathit{Frag}^{M}(\alpha)$ if its states satisfy the relevant $\mathsf{pathCSRL}$ state formulas of the regular expression and the action labels also conform.

The set $\mathit{Frag}(\alpha)$ is inductively defined as it was the case for the language $L(\alpha)$.

- Each one-state fragment is accepted by ε:

$$\omega[i,j] \in \mathit{Frag}(\varepsilon) \quad \Longleftrightarrow \quad i = j, \text{ that is, } \omega[i,j] = s_i.$$

- The alphabet symbol (Φ, a) accepts fragments $(s_i, t_i, a_i)s_{i+1}$ consisting of two states where the first state satisfies Φ and the transition from state s_i to state s_{i+1} is labelled with action a, that is, $a_i = a$:

$$\omega[i,j] \in \mathit{Frag}((\Phi, a)) \quad \Longleftrightarrow \quad j = i+1,\ s_i \models \Phi,\ a_i = a.$$

- The concatenation of two regular expressions accepts a path fragment that can be divided into two parts each of which is accepted by the respective sub-expression:

$$\omega[i,j] \in \mathit{Frag}(\alpha_1 \alpha_2) \quad \Longleftrightarrow \quad \exists k \in [i,j].\, (\omega[i,k] \in \mathit{Frag}(\alpha_1) \text{ and } \omega[k,j] \in \mathit{Frag}(\alpha_2))\,.$$

- The choice between two regular expressions accepts fragments that are recognised by either of the expressions:

$$\omega[i,j] \in \mathit{Frag}(\alpha_1 + \alpha_2) \quad \Longleftrightarrow \quad \omega[i,j] \in \mathit{Frag}(\alpha_1) \text{ or } \omega[i,j] \in \mathit{Frag}(\alpha_2).$$

- Finally, Kleene's star α^* accepts fragments that are recognised by some power of α:

$$\omega[i,j] \in \mathit{Frag}(\alpha^*) \quad \Longleftrightarrow \quad \exists n \in \mathbb{N}.\omega[i,j] \in \mathit{Frag}(\alpha^n).$$

If $A = \{a_1, \dots, a_k\} \subseteq \mathsf{Act}$ is a set of action labels, we use (Φ, A) as an abbreviation for the regular expression $((\Phi, a_1) + (\Phi, a_2) + \cdots + (\Phi, a_k))$, that is, any of the actions in A might be taken.

Example 32. Consider the following property in the context of the running example.

After some time without arriving packets the device processes a packet that requires two send *operations. The first failure afterwards occurs before sending any more packets and can be made up for by a* reboot.

A regular expression describing this property is given by

$$\begin{aligned}\alpha_1 = &(\mathtt{empty}, \mathsf{Act})^* \\ &(\mathtt{active}, \mathsf{send})(\mathtt{active}, \mathsf{send}) \\ &(\mathtt{empty}, \mathsf{Act}\backslash\mathsf{fail})^* \\ &(\mathtt{true}, \mathsf{fail})(\mathtt{true}, \mathsf{reboot}).\end{aligned}$$

Regular expressions and finite automata are equally expressive. We chose the more textual regular expressions for the syntax of the logic pathCSRL. However, we want to stress that we can equally well employ finite automata to describe path properties. In many cases automata might even be more convenient to specify. However, we must adapt the acceptance condition also for automata in order to define a set of path fragments. We do so by defining a different extended transition function $\tilde{\delta} : 2^Z \times \mathit{Frag} \to 2^Z$ which maps a set of automaton states and a path fragment to the set of automaton states reachable via this fragment.

Only actions can alter the automaton state. Hence, a path fragment of length 0 (without transition, just one state) leaves the automaton in the same set of states.

$$\tilde{\delta}(Z', \omega[i,j]) = Z', \text{ for } i = j.$$

If the fragment has at least length 1, firstly all states are determined that are reachable from Z' when feeding the fragment up to its last action into the automaton. Then all formulas are considered that are satisfied by the second last state (s_{j-1}) of the path fragment. Taking all symbols from the alphabet into account that consist of one of these formulas and the last action a_{j-1} of the fragment, the transition function δ is used to determine the set of reachable states.

$$\tilde{\delta}(Z', \omega[i,j]) = \bigcup_{z \in \tilde{\delta}(Z', \omega[i,j-1])} \bigcup_{\substack{\Phi \in \mathsf{F} \\ s_{j-1}(\omega) \models \Phi}} \delta(z, (\Phi, a_{j-1}(\omega))), \text{ for } i < j.$$

The set of path fragments $\mathit{Frag}^{\mathcal{M}}(\mathcal{A})$ accepted by the path automaton $\mathcal{A}$ consists of the paths that leave the automaton in at least one final state:

$$\omega[i,j] \in \mathit{Frag}(\mathcal{A}) \quad \iff \quad \tilde{\delta}(Z_0, \omega[i,j]) \cap Z_F \neq \varnothing.$$

Regular expressions and finite automata are equivalent when recognising languages. By the straightforward definition of the path fragments accepted by regular expressions and automata, we can deduce that

$$\mathcal{L}(\alpha) = \mathcal{L}(\mathcal{A}) \quad \Longrightarrow \quad \mathit{Frag}^{\mathcal{M}}_{\alpha} = \mathit{Frag}^{\mathcal{M}}_{\mathcal{A}}.$$

For pathCSRL it is consequently completely irrelevant whether a property is specified by a regular expression or a finite automaton. If a path property is given by a regular expression α we can easily construct a finite automaton $\mathcal{A}$ specifying the same property and vice versa.

Example 33. Finite automata can easily be represented as graphs: automaton states are the nodes, a directed edge labelled with a symbol $(\Phi, a) \in \Sigma$ is drawn between two states z and z' if $z' \in \delta(z, (\Phi, a))$. Initial states have an incoming edge without source, final states are shaded. We can use edge labels consisting of a formula and a *set* of actions to represent several parallel edges.

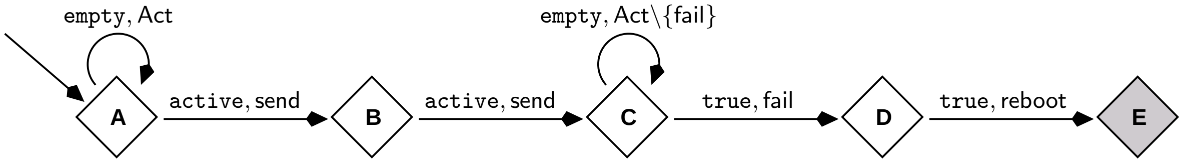


Figure 7.2: Automaton $\mathcal{A}_1$ specifying the same path property as regular expression α_1 (Example 32/33)

Figure 7.2 depicts an automaton $\mathcal{A}_1$ with state set $Z = \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}, \mathsf{E}\}$. It accepts the same path fragment as the regular expression α_1 does. $\Diamond$

The semantics of pathCSRL state formulas is identical to the semantics of CSRL state formulas (cf. Section 4.2). Recall that the semantics of the probability operator is based on the probability measure of all paths satisfying the inner path formula:

$$s \models \mathcal{P}_{\bowtie p}\left(\alpha^I_J \Psi\right) \quad \iff \quad \mathit{Prob}_s(\alpha^I_J \Psi) \bowtie p,$$

where

$$\mathit{Prob}_s(\alpha^I_J \Psi) = P_s\{\omega \in \Omega \mid \omega \models \alpha^I_J \Psi\}.$$

What is still missing is the definition of the satisfaction relation between paths and pathCSRL path formulas. The intuitive meaning of a path formula $\alpha^I_J\Psi$ is the following: a path ω satisfies the path formula $\alpha^I_J\Psi$ if there is an $n \in \mathbb{N}$ such that

(1) the prefix $\omega[0, n]$ is accepted by α,

(2) the *duration* $\sum_{i=0}^{n-1} t_i(\omega)$ of this prefix is in the time interval I,

(3) the accumulated reward $\sum_{i=0}^{n-1} y_i(\omega) \in J$ of this prefix is in J, and

(4) the last state $s_n(\omega)$ of the prefix is a Ψ-state.

More formally, we have

$$\omega \models \alpha^I_J\Psi \iff \exists n \in \mathbb{N}. \left(\omega[0,n] \in \mathit{Frag}(\alpha) \;\wedge\; \sum_{i=0}^{n-1} t_i(\omega) \in I \;\wedge\; \sum_{i=0}^{n-1} y_i(\omega) \in J \;\wedge\; s_n \models \Psi\right).$$

Example 34. A complete path formula involving the regular expression α_1 (Example 32) is

$$(\alpha_1)^{[0.1,0.5]}_{(2,10]}\texttt{idle}.$$

Consider the path fragment

$$0 \overset{0.01,\mathsf{suspend}}{\longrightarrow} 1 \overset{0.02,\mathsf{resume}}{\longrightarrow} 0 \overset{0.03,\mathsf{arrive}}{\longrightarrow} 2 \overset{0.01,\mathsf{send}}{\longrightarrow} 2 \overset{0.01,\mathsf{send}}{\longrightarrow} 0 \overset{0.02,\mathsf{fail}}{\longrightarrow} 3 \overset{0.08,\mathsf{reboot}}{\longrightarrow} 0.$$

It is accepted by α_1, its duration is $0.18 \in [0.1, 0.5]$ and the accumulated reward is $5.8 \in (2, 10]$. Its last state 0 is an `idle`-state. Any (infinite or finite) path starting with this fragment satisfies the path formula.

A complete state formula involving the given path formula is

$$\mathcal{P}_{<0.1}\left((\alpha_1)^{[0.1,0.5]}_{(2,10]}\texttt{idle}\right). \qquad \Diamond$$

CSRL and pathCSRL

An evident question is how the two logics pathCSRL and CSRL relate to each other. In [5] there is an extensive comparison of the expressive power of asCSL and CSL which can directly be transferred to pathCSRL and CSRL. We therefore only point out one important difference in the semantics of CSRL and pathCSRL path formulas. Consider the pathCSRL path formula

$$((\Phi, \mathsf{Act})^*)^I_J\Psi$$

which can be seen as a simulation of the CSRL path formula

$$\Phi\,\mathcal{U}^I_J\Psi.$$

But if $\inf I > 0$, there is a subtle difference in the semantics: for pathCSRL, a Ψ-state must be *entered* in the time interval I, while for CSRL the MRM must simply *reside* there at some time in I. The latter also allows to enter the Ψ-state before I starts and then stay there sufficiently long. Note that the Ψ-state then must be a Φ-state as well. Hence, if $\inf I > 0$ and there are states that satisfy both, Φ and Ψ, there is no way to simulate the CSRL until formula with a pathCSRL formula. If a formula restricts the actions to be taken in the course of a realisation of the MRM this naturally is also not expressible by a CSRL formula.

Example 35. Consider a path starting with

$$\omega_4 = 0 \overset{0.02,\mathsf{suspend}}{\longrightarrow} 1 \overset{0.04,\mathsf{resume}}{\longrightarrow} 0 \overset{0.02,\mathsf{fail}}{\longrightarrow} 3 \overset{0.06,\mathsf{reboot}}{\longrightarrow} \cdots$$

Ignoring the actions, it satisfies the CSRL path formula $\mathtt{empty}\,\mathcal{U}^{[0.1,0.5]}(\neg\mathtt{intact})$ but does not satisfy the pathCSRL path formula $((\mathtt{empty}, \mathsf{Act})^*)^{[0.1,0.5]}(\neg\mathtt{intact})$, because state 3 (the only $(\neg\mathtt{intact})$-state) is entered before $\inf I = 0.1$. ◇

7.4 The probability measure for path formulas

The path formula $\alpha^I_J\,\Psi$ defines a set of paths of an MRM. We must calculate the probability measure of these paths in order to model check the probability operator wrapped around the regular expression. In the following we assume that the path property is specified by an automaton. From Sections 7.1 and 7.3 we know that we can always construct an automaton accepting the same paths as a given regular expression.

Following the idea of Obal and Sanders [80] we combine the original MRM $\mathcal{M} = (AP, \mathsf{Act}, S, \mathbf{R}, L, \rho)$ and the automaton $\mathcal{A} = (Z, \Sigma, \delta, Z_0, Z_F)$ into the *product MRM*

$$\mathcal{M}\times\mathcal{A} = (AP^\times, \mathsf{Act}, S^\times, \mathbf{R}^\times, L^\times, \rho^\times), \tag{7.1}$$

which automatically encodes the structural property specified by the automaton in the MRM. The ingredients of the product MRM are defined as follows.

- The set of atomic propositions $AP^\times = AP \uplus \{\mathsf{accept}\}$ consists of the existing atomic propositions and a new atomic proposition $\mathtt{accept}$.
- The action set Act remains unchanged.
- The state space is given by $S^\times = S \times 2^Z$. A state is a pair of one MRM state and a set of automaton states. Note that the set of automaton states can also be empty.
- In the product MRM a transition labelled with a exists if this transition exists also between the corresponding states in the original MRM and if the sets of automaton

states correctly describe the evolution of the automaton induced by the current state and action. Hence, the rate matrix for each of the actions becomes

$$R^a_{(s,Z_1)(s',Z_2)} = \begin{cases} R^a_{ss'}, & Z_2 = \tilde{\delta}(Z_1, (s,t,a)s') \text{ for } t \in \mathbb{R}_{\geq 0}, \\ 0, & \text{otherwise.} \end{cases}$$

The matrices R^a constitute the complete rate matrix $\mathbf{R}^\times$.

- The labelling function $L^\times$ is transferred from the original MRM, additionally it assigns `accept` to states that have a final automaton state in the automaton part and satisfy formula Ψ:

$$L^\times((s,Z')) = \begin{cases} L(s) \uplus \{\mathsf{accept}\}, & Z' \cap Z_F \neq \varnothing \text{ and } s \models \Psi, \\ L(s), & \text{otherwise.} \end{cases}$$

- The reward vector adopts the reward rates from the original MRM:

$$\rho^\times_{(s,Z')} = \rho_s.$$

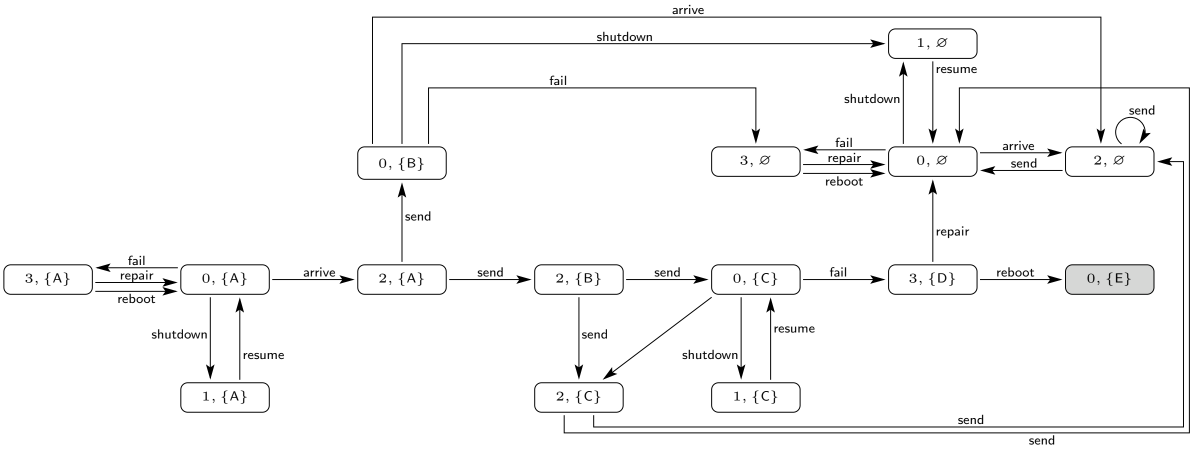


Figure 7.3: Product MRM $\mathcal{M} \times \mathcal{A}_1$ (Example 36)

Example 36. Figure 7.3 depicts the product MRM for the state- and transition-labelled MRM of Example 30 and the automaton $\mathcal{A}_1$ for the property described in Examples 32 and 33. We omit states that are not reachable from the states with initial automaton component $\{\mathsf{A}\}$.

The left part of the graph shows the four MRM states paired with the initial automaton state A. In the upper right corner are the MRM states in conjunction with the empty set of automaton states. Reaching such a state is tantamount to not satisfying the underlying path formula. In practice, these states can be merged into one single absorbing state. All other states of the product MRM might still lead to the `accept`-state $(0, \{\mathsf{E}\})$ which is shaded. Only here the automaton has reached the single final state E. ♢

Any path in the product MRM can uniquely be reduced to a path in the original MRM by omitting the automaton part of the states. Vice versa, each path in the original MRM expands uniquely to a path in the product MRM by tracking the evolution of the automaton. The probability measure of corresponding sets of paths in $\mathcal{M}$ and $\mathcal{M} \times \mathcal{A}$ is identical. Consequently, the probability measure of the paths satisfying the path property given by $\alpha_J^I \Psi$ can be calculated in either MRM:

$$Prob_s^{\mathcal{M}}(\alpha_J^I \Psi) = Prob_{<s,Z_0>}^{\mathcal{M}\times\mathcal{A}_\alpha}(\alpha_J^I \Psi).$$

To compute these probability measures, we proceed similarly as for CSRL until formulas. We construct a time- and reward-inhomogeneous MRM that supports the calculation of the measure via its joint distribution of state and accumulated reward.

The probability measure for $\alpha_J^I \Psi$ is the probability measure of all paths in $\mathcal{M} \times \mathcal{A}$ that *enter* an `accept`-state at a time in I and with an accumulated reward in J. We build an inhomogeneous MRM with two phases. As long as we are before $\inf I$ and the accumulated reward is below $\inf J$, the MRM operates in the first phase. Here, reaching an `accept`-state has no special meaning. In the second phase it is tantamount to satisfying the path formula. We can therefore make all `accept`-states absorbing in the second phase. In the first phase we use duplicates of the accept states: with this procedure we guarantee that only paths that really enter an `accept`-state within the given intervals contribute to the measure. Paths that end up in an `accept`-state before, are explicitely excluded.

The inhomogeneous MRM

$$\mathcal{M}\langle \alpha_J^I \Psi \rangle = (AP^\times, S^*, \mathbf{R}^*, L^\times, \rho^*) \tag{7.2}$$

is then formally defined as follows.

- The state space contains duplicates of all `accept`-states: $S^* = S^\times \uplus \overline{\mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept})}$ where $\overline{\mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept})} = \{\overline{(s, Z')} \mid (s, Z') \in \mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept})\}$.
- The rate matrix $\mathbf{R}^*$ has two versions, $\mathbf{R}^{(1)}$ and $\mathbf{R}^{(2)}$, one for each phase of the MRM. In both phases the rates for different actions are summed up. In the first phase, any transition to an `accept`-state is rerouted to the appropriate duplicate. The original `accept`-states are not connected. In the second phase, the `accept`-states are again reachable and made absorbing.

$$R_{ss'}^{(1)} = \begin{cases} \sum\limits_{a \in \mathsf{Act}} R_{s^*s'}^a, & s = \overline{s^*} \in \overline{\mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept})}, \\ \sum\limits_{a \in \mathsf{Act}} R_{ss^*}^a, & s' = \overline{s^*} \in \overline{\mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept})}, \\ 0, & s \text{ or } s' \in \mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept}), \\ \sum\limits_{a \in \mathsf{Act}} R_{ss'}^a, & \text{otherwise.} \end{cases}$$

In the second phase, the duplicates are not connected and all transitions emerging from `accept`-states are set to zero.

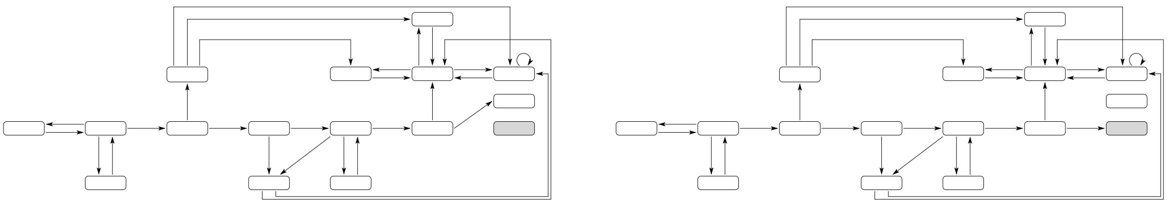
$$R^{(2)}_{ss'} = \begin{cases} 0, & s \text{ or } s' \in \overline{\mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept})}, \\[2ex] 0, & s \in \mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept}), \\[2ex] \displaystyle\sum_{a\in\mathsf{Act}} R^{a}_{ss'}, & \text{otherwise.} \end{cases}$$

- The reward rate vector also comes in two versions. The first version contains the reward rates of the product MRM, while the duplicates adopt the reward rates of their originals.

$$\rho^{(1)}_{s} = \begin{cases} \rho^{\times}_{s^*}, & s = \overline{s^*} \in \overline{\mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept})}, \\[2ex] \rho^{\times}_{s}, & \text{otherwise.} \end{cases}$$

In the second phase, the reward rates of the `accept`-states and their duplicates are set to zero.

$$\rho^{(2)}_{s} = \begin{cases} 0, & s \in \mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept}), \\[2ex] 0, & s = \overline{s^*} \in \overline{\mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept})}, \\[2ex] \rho^{\times}_{s}, & \text{otherwise.} \end{cases}$$



(a) phase 1

(b) phase 2

Figure 7.4: The two phases of the MRM $\mathcal{M}\left\langle(\alpha_1)^{[0.1,0.5]}_{(2,10]}\mathtt{idle}\right\rangle$ (Example 37)

Example 37. Figure 7.4 shows the two phases of the MRM $\mathcal{M}\left\langle(\alpha_1)^{[0.1,0.5]}_{(2,10]}\mathtt{idle}\right\rangle$. With the duplication of the single `accept`-state, it is guaranteed that, for example, paths starting with the fragment

$$(0,\{\mathsf{A}\}) \overset{0.01,\mathsf{arrive}}{\rightarrow} (2,\{\mathsf{A}\}) \overset{0.01,\mathsf{send}}{\rightarrow} (2,\{\mathsf{B}\}) \overset{0.01,\mathsf{send}}{\rightarrow} (0,\{\mathsf{C}\}) \overset{0.01,\mathsf{fail}}{\rightarrow} (3,\{\mathsf{D}\}) \overset{0.04,\mathsf{reboot}}{\rightarrow} \overline{(0,\{\mathsf{E}\})}.$$

are not included in the calculation of the probability measure. They are accepted by the regular expression α_1 but the duration is only $0.08 \notin [0.1, 0.5]$ and so they do not satisfy the path formula. ◊

The probability measure for $\alpha_J^I \Psi$ is reduced to the sum over all `accept`-states of the joint distribution of state and accumulated reward at the end of the restricting time and reward intervals.

Theorem 4.

$$Prob_s^M(\alpha_J^I \Psi) = Prob_{(s,Z_0)}^{\mathcal{M}\times\mathcal{A}}(\alpha_J^I \Psi) = \sum_{(s',Z')\in \mathsf{Sat}^{\mathcal{M}\times\mathcal{A}}(\mathtt{accept})} Y_{(s,Z_0)(s',Z')}^{\mathcal{M}\langle\alpha_J^I \Psi\rangle}(\sup I, \sup J).$$

Like for CSRL until formulas it depends on the precise structure of I and J whether really an inhomogeneous MRM has to be employed for the computation (cf. Section 4.7). Algorithms for the actual computation of the joint probabilities can be found in Chapter 5.

The complete procedure for model checking a path property in a state- and transition-labelled MRM $\mathcal{M}$ is then as follows.

(1) If the path property is given by a regular expression α, derive an equivalent finite automaton $\mathcal{A}$ (cf. Section 7.1).

(2) Construct the product MRM $\mathcal{M}\times\mathcal{A}$ (7.1).

(3) Using $\mathcal{M}\times\mathcal{A}$ as base, construct the inhomogeneous MRM $\mathcal{M}\langle\alpha_J^I \Psi\rangle$ (7.2).

(4) Compute $Prob_s^M(\alpha_J^I \Psi)$ using $\mathcal{M}\langle\alpha_J^I \Psi\rangle$ (Theorem 4).

(5) Return the satisfaction set of the $\mathsf{pathCSRL}$ formula $\mathcal{P}_{\bowtie p}\left(\alpha_J^I \Psi\right)$, as given by

$$\mathsf{Sat}^{\mathcal{M}}(\mathcal{P}_{\bowtie p}\left(\alpha_J^I \Psi\right)) = \left\{ s \mid Prob_s^{\mathcal{M}}(\alpha_I^J \Psi) \bowtie p \right\}.$$

Example 38. Steps (1)–(3) have already been performed in Examples 33, 36 and 37. Using the Markovian approximation (cf. Section 5.5) we obtain the vector containing the probability measures of the path formula for all starting states:

$$Prob_s((\alpha_1)_{(2,8]}^{[0.1,0.5]}\mathtt{idle}) = (0.0973, 0, 0614, 0.1347, 0.0422).$$

The satisfaction set of $\mathcal{P}_{<0.1}\left((\alpha_1)_{(2,8]}^{[0.1,0.5]}\mathtt{idle}\right)$ then equals

$$\mathsf{Sat}^{\mathcal{M}}\left(\mathcal{P}_{<0.1}\left((\alpha_1)_{(2,8]}^{[0.1,0.5]}\mathtt{idle}\right)\right) = \{0, 1, 3\} \qquad \diamond$$

The computation of the probability measure of a $\mathsf{pathCSRL}$ path formula is mapped to the computation of the joint distribution of state and accumulated reward in an inhomogeneous version of the so-called product MRM. In the worst case, the product MRM has $|S| \times 2^{|Z|}$ states. In practice, only the reachable states have to be generated and all sink states (with empty automaton part) can be merged into one absorbing state, which substantially reduces the size of the state space. The actual size of the product MRM depends heavily on the structure of the model and automaton.

7.5 Case study: handover in a cellular mobile communication network

We consider a scalable cellular mobile communication network. Each cell is ruled by a base station subsystem (BSS). We are especially interested in the behaviour of the system concerning a distinguished mobile radio station (MS) (also called the distinguished user) moving from one cell to another, thereby possibly triggering a so-called handover procedure. Handovers between the different cells are managed by the corresponding BSSs and the global mobile switching centre (MSC). Depending on the load of the MSC and the availability of channels at the BSSs, a handover might succeed or fail. The model is inspired by the description of the GSM handover procedure in [106] and [101].

In Section 7.5.1 we describe the system as a set of synchronising processes, namely the switching centre, the distinguished user's spatial movement and the user's functional behaviour. The properties of interest are expressed with `pathCSRL`-formulas involving regular expressions (Section 7.5.2). We show the corresponding finite automata and relate the size of the resulting product MRMs to the size of the original model in Section 7.5.3. Using transient analysis we compute the measures needed for model checking the formulas. Finally, in Section 7.5.4 the result of the model checking is briefly discussed.

This case study is an adapted version of the case study presented in [4, 5]. The model does not define reward rates for its states and also the formulas to be evaluated do not include reward bounds. In fact, it is a case study for `pathCSL`. For the probability measure of path formulas, only the transient probabilities have to be calculated (instead of the joint probability of state and accumulated reward).

7.5.1 The model

The distinguished MS is located in one of several GSM cells and is allowed to move between neighbouring cells. Each cell has hexagonal shape. Together, cells are arranged in such a way that they form again a hexagon. The size of this hexagon is described by the number of cells M that constitute one edge of it. Figure 7.5 depicts the cell topologies for $M = 2$ and $M = 3$. It also illustrates the unique cell identifiers. The parameter M is used for scaling the model; the complete hexagon has $3(M^2 - M) + 1$ cells.

We now describe the functional behaviour of the MS. When not being busy with a connection, the MS is `idle`. At any time, the MS can become `active`, meaning that it either accepts or establishes a (radio) connection. After a while, the connection can be terminated and the MS becomes idle again. If it moves from one cell to another while being active, the corresponding BSS commands a handover to the new cell from the MSC. If the handover is eventually completed, the MS returns to the `active` state (note that the connection is continued during the entire handover procedure). If the handover procedure is not completed in time, the connection is lost. The connection is then terminated (assume, that the distance to the former cell has become too large) and the MS is back in idle state.

Figure 7.6 shows a state-transition diagram for the distinguished MS. Transitions are

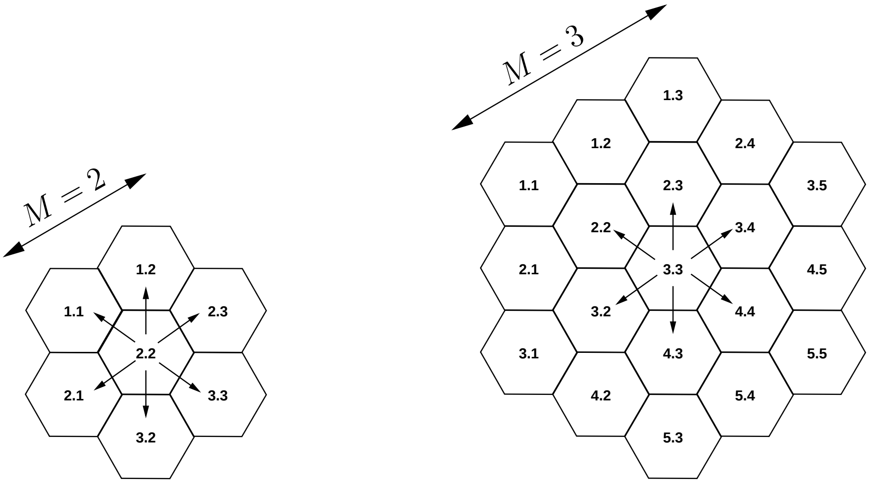


Figure 7.5: Hexagon of cells for $M = 2$ and $M = 3$

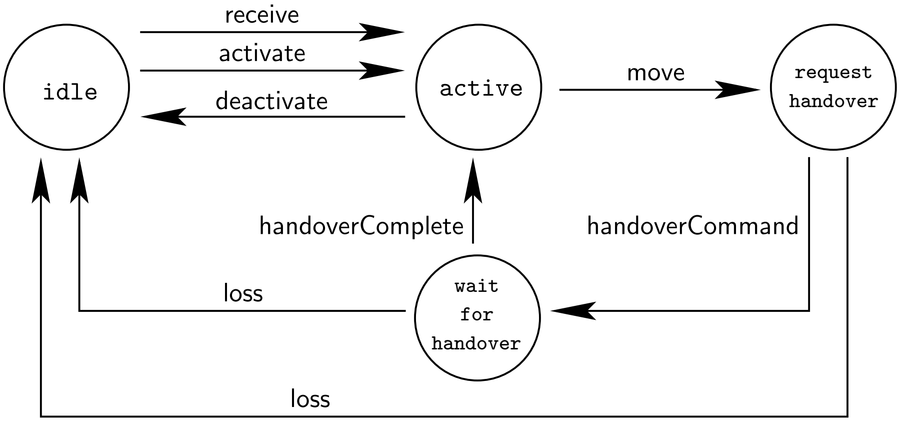


Figure 7.6: State machine for the distinguished MS behaviour

labelled with action names. Remember that in state- and transition-labelled MRMs we allow more than one transition between two states, as long as they are labelled with different actions. Such "parallel" (or coexisting) transitions (receive and activate) can be found between states `idle` and `active`. The move transition synchronises with the user's spatial movement whenever active.

The mobile services switching centre (MSC) is modelled by its load. It has either low, medium or high load. The time needed for the handover procedure depends on the current load. Under high load, the MSC does not process any request for handover at all. The MRM representing the complete system (MS position and behaviour and MSC) results from synchronising the processes for the MS position, the MS behaviour and the MSC.

Table 7.1 states the rates for transitions labelled with the given actions. Note that all numbers are educated guesses made on the basis of [105].

process	action	rate	description
MS position	move	0.02	from cell (i, j) to up to six neighbouring cells (equi-probable, on average, 50 seconds residence per cell)
MS behaviour	activate	0.0006250	average time between outbound connections is 1600 seconds
	receive	0.0003125	average time between inbound connections is 3200 seconds
	deactivate	0.008	connections last on average 125 seconds
	handoverCommand	1.0/0.5	for low/medium load of MSC, not available if MSC is blocking
	handoverComplete	1.0	connection has been transferred to new cell
	loss	0.1	might happen during handover procedure
MSC	lowtoMedium	0.5	from low load to medium load
	mediumToHigh	1.0	from medium load to high (blocking) load
	highToMedium	3.0	from high (blocking) load to medium load
	mediumToLow	1.0	from medium to low load

Table 7.1: Action labels and rates of the transitions of the cellular network model

7.5.2 pathCSRL-formulas

In this section we introduce several pathCSRLformulas which are constructed with two goals in mind. On the one hand they show the expressive power of pathCSRL. On the other hand they formalise interesting properties of the handover procedure. For each formula, the model checking procedure involves the construction of a product MRM. Note that all time intervals have zero as lower time bound, leading to a computation in a homogeneous MRM. We compute the transient distribution in these product MRMs and compare the resulting probabilities with the bounds given in the state formulas. A short interpretation of the result of the model checking completes the case study.

Move. The MS is always free to move from one cell to one of its neighbouring cells. We are interested, whether the probability of moving within the next two minutes (120 seconds) is at least 98%. A regular expression describing this behaviour is

$$\alpha_{\text{move}} = (\texttt{true}, \mathsf{Act}\backslash\{\mathsf{move}\})^*(\texttt{true}, \mathsf{move})$$

Firstly, arbitrary transitions may take place, as long as they are not labelled move. If then a move transition occurs, the model has shown the specified behaviour. Figure 7.7 shows a finite automaton for α_{move}. The complete pathCSRL formula becomes $\phi_{\text{move}} = \mathcal{P}_{>0.2}(\alpha_{\text{move}}^{[0,120]}\texttt{true})$. For this case, we can still state a CSRL formula which has the same meaning. Moving is equivalent to being in one cell at one moment and in another cell at the next moment. So the following CSRL path formula describes moving out of a cell (i, j)

within 120 seconds:

$$\varphi(i,j) = \mathtt{InCell}(i,j)\,\mathcal{U}^{[0,120]}(\neg \mathsf{InCell}(i,j)).$$

A CSRL formula equivalent to ϕ_{move} is then $\psi = \bigvee_{i,j} \mathcal{P}_{>0.2}(\varphi(i,j))$. It has to account for every cell the MS might be in. This makes the formula lengthy. We think that the pathCSRL version is much more readable and elegant.

Inbound connection. Next, we describe a pathCSRL state formula that is able to distinguish between coexisting transitions between two states. In our model both transitions activate and receive lead from state `idle` to state `active`. We can never decide whether a connection is inbound or outbound by just looking at state properties. Only the transitions tell us what is the case. The following regular expression resembles α_{move} but cannot be replaced by a CSRL path formula.

$$\alpha_{\text{inbound}} = (\mathtt{true}, \mathsf{Act}\backslash\{\mathsf{activate}, \mathsf{receive}\})^*(\mathtt{idle}, \mathsf{receive})$$

The corresponding finite automaton is given in Figure 7.8. With $\phi_{\text{inbound}} = \mathcal{P}_{\geq 0.3}(\alpha^{[0,2500]}_{\text{inbound}})$ we check whether the probability of receiving an inbound connection within the next 2500 seconds (without activating an outbound call) is at least 30%. Consider also the following pathCSRL state formula: $\phi_{\mathcal{S}} = \mathcal{S}_{\geq 0.85}(\phi_{\text{inbound}})$. It holds, if the steady-state probability of states that satisfy Φ_{inbound} is at least 85%.

Outdated handover. When the MS moves from one cell to the next, the BSS requests a handover to the new cell. However, the model (nor reality) does not prevent the MS from moving on to yet another cell. This behaviour is not explicitly visible in the model: here a handover is simply made to the cell the MS is in, no matter where it has been in between. In reality this type of movement could cause a problem. So, we would like to know whether the probability of such an outdated handover is lower than, say, 3.5%. As pathCSRL-formula, this becomes: $\phi_{\text{outdated}} = \mathcal{P}_{\leq 0.035}(\alpha^{[0,\infty]}_{\text{outdated}}\mathtt{true})$, with

$$\begin{aligned}
\alpha_{\text{outdated}} \;=\;& (\mathtt{active}, \mathsf{move}) && (7.3)\\
& (\mathtt{request} \vee \mathtt{wait}, \mathsf{Act}\backslash\{\mathsf{handoverComplete}, \mathsf{move}\})^* && (7.4)\\
& (\mathtt{request} \vee \mathtt{wait}, \mathsf{move}) && (7.5)
\end{aligned}$$

A move while the MS is active triggers a handover. Line (7.4) describe the system while being in the handover procedure. A move leads to an outdated handover (7.5). A finite automaton for the regular expression α_{outdated} is given in Figure 7.9.

Return without interruption. Assume that the MS initiates a connection while in the central cell (M, M). It is free to move between cells. We would like it to leave the central cell and to return within 10 minutes (600 seconds) without terminating or losing the connection. Is the probability for this scenario at least 10%? Coded into a pathCSRL-formula this reads $\phi_{\text{return}} = \mathcal{P}_{>0.1}(\alpha^{[0,600]}_{\text{return}}\texttt{inCentralCell})$, with

$$\begin{aligned}
\alpha_{\text{return}} &= (\texttt{inCentralCell}, \mathsf{activate}) && (7.6)\\
&\quad (\texttt{true}, \mathsf{Act}\backslash\{\mathsf{deactivate}, \mathsf{loss}\})^* && (7.7)\\
&\quad (\neg\texttt{inCentralCell}, \mathsf{Act}\backslash\{\mathsf{deactivate}, \mathsf{loss}\})^* && (7.8)
\end{aligned}$$

The regular expression firstly ensures that the user activates a connection while being in the central cell (7.6). Then the user can behave arbitrarily, as long as the connection is not ended via a deactivate or loss event (7.7). At some time, the user must have left the central cell and can again behave arbitrarily, as long the connection remains established (7.8). The return to the central cell (7.8) is specified by the trailing state formula in ϕ_{return}. Figure 7.10 shows a finite automaton for the regular expression α_{return}.

Ping-pong. Sometimes there are handovers from a cell (i, j) to a neighbouring cell (i', j') and back to cell (i, j) within a short time interval [106]. From a performance point of view this is not desirable since presumably the call could have remained in cell (i, j). During an active connection, is the probability of such a ping-pong handover to occur within 10 seconds at most 1%? As pathCSRL-formula: $\phi_{\text{pingpong}} = \mathcal{P}_{\leq 0.01}(\alpha^{[0,10]}_{\text{pingpong}}\texttt{true})$. A ping-pong between cell (i, j) and its neighbouring cells is described by the regular expression $\beta_{(i,j)}$:

$$\begin{aligned}
\beta_{(i,j)} = {} & (\texttt{inCell}(i,j) \wedge \texttt{active}, \mathsf{move})(\texttt{true}, B_1)^* \\
& \bigcup_{\substack{(i',j')\\ \text{neighbour of}\\ (i,j)}} \big((\texttt{inCell}(i',j'), \mathsf{handoverComplete})(\texttt{inCell}(i',j'), B_2)^*(\texttt{inCell}(i',j'), \mathsf{move})\big) \\
& (\texttt{true}, B_1)^*(\texttt{inCell}(i,j), \mathsf{handoverComplete}),
\end{aligned}$$

where $B_1 = \mathsf{Act}\backslash\{\mathsf{move}, \mathsf{loss}, \mathsf{handoverComplete}\}$ and $B_2 = \mathsf{Act}\backslash\{\mathsf{deactivate}, \mathsf{move}\}$. If (i, j) is an inner cell, that is, has all six neighbours, a finite automaton for the regular expression $\beta_{(i,j)}$ is given in Figure 7.11. All possible ping-pong situations are described by $\alpha_{\text{pingpong}} = \bigcup_{(i,j)} \beta_{(i,j)}$. The finite automaton for α_{pingpong} consists of one replica of the automaton in Figure 7.11 for each cell (i, j). It has an initial and a final state for each cell.

An implementation that performs the construction of the product MRM given a state- and transition-labelled MRM and a finite automaton has been implemented in C++. For the modelling and evaluation we employ a stochastic Petri net (SPN) model of the cellular system. All components of the systems are described by simple state machines, we therefore do not show their SPN representation here. The SPN is described in the extension of CSPL briefly described in Section 2.8. Path formulas including regular expressions are described

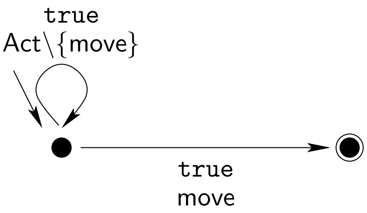


Figure 7.7: Finite automaton for α_{move}

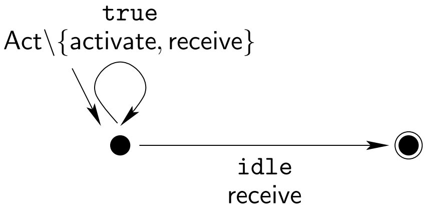


Figure 7.8: Finite automaton for $\alpha_{inbound}$

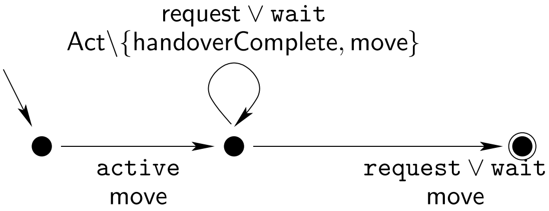


Figure 7.9: Finite automaton for $\alpha_{outdated}$

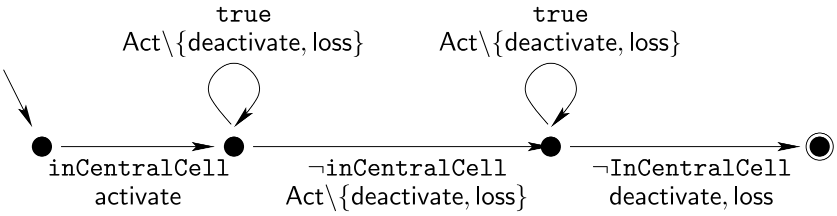


Figure 7.10: Finite automaton for α_{return}

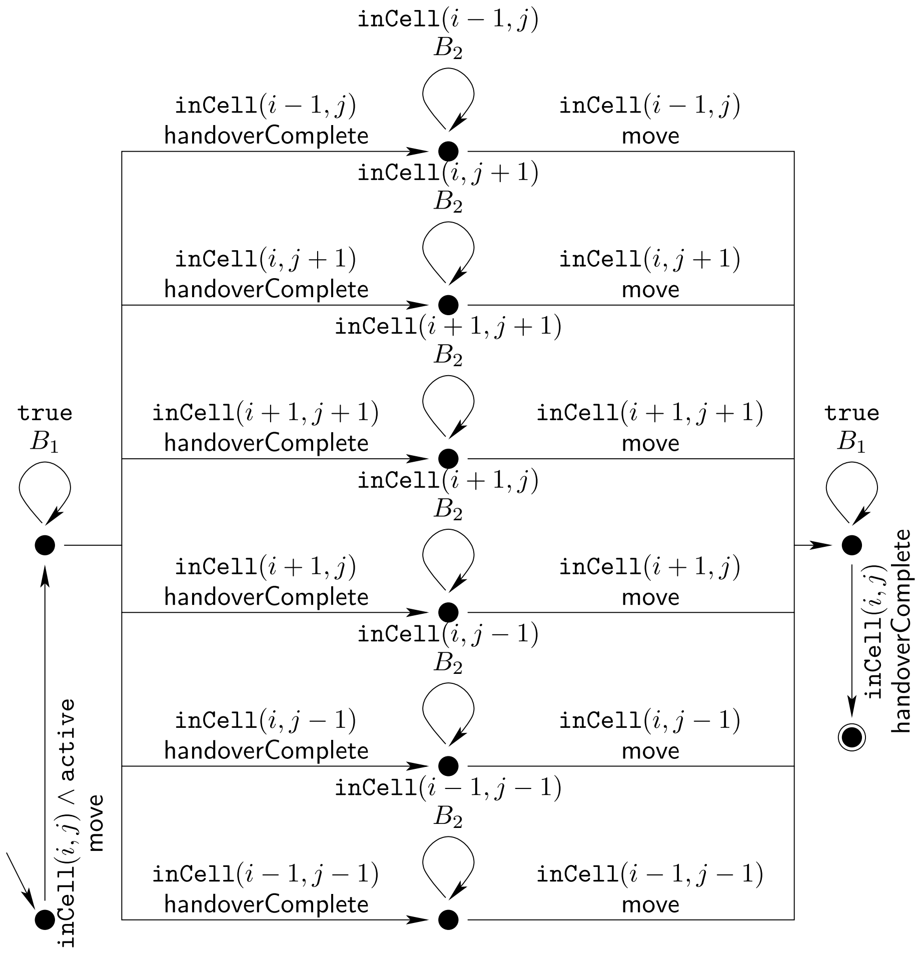


Figure 7.11: Finite automaton for $\beta_{(i,j)}$

directly via their corresponding finite automaton. The implementation takes the MRM and the automaton as input and computes the reduced product MRM, where only reachable states are generated and `accept`-states and reject states $\langle s, \emptyset \rangle$ with empty automaton part are merged into two special states.

Since the properties do not contain any reward bounds, uniformisation-based transient solution is used to actually perform the model checking of the formulas (besides the steady-state formula ϕ_S).

7.5.3 Product MRMs

Figure 7.12 shows the number of states and Figure 7.13 shows the number of transitions of the original model and of the product MRMs needed for the model checking procedure of the given pathCSRL-formulas as a function of the number M of cells per hexagon edge that ranges from 2 to 10.

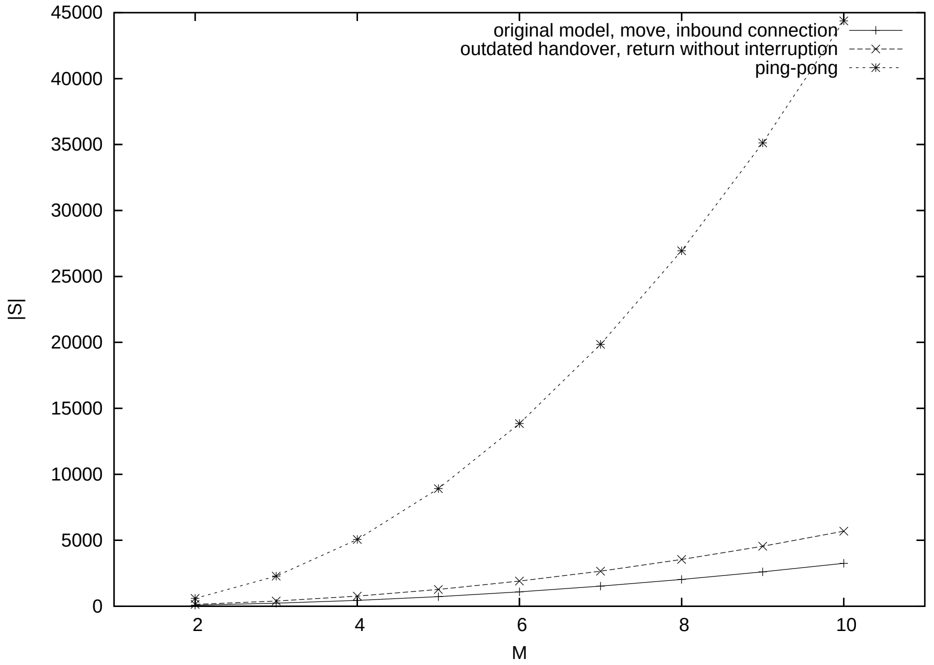


Figure 7.12: Size of state space in the original MRM and the product MRMs

The original model has 3252 states and 27900 transitions for $M = 10$. For all presented regular expressions, the number of states in the product MRM is equal to or larger than in the original model. This could be expected, since the generated state space is a subset of the Cartesian product $S \times 2^{|Z|}$. For the "move" and "inbound connection" regular expressions, the state space is the original state space plus the two special merge states for rejecting and accepting states. No additional states are created because after allowing arbitrary behaviour, the automata directly move into their final states.

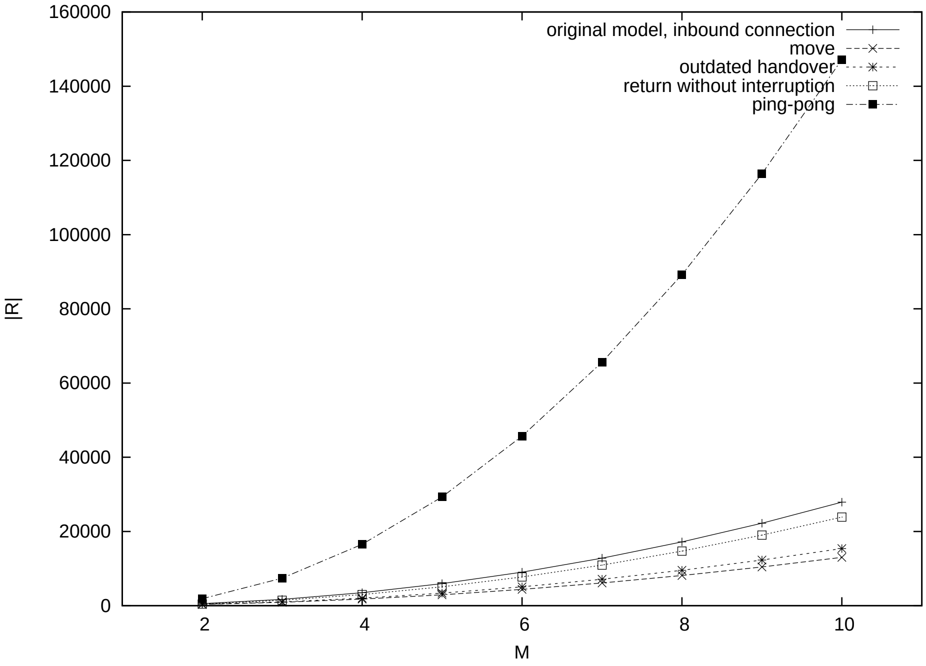


Figure 7.13: Number of transitions in the original MRM and the product MRMs

For the regular expressions of “outdated handover” and “return without interruption”, the size of the state space is roughly scaled by a factor 1.75. This is the result of having more than one automaton state to be visited before reaching a final state.

The largest state space is the one of the “ping-pong” property, it has more than 44000 states for $M = 10$. If we kept only those states from which the accept-state is reachable and merge the others into one absorbing state, the number of states could also become smaller than the original state space.

The regular expression for “inbound connection” leads to a product MRM in which there is exactly one transition for each transition in the original model. Transitions labelled receive now lead into the newly created accept state, transitions labelled activate lead into the reject state.

Even though the regular expression for “move” has exactly the same structure as the regular expression for “inbound connection”, it generates fewer transitions. This is because of the merging of accept-states into one state, which causes also all move-transitions (up to six) originating from a state to be aggregated into a single transition.

For the properties “outdated handover” and “return without interruption”, the number of transitions in the product MRM is smaller than in the original model as well. The corresponding automata are very restrictive, in the sense that in each state of the original model only a subset of all outgoing transitions is allowed. The automaton for “ping-pong” allows a broad range of different combinations of states and transitions. Consequently, it shows the largest growth in state space, and the number of transitions is much larger than in the original model (147175 for $M = 10$).

7.5.4 The model checking

Calculating the transient distribution of the "move" product MRMs reveals that all states of the model satisfy the "move" formula ϕ_{move} for all parameters M. This is not surprising, since the move transition exists at least once in every state and the mean time between two moves is $1/0.02 = 50$ seconds. This results in a sufficiently high probability of moving within two minutes.

The "inbound connection" formula ϕ_{inbound} is only satisfied by some of the states. That means that for some of the states the probability of having an inbound call within the next 2500 seconds is less than 30%. To see the satisfaction of ϕ_{inbound} on the long run, we consider the formula $\phi_{\mathcal{S}} = \mathcal{S}_{\geq 0.85}(\phi_{\text{inbound}})$. Since the graph of the model is strongly connected, the satisfaction set of $\phi_{\mathcal{S}}$ is either empty or equals the complete state space. For $M = 2, \ldots, 6$, no state satisfies the steady-state formula. The accumulated steady-state probability for all ϕ_{inbound}-states is smaller than 85%. For $M = 7, \ldots, 10$, all states satisfy formula $\phi_{\mathcal{S}}$.

Not all states satisfy the "outdated handover" formula. For states where the MS is `active` and the MSC has low load there are some states that do not fulfil ϕ_{outdated}. Looking at Figure 7.5, the cells are arranged in rings around the central cell (M, M). If the MS resides in one of the $M - 2$ inner rings (and is active and the MSC load is low), the probability of following the behaviour defined by α_{outdated} is above 3.5% and the state does not satisfy ϕ_{outdated}.

Formula ϕ_{return} ("return without interruption") is not valid in any state. For most of the states the probability of following a path specified by α_{return} is 0 anyway, because the MS is not in the central cell. But also for those states where the MS is in the central cell, the probability of returning with an ongoing call is too small to meet the bound.

Finally, in all states of the model the "pingpong" formula ϕ_{pingpong} holds. This is not surprising when comparing to the result of checking ϕ_{return}: already the less restrictive specification of returning to the same cell yields very low probabilities and the probability of having a pingpong handover is always below 1%.

7.6 Summary

Action labels for transitions arise naturally from the high-level description of MRMs. We introduced the logic pathCSRL that is based on CSRL but allows for reasoning about state properties and transition labels. Path properties are described by regular expressions or — equivalently — by finite automata, which permit the formulation of more complex behaviour than it is possible with CSRL. For the model checking, the model is combined with the automaton in order to obtain a MRM that supports the computation for the needed measure. We completed the presentation by checking some interesting properties of the handover procedure in a cellular mobile network.

Chapter 8

Conclusions

With this thesis we have covered in detail the concepts and algorithms needed for CSRL and pathCSRL model checking of Markov reward models. We have also shown that with CSRL model checking we can easily evaluate intricate system properties like survivability. For CSRL model checking, system properties are phrased as state formulas in the logic CSRL. Each CSRL state formula fits in one of three categories. Firstly, it can be a propositional formula that is constructed from Boolean operators which makes their checking straightforward. Secondly, it can be a steady-state formula. The model checking then relies on the computation of the steady-state distribution over the states of the MRM. Finally, a state formula can consist of the probability operator wrapped around a path formula. In this case, the model checking depends on the probability measure of all paths of the MRM that satisfy the path formula. For next formulas, the computation of the probability measure follows directly from the semantics. The real challenge is the computation of the probability measure for until formulas. Neither the semantics nor the alternative characterisation via integral equations lead to a suitable procedure. We have shown that we can map the computation of the probability measure of an until formula to the computation of the joint transient distribution of state and accumulated reward in a time- and reward-inhomogeneous MRM. This inhomogeneous MRM is derived from the MRM representing the real-world system under study. It also depends on the structure of the time interval and the reward interval. If the lower bounds of the intervals are zero, the derived MRM is actually homogeneous. Only if the lower bounds are larger than zero, a computation in an inhomogeneous MRM becomes necessary.

Some algorithms for the joint distribution of state and accumulated reward in homogeneous MRMs were already published in the CSRL context [28, 49]. We have compared five algorithms with respect to the accuracy of the results and with respect to scalability. For small models, we recommend the uniformisation algorithm published by Sericola because it gives results within a predefined error bound. However, it is too slow for larger models. We then prefer the Markovian approximation which allows the evaluation of MRMs with several thousands of states. New are the algorithms for inhomogeneous MRMs. Using them we are able to check CSRL until formulas with arbitrary time and reward intervals. Two algorithms are suitable for this purpose, namely, the discretisation algorithm and the Markovian approximation. Since no error bounds are known for these algorithms, the results have to be interpreted with great care.

To summarise, we have completed the theoretical foundation for the model checking of CSRL formulas, including until formulas with arbitrary time and reward intervals. We have also provided the numerical algorithms required for the actual model checking. Still missing is an integrated CSRL model checking tool. There exist software tools that facilitate the checking of the CSL part of CSRL, namely ETMCC [57], PRISM [66], GreatSPN [30], and the APNN toolbox [12]. Under development is MRMC, the Markov Reward Model Checker [61]. It includes some of the algorithms for the homogeneous case. A complete tool implementing optimised versions of all the algorithms discussed in this thesis does not yet exist. In the optimal case it would also contain the possibility to specify and check formulas of the logic pathCSRL because this extends the expressive power of CSRL substantially.

The specification of even complex system properties should be made easy as well. This can be done by providing templates that have only to be instantiated for the current model under study. The pattern derived for survivability is an example for this approach. Other performance and dependability properties are to be investigated and can then be integrated in a CSRL model checking tool.

A comprehensive and user-friendly software tool supporting CSRL model checking of MRMs would be a strong instrument for the model-based formal evaluation of important and complex properties of real-world systems. Only with such a tool available, CSRL model checking will prove its value in practice.

Appendix A

Proof of Theorem 3 (Section 4.6)

In Section 3.4 we derived a system of PDEs (3.10) for the joint distribution of state and accumulated reward for homogeneous MRMs. For nonhomogeneous MRMs, we extend the notion of the joint distribution and indicate the starting time. Thus

$$\Upsilon_{ss'}(t_1, t_2, y) = \Pr\left\{X_{t_2} = s', Y_{[t_1,t_2]} \leq y \mid X_{t_1} = s\right\}$$

is the probability of being in state s' at time t_2, having started at time t_1 in state s and accumulating a reward of at most y in the time interval $[t_1, t_2]$. For a *homogeneous* MRM $\mathcal{M}$ this joint probability is independent of the starting time:

$$\Upsilon^{\mathcal{M}}_{ss'}(t_1, t_1 + \tau, y) = \Upsilon^{\mathcal{M}}_{ss'}(t_1', t_1' + \tau, y) \text{ for all } t_1, t_1' \in \mathbb{R}_{\geq 0}.$$

The PDEs are similar to the ones derived in Section 3.4. We only have to replace $\mathbf{Q}$ by $\mathbf{Q}(t_1, y)$ and the diagonal matrix $\mathbf{D}$ containing the reward rates by $\mathbf{D}(t_1, y)$:

$$\frac{\partial \mathbf{\Upsilon}(t_1, t_2, y)}{\partial t_1} + \mathbf{D}(t_1, y) \cdot \frac{\partial \mathbf{\Upsilon}(t_1, t_2, y)}{\partial y} = \mathbf{Q}(t_1, y) \cdot \mathbf{\Upsilon}(t_1, t_2, y),$$

with initial values

$$\mathbf{\Upsilon}(t_2, t_2, y) = \begin{cases} 1, & s = s' \text{ and } y \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

Note that $t_s = \inf I^J_s$ is the time of the inhomogeneity in $\mathcal{M}\langle \Phi \,\mathcal{U}^I_J \Psi\rangle$ when residing completely in state s: both time and reward bound are exceeded for the first time at t_s. If we are interested in $\Upsilon_{ss'}(t_1, t_2, y)$ for $t_2 < t_s$, only the first phase of the MRM is used and the integral equation is roughly the same as for homogeneous MRMs (3.9) (adapted to the starting time t_1) with $\mathbf{Q}^{(1)}$ instead of $\mathbf{Q}$:

$$\begin{aligned} \Upsilon^{\mathcal{M}\langle \Phi \mathcal{U}^I_J \Psi\rangle}_{ss'}(t_1, t_2, y) = {} & e^{Q^{(1)}_{ss} t_2 - t_1} \Upsilon^{\mathcal{M}\langle \Phi \mathcal{U}^I_J \Psi\rangle}_{ss'}(t_2, t_2, y - \rho_s(t_2 - t_1)) \\ & + \int_0^{t_2 - t_1} \sum_{z \neq s} e^{Q^{(1)}_{ss} x} Q^{(1)}_{sz} \Upsilon^{\mathcal{M}\langle \Phi \mathcal{U}^I_J \Psi\rangle}_{zs'}(t_1 + x, t_2, y - \rho_s x)\, dx. \end{aligned}$$

Almost the same applies, if $t_1 \geq t_s$, we only have to use $\mathbf{Q}^{(2)}$.

If $t_1 < t_s \leq t_2$, we must take care of the inhomogeneity, as follows.

$$\begin{aligned}
&\Upsilon_{ss'}^{\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle}(t_1, t_2, y) = \\
&\quad e^{\left[Q_{ss}^{(1)}(t_s-t_1)+Q_{ss}^{(2)}(t_2-t_s)\right]}\Upsilon_{ss'}^{\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle}(t_2, t_2, y - \rho_s^{(1)}(t_s - t_1) - \rho_s^{(2)}(t_2 - t_s))) \\
&\quad + \int_0^{t_s-t_1} \sum_{z\neq s} e^{Q_{ss}^{(1)}x} Q_{sz}^{(1)} \cdot \Upsilon_{zs'}^{\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle}(t_1 + x, t_2, y - \rho_s^{(1)}x)\, dx \\
&\quad + \int_0^{t_2-t_s} \sum_{z\neq s} e^{\left[Q_{ss}^{(1)}(t_s-t_1)+Q_{ss}^{(2)}x\right]} Q_{sz}^{(2)} \Upsilon^{\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle}(t_s + x, t_2, y - \rho_s^{(1)}(t_s - t_1) - \rho_s^{(2)}x)\, dx.
\end{aligned} \tag{A.1}$$

The above expression roughly results by splitting the terms of (3.9) in two parts, one for each phase of the inhomogeneous MRM.

With this knowledge we slightly rephrase Theorem 3, taking into account the *start time* of the calculation. Note that the statement of the theorem remains unchanged since the considered start time is always zero.

Theorem 3. Let $\mathcal{M} = (AP, S, R, L, \rho)$ be a homogeneous MRM and let $\Phi\,\mathcal{U}_J^I\Psi$ be a CSRL until formula. For $s \in S$ and $x \in \mathbb{R}_{\geq 0}$, define

$$\mathit{initial}(s, x) = \begin{cases} \bar{s}, & s \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi) \text{ and } x < t_s \\ s, & \text{otherwise.} \end{cases}$$

Then

$$\mathit{Prob}_s^{\mathcal{M}}(\Phi\,\mathcal{U}_J^I\Psi) = \sum_{s'\in\mathsf{Sat}(\Psi)} \Upsilon_{\mathrm{initial}(s,0)s'}^{\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle}(0, \sup I, \sup J).$$

Proof. First note that if we shift the time and reward intervals in the until formula, we obtain the same joint distribution if we shift the start and end time accordingly:

$$\Upsilon_{ss'}^{\mathcal{M}\langle\Phi\,\mathcal{U}_J^I\Psi\rangle}(x, t, y) = \Upsilon_{ss'}^{\mathcal{M}\left\langle\Phi\,\mathcal{U}_{J\ominus\rho_s x}^{I\ominus x}\Psi\right\rangle}(0, t - x, y - \rho_s x). \tag{A.2}$$

We now show that the defining integral equations for

$$\mathit{Prob}_s^{\mathcal{M}}(\Phi\,\mathcal{U}_{[\inf J,y]}^{[\inf I,t]}\Psi)$$

and

$$\sum_{s'\in\mathsf{Sat}(\Psi)} \Upsilon_{\mathrm{initial}(s)s'}^{\mathcal{M}\left\langle\Phi\,\mathcal{U}_{[\inf J,y]}^{[\inf I,t]}\Psi\right\rangle}(0, t, y) \qquad \square$$

are equal for $t, y \geq 0$. Their solution then leads to the same measure. We distinguish between the five cases introduced for the characterisation of the probability measure in Section 4.6.1.

i) If state $s \models \Psi$ and $\inf I_s^J = 0$, the state s is absorbing in the MRM $\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi \right\rangle$ for all $t, y \geq 0$. Thus, the only state reachable from s is s itself. The reward rate ρ_s is 0 and so the accumulated reward by time t having started in s at time 0 is also $0 = \inf J$. The initial state $\mathrm{initial}(s,0)$ is s itself. Hence,

$$\begin{aligned}\sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi \right\rangle}_{\mathrm{initial}(s,0)s'}(0,t,y) &= \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi \right\rangle}_{ss}(0,t,y)\\ &= 1 = \mathit{Prob}^{\mathcal{M}}(s, \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi).\end{aligned}$$

ii) If $s \models \neg\Phi \wedge \Psi$ and $\inf I_s^J > 0$, the initial state is $\mathrm{initial}(s,0) = \bar{s}$. The state $\bar{s}$ is not a Ψ-state and it is absorbing, so no other state is reachable. We obtain an empty sum which equals 0:

$$\begin{aligned}\sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi \right\rangle}_{\mathrm{initial}(s,0)s'}(0,t,y) &= \sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi \right\rangle}_{\bar{s}s'}(0,t,y)\\ &= 0 = \mathit{Prob}^{\mathcal{M}}(s, \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi).\end{aligned}$$

Here we really need the definition of the initial states, otherwise the probability would not be zero.

iii) If state $s \models \neg\Phi \wedge \neg\Psi$, it is absorbing for all $t, y \geq 0$. Hence, no Ψ-state is reachable from state s and so the probability of reaching a Ψ-state is always 0.

$$\begin{aligned}\sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi \right\rangle}_{\mathrm{initial}(s)s'}(0,t,y) &= \sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi \right\rangle}_{ss'}(0,t,y)\\ &= 0 = \mathit{Prob}^{\mathcal{M}}(s, \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi).\end{aligned}$$

iv) For a state s with $s \models \Phi \wedge \neg\Psi$ we have to apply (A.1):

$$\begin{aligned}&\sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi \right\rangle}_{\mathrm{initial}(s)s'}(t,t,y)\\ &= \underbrace{\sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} e^{\left[Q^{(1)}_{ss}t_s + Q^{(2)}_{ss}(t-t_s)\right]}\Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi \right\rangle}_{ss'}(t,t,y-\rho^{(1)}_s t_s - \rho^{(2)}_s(t-t_s)))}_{=0}\\ &+ \sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t_s} \sum_{z\neq s} e^{Q^{(1)}_{ss}x} Q^{(1)}_{sz} \cdot \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi \right\rangle}_{zs'}(x,t,y-\rho^{(1)}_s x)\,dx\\ &+ \sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t-t_s} \sum_{z\neq s} e^{\left[Q^{(1)}_{ss}t_s + Q^{(2)}_{ss}x\right]} Q^{(2)}_{sz} \cdot \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi \right\rangle}(t_s + x, t, y - \rho^{(1)}_s t_s - \rho^{(2)}_s x)\,dx\end{aligned}$$

The first term is 0 because state s itself is not contained in the set $\mathsf{Sat}(\Psi)$, but only in this case $\Upsilon_{ss}(t,t,\cdot)$ could be $\neq 0$. Furthermore, $\rho_s^{(1)} = \rho_s^{(2)} = \rho_s$ and $Q_{ss}^{(1)} = Q_{ss}^{(2)} = Q_{ss}$.

$$= \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t_s} \sum_{z \neq s} e^{Q_{ss}x} Q_{sz}^{(1)} \cdot \Upsilon_{zs'}^{\mathcal{M}\left\langle \Phi \mathcal{U}_{[\inf J,y]}^{[\inf I,t]} \Psi \right\rangle}(x,t,y-\rho_s x)\,dx$$

$$+ \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t-t_s} \sum_{z \neq s} e^{Q_{ss}(t_s+x)} Q_{sz}^{(2)} \cdot \Upsilon^{\mathcal{M}\left\langle \Phi \mathcal{U}_{[\inf J,y]}^{[\inf I,t]} \Psi \right\rangle}(t_s+x,t,y-\rho_s(t_s+x))\,dx$$

In the following we distinguish between different types of states z. If $z \notin \mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi)$ and $z \notin \overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi)}$, $Q_{sz}^{(1)} = Q_{sz}^{(2)} = Q_{sz}$ and we can combine the two integrals. The states $z \in \overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi)}$ are disconnected in the second phase, so we keep only the first integral. The states $z \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi)$ are disconnected in the first phase, so we keep only the second integral.

$$= \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t} \sum_{\substack{z \neq s \\ z \notin \mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi) \\ z \notin \overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)}}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon_{zs'}^{\mathcal{M}\left\langle \Phi \mathcal{U}_{[\inf J,y]}^{[\inf I,t]} \Psi \right\rangle}(x,t,y-\rho_s x)\,dx$$

$$+ \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t_s} \sum_{\substack{z \neq s \\ z \in \overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)}}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon_{\bar{z}s'}^{\mathcal{M}\left\langle \Phi \mathcal{U}_{[\inf J,y]}^{[\inf I,t]} \Psi \right\rangle}(x,t,y-\rho_s x)\,dx$$

$$+ \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_{t_s}^{t} \sum_{\substack{z \neq s \\ z \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon_{zs'}^{\mathcal{M}\left\langle \Phi \mathcal{U}_{[\inf J,y]}^{[\inf I,t]} \Psi \right\rangle}(x,t,y-\rho_s x)\,dx$$

Now we switch from z to initial(z,x). This does not change the first and the third integral, in the second integral we now have to consider states in the set $\mathsf{Sat}^{\mathcal{M}}(\neg\Phi \wedge \Psi)$.

$$= \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t} \sum_{\substack{z \neq s \\ z \notin \mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi) \\ z \notin \overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)}}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon_{\mathrm{initial}(z,x)s'}^{\mathcal{M}\left\langle \Phi \mathcal{U}_{[\inf J,y]}^{[\inf I,t]} \Psi \right\rangle}(x,t,y-\rho_s x)\,dx$$

$$+ \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t_s} \sum_{\substack{z \neq s \\ z \in \overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)}}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon_{\mathrm{initial}(z,x)s'}^{\mathcal{M}\left\langle \Phi \mathcal{U}_{[\inf J,y]}^{[\inf I,t]} \Psi \right\rangle}(x,t,y-\rho_s x)\,dx$$

$$+ \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_{t_s}^{t} \sum_{\substack{z \neq s \\ z \in \mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon_{\mathrm{initial}(z,x)s'}^{\mathcal{M}\left\langle \Phi \mathcal{U}_{[\inf J,y]}^{[\inf I,t]} \Psi \right\rangle}(x,t,y-\rho_s x)\,dx$$

The three integrals can now be combined into a single one.

$$= \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^t \sum_{\substack{z \neq s \\ z \notin \mathsf{Sat}(\neg\Phi\wedge\Psi)}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}_{\text{initial}(z,x)s'}(x,t,y-\rho_s x)\,dx$$

We push the sum over Ψ-states under the integral and use the shift of starting time (A.2).

$$= \int_0^t \sum_{\substack{z \neq s \\ z \notin \mathsf{Sat}(\neg\Phi\wedge\Psi)}} e^{Q_{ss}x} Q_{sz} \cdot \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]\ominus x}_{[\inf J,y]\ominus\rho_s x}\Psi\right\rangle}_{\text{initial}(z,x)s'}(0,t-x,y-\rho_s x)\,dx$$

Finally we apply an induction argument.

$$= \int_0^t \sum_{\substack{z \neq s \\ z \notin \mathsf{Sat}(\neg\Phi\wedge\Psi)}} e^{Q_{ss}x} Q_{sz} \cdot Prob_z(\Phi\,\mathcal{U}^{[\inf I,t]\ominus x}_{[\inf J,y]\ominus\rho_s x}\Psi)\,dx$$
$$= Prob_s(\Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi).$$

v) The integral equation for a state s with $s \models \Phi\wedge\Psi$ is also affected by the inhomogeneity of the MRM $\mathcal{M}\langle\Phi\,\mathcal{U}^I_J\Psi\rangle$.

$$\sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}_{ss'}(0,t,y)$$
$$= \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} e^{\left[Q^{(1)}_{ss}t_s + Q^{(2)}_{ss}(t-t_s)\right]} \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}_{ss'}(t,t,y-\rho^{(1)}_s t_s - \rho^{(2)}_s(t-t_s)))$$
$$+ \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t_s} \sum_{z\neq s} e^{Q^{(1)}_{ss}x} Q^{(1)}_{sz} \cdot \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}_{zs'}(x,t,y-\rho^{(1)}_s x)\,dx$$
$$+ \sum_{s' \in \mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t-t_s} \sum_{z\neq s} e^{\left[Q^{(1)}_{ss}t_s + Q^{(2)}_{ss}x\right]} \underbrace{Q^{(2)}_{sz}}_{=0} \cdot \Upsilon^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}(t_s+x,t,y-\rho^{(1)}_s t_s - \rho^{(2)}_s x)\,dx$$

Only s itself is reachable in no time and so it is the only summand to be considered in the first term. The diagonal entry $Q^{(2)}_{ss} = 0$ and the reward rate $\rho^{(2)}_s = 0$ because s is absorbing in the second phase. Since $y > \inf J$ we have that $\rho_s t_s < y$ and the first term reduces to $e^{Q_{ss}t_s}$. The following argumentation is identical to the one of case iv)

$$
\begin{aligned}
&= e^{Q_{ss}t_s} \underbrace{\Upsilon_{ss}^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}(t,t,y-\rho_s t_s)}_{=1} \\
&+ \sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t_s} \sum_{z\neq s} e^{Q_{ss}x} Q^{(1)}_{sz} \cdot \Upsilon_{zs'}^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}(x,t,y-\rho_s x)\,dx \\
&= e^{Q_{ss}t_s} + \sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t_s} \sum_{\substack{z\neq s\\ z\notin\mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)\\ z\notin\overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)}}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon_{zs'}^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}(x,t,y-\rho_s x)\,dx \\
&+ \sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t_s} \sum_{\substack{z\neq s\\ z\in\overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)}}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon_{\bar{z}s'}^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}(x,t,y-\rho_s x)\,dx \\
&= e^{Q_{ss}t} + \sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t_s} \sum_{\substack{z\neq s\\ z\notin\mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)\\ z\notin\overline{\mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)}}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon_{\mathrm{initial}(z,x)s'}^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}(x,t,y-\rho_s x)\,dx \\
&+ \sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t_s} \sum_{\substack{z\neq s\\ z\in\mathsf{Sat}^{\mathcal{M}}(\neg\Phi\wedge\Psi)}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon_{\mathrm{initial}(z,x)s'}^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}(x,t,y-\rho_s x)\,dx \\
&= e^{Q_{ss}t} + \sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \int_0^{t_s} \sum_{\substack{z\neq s\\ z\notin\overline{\mathsf{Sat}(\neg\Phi\wedge\Psi)}}} e^{Q_{ss}x} Q_{sz} \cdot \Upsilon_{\mathrm{initial}(z,x)s'}^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi\right\rangle}(x,t,y-\rho_s x)\,dx \\
&= e^{Q^y_{ss}} + \int_0^{t_s} \sum_{\substack{z\neq s\\ z\notin\overline{\mathsf{Sat}(\neg\Phi\wedge\Psi)}}} e^{Q_{ss}x} Q_{sz} \cdot \sum_{s'\in\mathsf{Sat}^{\mathcal{M}}(\Psi)} \Upsilon_{\mathrm{initial}(z,x)s'}^{\mathcal{M}\left\langle \Phi\,\mathcal{U}^{[\inf I,t]\ominus x}_{[\inf J,y]\ominus\rho_s x}\Psi\right\rangle}(0,t-x,y-\rho_s x)\,dx \\
&= e^{Q_{ss}t} + \int_0^{t_s} \sum_{\substack{z\neq s\\ z\notin\overline{\mathsf{Sat}(\neg\Phi\wedge\Psi)}}} e^{Q_{ss}x} Q_{sz} \cdot \mathit{Prob}_z(\Phi\,\mathcal{U}^{[\inf I,t]\ominus x}_{[\inf J,y]\ominus x}\Psi)\,dx \\
&= \mathit{Prob}_s(\Phi\,\mathcal{U}^{[\inf I,t]}_{[\inf J,y]}\Psi).
\end{aligned}
$$

A proof of Theorem 3 in case only a homogeneous transformed MRM is needed can be found in [24].

Appendix B

Publications by the author

[49] B. R. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier. Model checking performability properties. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)*, pages 102–112. IEEE Press, 2002.

[27] L. Cloth, B. R. Haverkort, H. Hermanns, J.-P. Katoen, and C. Baier. Model checking pathCSL. In *Proceedings of the 6th International Workshop on Performability Modeling of Computer and Communications Systems (PMCCS'03)*, pages 19–22, 2003.

[4] C. Baier, L. Cloth, B. R. Haverkort, M. Kuntz, and M. Siegle. Model checking action- and state-labelled Markov chains. In *International Conference on Dependable Systems and Networks (DSN'04)*, pages 701–710. IEEE Press, 2004.

[89] A. Remke, B. R. Haverkort, and L. Cloth. Model checking infinite-state Markov chains. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*, Lecture Notes in Computer Science 3440, pages 237–252. Springer-Verlag, 2005.

[28] L. Cloth, J.-P. Katoen, M. Khattri, and R. Pulungan. Model checking Markov reward models with impulse rewards. In *International Conference on Dependable Systems and Networks (DSN'05).* IEEE Press, 2005.

[26] L. Cloth and B. R. Haverkort. Surviving survivability specifications. In *Supplemental Volume of the International Conference on Dependable Systems and Networks (DSN'05).* IEEE Press, 2005.

[24] L. Cloth and B. R. Haverkort. Hyperbolic PDEs for CSRL model checking: A déjà vu. In *Proceedings of the 7th International Workshop on Performability Modeling of Computer and Communications Systems (PMCCS'05)*, pages 19–22, 2005.

[25] L. Cloth and B. R. Haverkort. Model checking for survivability! In *Proceedings of the 2nd International Conference on the Quantitative Evaluation of Systems (QEST'05)*, pages 145–154, 2005.

[88] A. Remke, L. Cloth, and B. R. Haverkort. Uniformization with representatives: Transient analysis of infinite-state Quasi-Birth-Death processes. *Submitted for publication*, 2005.

[5] C. Baier, L. Cloth, B. R. Haverkort, M. Kuntz, and M. Siegle. Model checking Markov chains with state labels and actions. *Submitted for publication*, 2005.

Bibliography

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modeling with Generalised Stochastic Petri Nets*. John Wiley & Sons, 1995.

[2] A. Avizienis, J.-C. Laprie, and B. Randell. Dependability and its threats: a taxonomy. In *Proceedings of the 18th IFIP World Computer Congress. Building the Information Society*, volume 12, pages 91–120. Kluwer Academic Publishers Group, 2004.

[3] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous-time Markov chains. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV'96)*, Lecture Notes in Computer Science 1102, pages 269–276. Springer-Verlag, 1996.

[4] C. Baier, L. Cloth, B. R. Haverkort, M. Kuntz, and M. Siegle. Model checking action- and state-labelled Markov chains. In *International Conference on Dependable Systems and Networks (DSN'04)*, pages 701–710. IEEE Press, 2004.

[5] C. Baier, L. Cloth, B. R. Haverkort, M. Kuntz, and M. Siegle. Model checking Markov chains with state labels and actions. *Submitted for publication*, 2005.

[6] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP'00)*, Lecture Notes in Computer Science 1853, pages 780–792. Springer-Verlag, 2000.

[7] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

[8] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, Lecture Notes in Computer Science 1664, pages 146–161. Springer-Verlag, 1999.

[9] J. Bechta Dugan and G. Ciardo. Stochastic Petri net analysis of a replicated file system. *IEEE Transactions on Software Engineering*, 15(4):394–401, 1989.

[10] M. Bernardo. An algebra-based method to associate rewards with EMPA terms. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP'97)*, Lecture Notes in Computer Science 1256, pages 358–368. Springer-Verlag, 1997.

[11] M. Bernardo. Let's evaluate performance algebraically. *ACM Computing Surveys*, 31(3es), 1999.

[12] P. Buchholz, J.-P. Katoen, P. Kemper, and C. Tepper. Model checking large structured Markov chains. *Journal of Logic and Algebraic Programming*, 56(1/2):69–97, 2003.

[13] O. Burkinshaw and C. D. Aliprantis. *Principles of Real Analysis.* Academic Press, 2nd edition, 1990.

[14] S. Carney, M. Heroux, and G. Li. A proposal for a sparse BLAS toolkit. Technical Report TR/PA/92/90 (Revised), CERFACS, Toulouse, 1993.

[15] G. Ciardo, A. Blakemore, P. F. Chimento, J. K. Muppala, and K. S. Trivedi. Automated generation and analysis of Markov reward models using stochastic reward nets. In *Linear Algebra, Markov Chains and Queueing Models*, pages 145–191. Springer-Verlag, 1993.

[16] G. Ciardo, J. Muppala, and K. Trivedi. SPNP: Stochastic Petri net package. In *Proceedings of the 3rd International Workshop on Petri Nets and Performance Models (PNPM'89)*, pages 142–151. IEEE Press, 1989.

[17] B. Ciciani and V. Grassi. Performability evaluation of fault-tolerant satellite systems. *IEEE Transactions on Communications*, 35(4):403–409, 1987.

[18] E. Cinlar. *Introduction to Stochastic Processes.* Prentice Hall, 1975.

[19] G. Clark. Formalising the specification of rewards with PEPA. In *Proceedings of the 4th International Workshop on Process Algebra and Performance Modelling (PAPM'96)*, pages 139–160. C.L.U.T. Press, 1996.

[20] G. Clark and J. Hillston. Towards automatic derivation of performance measures from PEPA models. In *Proceedings of the 12th Annual UK Performance Engineering Workshop*, pages 65–81. Edinburgh University Press, 1996.

[21] E. Clarke and E. Emerson. Design and synthesis of synchronisation skeletons using branching time temporal logic. In *Logic of Programs*, Lecture Notes in Computer Science 131, pages 52–71. Springer-Verlag, 1981.

[22] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[23] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[24] L. Cloth and B. R. Haverkort. Hyperbolic PDEs for CSRL model checking: A déjà vu. In *Proceedings of the 7th International Workshop on Performability Modeling of Computer and Communications Systems (PMCCS'05)*, pages 19–22, 2005.

[25] L. Cloth and B. R. Haverkort. Model checking for survivability! In *Proceedings of the 2nd International Conference on the Quantitative Evaluation of Systems (QEST'05)*, pages 145–154, 2005.

[26] L. Cloth and B. R. Haverkort. Surviving survivability specifications. In *Supplemental Volume of the International Conference on Dependable Systems and Networks (DSN'05).* IEEE Press, 2005.

[27] L. Cloth, B. R. Haverkort, H. Hermanns, J.-P. Katoen, and C. Baier. Model checking pathCSL. In *Proceedings of the 6th International Workshop on Performability Modeling of Computer and Communications Systems (PMCCS'03)*, pages 19–22, 2003.

[28] L. Cloth, J.-P. Katoen, M. Khattri, and R. Pulungan. Model checking Markov reward models with impulse rewards. In *International Conference on Dependable Systems and Networks (DSN'05).* IEEE Press, 2005.

[29] A. Conway and N. Georganas. *Queueing Networks: Exact Computational Analysis.* MIT Press, 1989.

[30] D. D'Aprile, S. Donatelli, and J. Sproston. CSL model checking for the GreatSPN tool. In *Proceedings of the 19th International Symposium on Computer and Information Sciences (ISCIS'04)*, Lecture Notes in Computer Science 3280, pages 543–552. Springer-Verlag, 2004.

[31] E. de Souza e Silva and H. R. Gail. Calculating cumulative operational time distributions of repairable computer systems. *IEEE Transactions on Computers*, C-35(4):322–332, 1986.

[32] E. de Souza e Silva and H. R. Gail. Calculating availability and performability measures of repairable computer systems using randomization. *Journal of the ACM*, 36(1):171–193, 1989.

[33] E. de Souza e Silva and H. R. Gail. An algorithm to calculate transient distributions of cumulative reward. Technical Report CSD-940021, University of California, 1994.

[34] E. de Souza e Silva, H. R. Gail, and R. Vallejos Campos. Calculating transient distributions of cumulative reward. *ACM SIGMETRICS Performance Evaluation Review*, 23(1):231–240, 1995.

[35] D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. Doyle, W. Sanders, , and P. Webster. The mbius framework and its implementation. *IEEE Transactions on Software Engineering*, 28(10):956–969, 2002.

[36] M. C. Diniz, E. de Souza e Silva, and H. R. Gail. Calculating the distribution of a linear combination of uniform order statistics. *INFORMS Journal on Computing*, 14(2):124–131, 2002.

[37] L. Donatiello and B. R. Iyer. Analysis of a composite performance reliability measure for fault-tolerant systems. *Journal of the ACM*, 34(1):179–199, 1987.

[38] R. Dudley. *Real Analysis and Probability.* Cambridge University Press, 2002.

[39] B. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. Longstaff, and N. R. Mead. Survivable network systems: An emerging discipline. Technical Report CMU/SEI-97-TR-013, Carnegie Mellon Software Engeneering Institute, 1997.

[40] B. L. Fox and P. W. Glynn. Computing Poisson probabilities. *Communications of the ACM*, 31(4):440–445, 1988.

[41] D. Furchtgott and J. F. Meyer. Performability solution method for degradable non-repairable systems. *IEEE Transactions on Computers*, 33(6):550–553, 1984.

[42] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 29–43. ACM, 2003.

[43] A. Goyal and A. Tantawi. Evaluation of performability for degradable computer systems. *IEEE Transactions on Computers*, 36(6):738–744, 1987.

[44] W. K. Grassmann. Finding transient solutions in Markovian event systems through randomization. In W. Stewart, editor, *Numerical Solution of Markov Chains*, pages 357–371. Marcel Dekker Inc., 1991.

[45] D. Gross and D. R. Miller. The randomization technique as a modeling tool and solution procedure for transient Markov processes. *Operations Research*, 32(2):343–361, 1984.

[46] V. Gupta, V. Lam, H. V. Ramasamy, W. H. Sanders, and S. Singh. Dependability and performance evaluation of intrusion-tolerant server architectures. In *First Latin-American Symposium on Dependable Computing (LADC'03)*, Lecture Notes in Computer Science 2847, pages 81–101. Springer-Verlag, 2003.

[47] B. R. Haverkort. *Performance of Computer Communication Systems. A Model-Based Approach.* John Wiley & Sons, 1998.

[48] B. R. Haverkort, H. Bohnenkamp, and A. Bell. On the efficient sequential and distributed evaluation of very large stochastic Petri nets. In *Proceedings of the 8th International Workshop on Petri Nets and Performance Models (PNPM'99)*, pages 12–21. IEEE Press, 1999.

[49] B. R. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier. Model checking performability properties. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)*, pages 102–112. IEEE Press, 2002.

[50] B. R. Haverkort, H. Hermanns, J.-P. Katoen, and C. Baier. Model checking CSRL-specified performability properties. In *Proceedings of the 5th International Workshop on Performability Modeling of Computer and Communications Systems (PMCCS'01)*, pages 105–109, 2001.

[51] B. R. Haverkort and J.-P. Katoen. The performability distribution for nonhomogeneous Markov reward models. In *Proceedings of the 7th International Workshop on Performability Modeling of Computer and Communications Systems (PMCCS'05)*, pages 32–34, 2005.

[52] B. R. Haverkort and I. G. Niemegeers. Performability modelling tools and techniques. *Performance Evaluation*, 25(1):17–40, 1996.

[53] B. R. Haverkort and K. S. Trivedi. Specification techniques for Markov reward models. *Discrete Event Systems: Theory and Applications*, 3(2/3):219–247, 1993.

[54] H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1-2):43–87, 2002.

[55] H. Hermanns, U. Herzog, K. U., V. Mertsiotakis, and M. Siegle. Compositional performance modeling with the TIPPtool. *Performance Evaluation*, 39(1–4):5–35, 2000.

[56] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. Towards model checking stochastic process algebra. In *Proceedings of the 2nd International Conference on Integrated Formal Methods (IFM'00)*, Lecture Notes in Computer Science 1945, pages 420–439. Springer-Verlag, 2000.

[57] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. *Journal on Software Tools for Technology Transfer*, 4(2):153–172, 2003.

[58] J. Hillston. *A Compositional Approach to Performance Modeling.* Camebridge University Press, 1996.

[59] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 2nd edition, 2001.

[60] R. Howard. *Dynamic Probabilistic Systems; Volume II: Semi-Markov and decision processes*. John Wiley & Sons, 1971.

[61] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Proceedings of the 2nd International Conference on the Quantitative Evaluation of Systems (QEST'05)*, pages 243–244. IEEE Press, 2005.

[62] M. Khattri and R. M. Pulungan. Model checking Markov reward models with impulse rewards. Master's thesis, University of Twente, Department of Computer Science, 2004.

[63] J. C. Knight, E. A. Strunk, and K. J. Sullivan. Towards a rigorous definition of information system survivability. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX'03)*, pages 78–89. IEEE Press, 2003.

[64] V. Kulkarni, V. Nicola, R. Smith, and K. Trivedi. Numerical evaluation of performability and job completion time in repairable fault-tolerant systems. In *Proceedings of the 16th International Symposium on Fault-Tolerant Computing (FTCS'85)*, pages 252–257, 1985.

[65] M. Kuntz and M. Siegle. A stochastic extension of the logic PDL. In *Proceedings of the 6th International Workshop on Performability Modeling of Computer and Communications Systems (PMCCS 2003)*, pages 58–61, 2003.

[66] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking using PRISM: a hybrid approach. *Journal on Software Tools for Technology Transfer*, 6(2):128–142, 2004.

[67] S. C. Liew and K. W. Lu. A framework for characterizing disaster-based network survivability. *IEEE Journal on Selected Areas in Communications*, 12(1):52–58, 1994.

[68] S. Ling, Z. Zesheng, and G. Hengshen. A GIS-based agricultural disaster evaluation system. In *Proceedings of the ESRI International User Conference*, 1998.

[69] Y. Liu, V. B. Mediratta, and K. S. Trivedi. Survivability analysis of telephone access network. In *Proceedings of the 5th IEEE International Symposium on Software Engineering (ISSRE'04)*, pages 367–378. IEEE Press, 2004.

[70] Y. Liu and K. S. Trivedi. A general framework for network survivability quantification. In *Proceedings of the 12th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB'04)*, pages 369–378. VDE Verlag, 2004.

[71] T. Matsunawa. The exact and approximate distributions of linear combinations of selected order statistics from uniform distributions. *The Annals of the Institute of Statistical Mathematics*, 37:1–16, 1985.

[72] D. Medhi. A unified approach to network survivability for teletraffic networks: Models, algorithms and analysis. *IEEE Transactions on Communications*, 42(2/3/4):534–548, 1994.

[73] J. F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Transactions on Computers*, 29(8):720–731, 1980.

[74] J. F. Meyer. Performability: a retrospective and some pointers to the future. *Performance Evaluation*, 14(3):139–156, 1992.

[75] J. F. Meyer, A. Movaghar, and W. H. Sanders. Stochastic activity networks: Structure, behavior and application. In *Proceedings of the International Workshop on Timed Petri Nets*, pages 106–115. IEEE Press, 1985.

[76] J. Meyer-Kayser. *Automatische Verifikation Stochastischer Prozesse.* PhD thesis, University of Erlangen-Nuremberg, 2004. (in German).

[77] C. Moler and C. van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–836, 1978.

[78] C. Moler and C. van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.

[79] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models. An Algorithmic Approach.* Dover Publications, 1994.

[80] W. D. Obal II and W. H. Sanders. State-space support for path-based reward variables. *Performance Evaluation*, 35(3-4):233–251, 1999.

[81] K. R. Pattipati, Y. Li, and H. A. Blom. A unified framework for the performability evaluation of fault tolerant computer systems. *IEEE Transactions on Computers*, 42(3):312–326, 1993.

[82] K. R. Pattipati, R. Mallubhatla, V. Gopalakrishna, and N. Viswanatham. Markov-reward models and hyperbolic systems. In *Performability Modelling*, pages 83–106. John Wiley & Sons, 2001.

[83] K. R. Pattipati and S. A. Shah. On the computational aspect of performability models of fault-tolerant computer systems. *IEEE Transactions on Computers*, 39(6):832–836, 1990.

[84] D. Pradhan, editor. *Fault-tolerant and dependable computer system design.* Prentice Hall, 2nd edition, 2003.

[85] M. A. Qureshi. Reward model solution methods with impulse and rate rewards: An algorithm and numerical results. Master's thesis, University of Arizona, 1992.

[86] M. A. Qureshi and W. H. Sanders. Reward model solution methods with impulse and rate rewards: an algorithm and numerical results. *Performance Evaluation*, 20(4):413–436, 1994.

[87] M. A. Qureshi, W. H. Sanders, A. P. van Moorsel, and R. German. Algorithms for the generation of state-level representations of stochastic activity networks with general reward structures. *IEEE Transactions on Software Engineering*, 22(9):603–613, 1996.

[88] A. Remke, L. Cloth, and B. R. Haverkort. Uniformization with representatives: Transient analysis of infinite-state Quasi-Birth-Death processes. *Submitted for publication*, 2005.

[89] A. Remke, B. R. Haverkort, and L. Cloth. Model checking infinite-state Markov chains. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*, Lecture Notes in Computer Science 3440, pages 237–252. Springer-Verlag, 2005.

[90] S. Ross. *Stochastic Processes.* John Wiley & Sons, 1983.

[91] G. Rubino and B. Sericola. Interval availability distribution computation. In *Proceedings of the 23rd IEEE International Symposium on Fault Tolerant Computing (FTCS'93)*, pages 48–55. IEEE Press, 1993.

[92] R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package.* Kluwer Academic Publishers Group, 1996.

[93] W. Sanders, W. Obal II, M. Qureshi, and F. Widnajarko. The UltraSAN modeling environment. *Performance Evaluation*, 24:89–115, 1995.

[94] B. Sericola. Closed-form solution for the distribution of the total time spent in a subset of states of a homogeneous Markov process during a finite observation period. *Journal for Applied Probability*, 27(3):713–719, 1990.

[95] B. Sericola. Occupation times in Markov processes. *Communications in Statistics — Stochastic Models*, 16(5):479–510, 2000.

[96] M. L. Shooman. *Reliability of Computer Systems and Networks. Fault Tolerance, Analysis and Design.* John Wiley & Sons, 2002.

[97] S. Singh, M. Cukier, and W. H. Sanders. Probabilistic validation of an intrusion-tolerant replication system. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'03)*, pages 615–624. IEEE Press, 2003.

[98] R. Smith, K. S. Trivedi, and A. Ramesh. Performability analysis: Measures, an algorithm and a case study. *IEEE Transactions on Computers*, 37(4):406–417, April 1988.

[99] F. Stevens, T. Courtney, S. Singh, A. Agbaria, J. F. Meyer, W. H. Sanders, and P. Pal. Model-based validation of an intrusion-tolerant information system. In *Proceedings of the 23rd Symposium on Reliable Distributed Systems (SRDS'04)*, pages 184–194. IEEE Press, 2004.

[100] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, 1994.

[101] J. M. Thomsen and R. Manggaard. Analysis of GSM handover using coloured Petri nets. Master's thesis, University of Aarhus, 2003.

[102] H. Tijms and R. Veldman. A fast algorithm for the transient reward distribution in continuous-time Markov chains. *Operations Research Letters*, 26:155–158, 2000.

[103] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* John Wiley & Sons, 2nd edition, 2001.

[104] K. S. Trivedi, J. K. Muppala, S. Woolet, and B. R. Haverkort. Composite performance and dependability analysis. *Performance Evaluation*, 14:197–215, 1992.

[105] J. Ventura Agustina, P. Zhang, and R. Kantola. Performance evaluation of GSM handover traffic in a GPRS/GSM network. In *Proceedings of the 8th IEEE International Symposium on Computers and Communications*, pages 137–142. IEEE Press, 2003.

[106] B. H. Walke. *Mobile Radio Networks.* John Wiley & Sons, 1999.

[107] H. Weisberg. The distribution of linear combinations of order statistics from the uniform distribution. *Annals of Institute of Statistics*, 42(2):704–709, 1971.

[108] A. Zolfaghari and F. J. Kaudel. Framework for network survivability performance. *IEEE Journal on Selected Areas in Communications*, 12(1):46–51, 1994.

Index