Bisimulation Minimisation and Probabilistic Model Checking

Tim Kemna

Master's Thesis in Computer Science

August 2006

University of Twente
Faculty of EEMCS
Formal Methods and Tools
Enschede, The Netherlands

Graduation committee:
prof. dr. ir. J.-P. Katoen
dr. D. N. Jansen
I. S. Zapreev MSc

Samenvatting

Probabilistisch model checking is een techniek voor de verificatie van probabilistische systemen. De grootte van de toestandsruimte is een beperkende factor voor model checking. Om dit op te lossen, hebben we een techniek toegepast genaamd *bisimulatie-minimalisatie*. Dit is een techniek waarbij het model geminimaliseerd wordt voorafgaande aan model checken. We hebben ook een techniek beschouwd waarbij het model geminimaliseerd wordt voor één specifieke formule. Het minimalisatie-algoritme is geïmplementeerd in de model checker MRMC. Aan de hand van case studies hebben we empirisch de effectiviteit van bisimulatie-minimalisatie voor probabilistisch model checking bestudeerd. De modellen die we beschouwen zijn Markov-ketens met discrete of continue tijd. Formules worden uitgedrukt in de temporele logica PCTL of CSL. Uit onze experimenten is gebleken dat bisimulatie-minimalisatie kan leiden tot grote reducties van de toestandsruimte. In een aantal gevallen is minimaliseren plus checken van het geminimaliseerde model sneller dan het originele model te checken. We concluderen dat bisimulatie-minimalisatie een goede techniek is voor het reduceren van de toestandsruimte.

Abstract

Probabilistic model checking is a technique for the verification of probabilistic systems. The size of the state space is a limiting factor for model checking. We used *bisimulation minimisation* to combat this problem. Bisimulation minimisation is a technique where the model under consideration is first minimised prior to the actual model checking. We also considered a technique where the model is minimised for a specific property, called *formula-dependent lumping*. The minimisation algorithm has been implemented into the model checker MRMC. Using case studies, we empirically studied the effectiveness of bisimulation minimisation for probabilistic model checking. The probabilistic models we consider are discrete-time Markov chains and continuous-time Markov chains. Properties are expressed in the temporal logic PCTL or CSL. Our experiments showed that bisimulation minimisation can result into large state space reductions. Formula-dependent lumping can lead to even larger state space reductions. For several cases, minimising the original model plus checking the minimised model is faster than model checking the original model. We conclude that bisimulation minimisation is a good state space reduction technique.

Preface

Working on my Master's thesis was one of the most challenging parts of my studies Computer Science at the University of Twente. I learnt a lot about an interesting research area which was relatively new to me when I started this assignment: probabilistic model checking. Also, I have been given the opportunity to look inside a model checker and to implement an extension for it.

Finally, I would like to thank Joost-Pieter Katoen and David Jansen for their supervision and support. Last, but not least, I would like to thank Ivan Zapreev for answering all my questions concerning MRMC.

Tim Kemna,
Enschede, August 2006

Contents

Chapter 1

Introduction

Model checking [9] is a technique for verifying software or hardware systems in an automated way, such as real-time embedded or safety-critical systems. Using a formal language, we can define a model which describes the system requirements or the design of the system. A model checking tool verifies if the model satisfies a formal specification, called a property or formula. This specification is often expressed in a temporal logic, such as Computation Tree Logic (CTL) [8]. In other words, model checking is a technique to establish the correctness of the system.

Probabilistic model checking is a verification technique for probabilistic systems. In these systems, there is a certain probability associated with events. The probabilistic models we consider are discrete-time Markov chains (DTMCs) and continuous-time Markov chains (CTMCs). Probabilistic Computation Tree Logic (PCTL) [13] is a temporal logic that extends CTL. It provides means to express properties which are interpreted over DTMCs. Continuous Stochastic Logic (CSL) [5] is used to express properties on CTMCs. These logics allow formulating properties such as: the probability a bad state is reached within 50 seconds is less than 10%.

For conventional as well as probabilistic model checking, the size of the state space (i. e. the number of states of the model) is a limiting factor for model checking. One way to combat this problem is to use state space reduction techniques, such as multi-terminal binary decision diagrams (MTBDDs) [17], symmetry reduction [23], or bisimulation minimisation. This thesis focuses on bisimulation minimisation.

Bisimulation minimisation is a technique where the model under consideration is first minimised prior to the actual model checking. For CTL model checking, the cost of performing this reduction outweighs that of model checking the original, non-minimised model [11]. In the probabilistic setting this is unclear as the computations for bisimulation minimisation are

as simple as for CTL model checking, whereas model checking is computationally more complex. In this thesis, we empirically study the effectiveness of bisimulation minimisation for probabilistic model checking.

We implemented the bisimulation minimisation algorithm (i. e. the lumping algorithm [10]) into the model checking tool Markov Reward Model Checker (MRMC) [20]. This tool is currently being developed at the University of Twente and at the RWTH Aachen University. We used several case studies from the PRISM website [26]. In these case studies, a probabilistic model of an algorithm or protocol is defined in the PRISM language. In our study, we only used PRISM to build and export the models. Using MRMC, we minimised this original model to compute a lumped model. We conducted several experiments using these models.

In chapter 2 the theoretical background of DTMCs, CTMCs, PCTL, CSL and bisimulation equivalence is introduced. Furthermore, the lumping algorithm is presented. In chapter 3 the implementation of the lumping algorithm into MRMC is explained. Chapter 4 describes experiments to check the effectiveness of bisimulation minimisation for PCTL model checking. For CSL model checking, experiments are described in chapter 6. This chapter also compares bisimulation minimisation to symmetry reduction. Symmetry reduction is a technique to reduce symmetric models prior to model checking. Chapters 5 and 7 are devoted to techniques and experiments to minimise the model for a specific PCTL or CSL formula, respectively. We call this technique formula-dependent lumping, whereas bisimulation minimisation in chapters 4 and 6 can be viewed as formula-independent lumping. Finally, chapter 8 presents the conclusion and future work.

Chapter 2

Preliminaries

This chapter introduces the basic concepts and definitions for DTMCs and CTMCs. Then the syntax, semantics and model checking algorithms of PCTL and CSL are explained. Finally, bisimulation equivalence and the lumping algorithm are presented. Definitions and notations in this chapter are used in the remainder of this thesis.

2.1 Discrete-time Markov chains

A DTMC is considered as a Kripke structure with probabilistic transitions. Every transition corresponds to one time unit.

Definition 1. *A (labelled) discrete-time Markov chain (DTMC) is a triple*

$$\mathcal{D} = (S, \mathbf{P}, L),$$

where

- S *is a finite set of states,*
- $\mathbf{P}$ *is the* transition probability matrix, $\mathbf{P} : S \times S \to [0, 1]$, *such that for all* s *in* S*:*
$$\sum_{s' \in S} \mathbf{P}(s, s') = 1,$$
- L *is a labelling function,* $L : S \to 2^{AP}$, *that labels any state* $s \in S$ *with those atomic propositions* $a \in AP$ *that are valid in* S*.*

The probability of going to state s' from state s is $\mathbf{P}(s, s')$. If $\mathbf{P}(s, s') = 0$, there is no transition from s to s'. Whenever $\mathbf{P}(s, s') > 0$, state s' is called a successor of s and s is a predecessor of s'. A state s is called *absorbing*, if $\mathbf{P}(s, s) = 1$. Such a state has a self-loop and no other outgoing transitions.

Definition 2. *A path σ in a DTMC $\mathcal{D}$ is an infinite sequence*

$$\sigma = s_0 \to s_1 \to \cdots s_i \to \cdots$$

of states with s_0 as the first state such that $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. The $(i+1)$-th state s_i of σ is denoted as $\sigma[i]$, and the prefix of σ of length n is denoted $\sigma \uparrow n$, i. e. $\sigma \uparrow n = s_0 \to s_1 \to \cdots \to s_n$. Let $Path^{\mathcal{D}}(s)$ denote the set of paths in $\mathcal{D}$ that start in s.

Following measure theory a probability measure can be defined on the sets of paths [13].

Definition 3. *The probability measure* Pr *on the sets of paths in DTMC $\mathcal{D}$ starting in s_0 is defined as follows for $n > 0$:*

$$\Pr(\{\sigma \in Path^{\mathcal{D}}(s_0) \mid \sigma \uparrow n = s_0 \to s_1 \to \cdots \to s_n\}) = \mathbf{P}(s_0, s_1) \times \cdots \times \mathbf{P}(s_{n-1}, s_n)$$

and for $n = 0$: $\Pr(\{\sigma \in Path^{\mathcal{D}}(s_0) \mid \sigma \uparrow 0 = s_0\}) = 1$

2.2 Continuous-time Markov chains

In a DTMC each transition corresponds to one time unit. A CTMC has a continuous time range. Each transition is equipped with an exponentially distributed delay.

Definition 4. *A (labelled) continuous-time Markov chain (CTMC) is a triple*

$$\mathcal{C} = (S, \mathbf{R}, L),$$

with S and L as before, and $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ as the rate matrix.

There is a transition from s to s', if $\mathbf{R}(s, s') > 0$. A state s is called absorbing, if $\mathbf{R}(s, s') = 0$ for all states s'. With probability $1 - e^{-\lambda \cdot t}$ the transition $s \to s'$ can be triggered within t time units. If $\mathbf{R}(s, s') > 0$ for more than one state s', a *race* exists between the outgoing transitions from s. The probability to move from nonabsorbing state s to state $s' \neq s$ within t time units is:

$$\mathbf{P}(s, s', t) = \frac{\mathbf{R}(s, s')}{E(s)} \cdot (1 - e^{-E(s) \cdot t}),$$

where $E(s) = \sum_{s' \in S} \mathbf{R}(s, s')$ denotes the *exit rate* at which any transition from s is taken.

A path in a CTMC is similar to a path in a DTMC except that the amount of time in each visited state is recorded.

Definition 5. *Let CTMC $\mathcal{C} = (S, \mathbf{R}, L)$ be a CTMC. An infinite path σ in $\mathcal{C}$ is a infinite sequence $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} \cdots$ with $s_i \in S$ and $t_i \in \mathbb{R}_{>0}$ such that $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. A finite path is a sequence $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \cdots s_{n-1} \xrightarrow{t_{n-1}} s_n$ such that s_n is absorbing and $\mathbf{R}(s_i, s_{i+1}) > 0$ for $0 \leq i < n$. Let $Path^{\mathcal{C}}(s)$ denote the set of (finite and infinite) paths in $\mathcal{C}$ that start in s.*

For infinite path σ and $i \geq 0$, let $\sigma[i]$ denote the $(i+1)$-th state of σ and $\delta(\sigma, i) = t_i$, the time spent in state s_i. For $t \in \mathbb{R}_{\geq 0}$ and i the smallest index with $t \leq \sum_{j=0}^{i} t_j$, the state of σ occupied at time t is denoted by $\sigma@t = \sigma[i]$. Let Pr denote the unique probability measure on sets of paths, for details see [5].

The time-abstract probabilistic behaviour of CTMC $\mathcal{C}$ is described by its *embedded* DTMC:

Definition 6. *The embedded DTMC of CTMC $\mathcal{C} = (S, \mathbf{R}, L)$ is given by $emb(\mathcal{C}) = (S, \mathbf{P}, L)$, where $\mathbf{P}(s, s') = \mathbf{R}(s, s')/E(s)$ if $E(s) > 0$ and $\mathbf{P}(s, s') = 0$ otherwise.*

Uniformisation is the transformation of a CTMC into a DTMC:

Definition 7. *For CTMC $\mathcal{C} = (S, \mathbf{R}, L)$, the uniformised DTMC is defined by $unif(\mathcal{C}) = (S, \mathbf{U}, L)$, where $\mathbf{U} = I + \mathbf{Q}/q$ with $\mathbf{Q} = \mathbf{R} - diag(\underline{E})$. The* uniformisation rate *q must be chosen such that $q \geq max_s\{E(s)\}$.*

$\mathbf{E} = diag(\underline{E})$ denotes the diagonal matrix with $\mathbf{E}(s, s) = E(s)$ and 0 otherwise.

2.3 Probabilistic Computation Tree Logic

The Probabilistic Computation Tree Logic (PCTL) extends the temporal logic CTL with discrete time and probabilities [13]. It consists of state formulas, which are interpreted over states of a DTMC, and path formulas, which are interpreted over paths in a DTMC.

2.3.1 Syntax and semantics

Definition 8. *The set of PCTL formulas is divided into path formulas and state formulas. Their syntax is defined inductively as follows:*

- *true is a state formula,*
- *Each atomic proposition $a \in AP$ is a state formula,*
- *If Φ and Ψ are state formulas, then so are $\neg\Phi$ and $\Phi \wedge \Psi$,*
- *If Φ is a state formula, then $X\,\Phi$ is a path formula,*
- *If Φ and Ψ are state formulas and $t \in \mathbb{N}$, then $\Phi\,\mathcal{U}^{\leq t}\,\Psi$ and $\Phi\,\mathcal{U}\,\Psi$ are path formulas,*
- *If ϕ is a path formula and p a real number with $0 \leq p \leq 1$ and let $\trianglelefteq \in \{\leq, <, >, \geq\}$ be a comparison operator, then $\mathcal{P}_{\trianglelefteq p}(\phi)$ is a state formula.*

The operator X is the next operator, $\mathcal{U}^{\leq t}$ is the bounded until operator, and $\mathcal{U}$ is the unbounded until operator. The next operator and the unbounded until operator have the same meaning as in CTL. The bounded until operator $\Phi\,\mathcal{U}^{\leq t}\,\Psi$ means that both Ψ will become true within t time units and that Φ will be true from now on until Ψ becomes true. The formula $\mathcal{P}_{\trianglelefteq p}(\phi)$ expresses that the probability measure of paths satisfying ϕ meets the bound $\trianglelefteq p$. This operator replaces the usual path quantifiers $\exists$ and $\forall$ from CTL. Other Boolean operators ($\vee$ and $\rightarrow$) can be derived from $\wedge$ and $\neg$ as usual.

Given a DTMC $\mathcal{D} = (S, \mathbf{P}, L)$ the meaning of PCTL formulas is defined by a satisfaction relation, denoted by $\models_{\mathcal{D}}$, with respect to a state s or a path σ.

Definition 9. *The satisfaction relation $\models_{\mathcal{D}}$ for PCTL formulas on a DTMC $\mathcal{D} = (S, \mathbf{P}, L)$ is defined by:*

$s \models_{\mathcal{D}} true$	*for all* $s \in S$
$s \models_{\mathcal{D}} a$	*iff* $a \in L(s)$
$s \models_{\mathcal{D}} \neg\Phi$	*iff* $s \not\models_{\mathcal{D}} \Phi$
$s \models_{\mathcal{D}} \Phi \wedge \Psi$	*iff* $s \models_{\mathcal{D}} \Phi$ *and* $s \models_{\mathcal{D}} \Psi$
$s \models_{\mathcal{D}} \mathcal{P}_{\trianglelefteq p}(\phi)$	*iff* $\Pr(\{\sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models_{\mathcal{D}} \phi\}) \trianglelefteq p$
$\sigma \models_{\mathcal{D}} X\,\Phi$	*iff* $\sigma[1] \models_{\mathcal{D}} \Phi$
$\sigma \models_{\mathcal{D}} \Phi\,\mathcal{U}^{\leq t}\,\Psi$	*iff* $\exists\, i \leq t.\big(\sigma[i] \models_{\mathcal{D}} \Psi \wedge (\forall j\,.\,0 \leq j < i\,.\,\sigma[j] \models_{\mathcal{D}} \Phi)\big)$.

2.3.2 Model checking

The model checking algorithm for checking PCTL property ψ on DTMC $\mathcal{D} = (S, \mathbf{P}, L)$ is based on the algorithm for model checking CTL [8]. It involves the calculation of satisfaction sets $Sat(\psi)$, where $Sat(\psi) = \{s \in S \mid s \models \psi\}$. In order to calculate these sets, the syntax tree of ψ is constructed. This syntax tree is traversed bottom-up while calculating the satisfaction sets of the subformulas of ψ.

Algorithms for calculating the satisfaction sets of until formulas are described below. Calculation of satisfaction sets of other subformulas is straightforward, for details see [13].

Bounded until operator

This algorithm calculates the satisfaction set for $\psi = \mathcal{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}^{\leq t}\, \Psi)$ assuming $Sat(\Phi)$ and $Sat(\Psi)$ are given.

The set of states S is partitioned into three subsets S_s, S_f and S_i:

$$\begin{aligned} S_s &= \{s \in S \mid s \in Sat(\Psi)\} \\ S_f &= \{s \in S \mid s \notin Sat(\Phi) \wedge s \notin Sat(\Psi)\} \\ S_i &= \{s \in S \mid s \in Sat(\Phi) \wedge s \notin Sat(\Psi)\} \end{aligned}$$

The probability measure $\pi_t(s)$ for the set of paths starting in s satisfying $\Phi\, \mathcal{U}^{\leq t}\, \Psi$ is defined in the following recursion [13]:

$$\pi_t(s) = \begin{cases} 0 & \text{if } s \in S_f \vee (t = 0 \wedge s \in S_i) \\ 1 & \text{if } s \in S_s \\ \sum_{s' \in S} \mathbf{P}(s, s') \cdot \pi_{t-1}(s') & \text{if } t > 0 \end{cases}$$

States in S_s and S_f are made absorbing. This can be done safely, because once such a state has been reached the future behaviour is irrelevant for the validity of ψ. To this end, the matrix $\mathbf{P}'$ is constructed:

$$\mathbf{P}'(s, s') = \begin{cases} \mathbf{P}(s, s') & \text{if } s \in S_i \\ 1 & \text{if } s \notin S_i \wedge s = s' \\ 0 & \text{otherwise} \end{cases}$$

For $t > 0$, $\underline{\pi_t} = \mathbf{P}' \cdot \underline{\pi_{t-1}}$. In total, this requires t matrix-vector multiplications.

This vector is used to construct the satisfaction set for ψ:

$$Sat(\psi) = \{s \in S \mid \pi_t(s) \trianglelefteq p\}$$

Unbounded until operator

This algorithm calculates the satisfaction set for $\psi = \mathcal{P}_{\trianglelefteq p}(\Phi \,\mathcal{U}\, \Psi)$ assuming $Sat(\Phi)$ and $Sat(\Psi)$ are given.

The set S_f is extended to also include states from which no state in S_s is reachable. Similarly, the set S_s is extended to also include states from which all paths through S_i eventually reach a state in S_s.

$$\begin{aligned} U_s &= S_s \cup \{s \in S_i \mid \text{all paths through } S_i \text{ starting in } s \text{ reach a state in } S_s\} \\ U_f &= S_f \cup \{s \in S_i \mid \text{there exists no path in } S_i \text{ from } s \text{ to a state in } S_s\} \\ U_i &= S \setminus (U_s \cup U_f) \end{aligned}$$

These sets can be calculated using conventional graph analysis, i. e. backward search. States in U_s and U_f are made absorbing.

The following linear equation system defines the state probabilities for the unbounded until operator [13]:

$$\pi_\infty(s) = \begin{cases} 0 & \text{if } s \in U_f \\ 1 & \text{if } s \in U_s \\ \sum_{s \in S} \mathbf{P}(s, s') \cdot \pi_\infty(s') & \text{otherwise} \end{cases}$$

This linear equation system can be solved using iterative methods like the Jacobi or the Gauss-Seidel method [30]. The iteration is generally continued until the changes made by an iteration are below some ϵ:

$$\pi_t(s) - \pi_{t-1}(s) < \epsilon \quad \text{for all states } s \in S$$

Similarly to the bounded until operator, the satisfaction set for ψ is constructed:

$$Sat(\psi) = \{s \in S \mid \pi_\infty(s) \trianglelefteq p\}$$

2.4 Continuous Stochastic Logic

The Continuous Stochastic Logic (CSL) provides means to specify logical properties for CTMCs [5]. It extends PCTL with a steady state operator and continuous time intervals on next and until operators. The steady state operator refers to the probability of residing in a set of states in the long-run.

2.4.1 Syntax and semantics

Definition 10. *Let p and $\trianglelefteq$ be as before and $I \subseteq \mathbb{R}_{\geq 0}$ a non-empty interval. The syntax of CSL is:*

- *true is a state formula,*
- *Each atomic proposition $a \in AP$ is a state formula,*
- *If Φ and Ψ are state formulas, then so are $\neg\Phi$ and $\Phi \wedge \Psi$,*
- *If Φ is a state formula, then so is $\mathcal{S}_{\trianglelefteq p}(\Phi)$,*
- *If ϕ is a path formula, then $\mathcal{P}_{\trianglelefteq p}(\phi)$ is a state formula,*
- *If Φ and Ψ are state formulas, then $X^I\,\Phi$ and $\Phi\,\mathcal{U}^I\,\Psi$ are path formulas.*

The state formula $\mathcal{S}_{\trianglelefteq p}(\Phi)$ asserts that the steady state probability of being in a state satisfying Φ meets the condition $\trianglelefteq p$. The path formula $X^I\,\Phi$ asserts that a transition is made to a state satisfying Φ at some time instant $t \in I$. The path formula $\Phi\,\mathcal{U}^I\,\Psi$ asserts that Ψ is satisfied at some time instant $t \in I$ and that at all preceding time instants Φ is satisfied. Path formula $\Phi\,\mathcal{U}^{[0,\infty)}\,\Psi$ is the unbounded until formula.

Similar to PCTL, the semantics of CSL is defined by a satisfaction relation.

Definition 11. *The satisfaction relation $\models_{\mathcal{C}}$ for CSL formulas on a CTMC $\mathcal{C} = (S, \mathbf{R}, L)$ is defined by:*

$$
\begin{array}{ll}
s \models_{\mathcal{C}} \mathit{true} & \mathit{for\ all}\ s \in S \\
s \models_{\mathcal{C}} a & \mathit{iff}\ a \in L(s) \\
s \models_{\mathcal{C}} \neg\Phi & \mathit{iff}\ s \not\models_{\mathcal{C}} \Phi \\
s \models_{\mathcal{C}} \Phi \wedge \Psi & \mathit{iff}\ s \models_{\mathcal{C}} \Phi\ \mathit{and}\ s \models_{\mathcal{C}} \Psi \\
s \models_{\mathcal{C}} \mathcal{S}_{\trianglelefteq p}(\Phi) & \mathit{iff}\ \lim_{t\to\infty} \Pr(\{\sigma \in Path^{\mathcal{C}}(s) \mid \sigma@t \models_{\mathcal{C}} \Phi\}) \trianglelefteq p \\
s \models_{\mathcal{C}} \mathcal{P}_{\trianglelefteq p}(\phi) & \mathit{iff}\ \Pr(\{\sigma \in Path^{\mathcal{C}}(s) \mid \sigma \models_{\mathcal{C}} \phi\}) \trianglelefteq p \\
\sigma \models_{\mathcal{C}} X^I\,\Phi & \mathit{iff}\ \sigma[1]\ \mathit{is\ defined\ and}\ \sigma[1] \models_{\mathcal{C}} \Phi\ \mathit{and}\ \delta(\sigma,0) \in I \\
\sigma \models_{\mathcal{C}} \Phi\,\mathcal{U}^I\,\Psi & \mathit{iff}\ \exists t \in I.\big(\sigma@t \models_{\mathcal{C}} \Psi \wedge (\forall t' \in [0,t).\sigma@t' \models_{\mathcal{C}} \Phi)\big).
\end{array}
$$

2.4.2 Model checking

CSL model checking [5, 21] is performed in the same way as for PCTL, by recursively computing satisfaction sets. For the Boolean operators and unbounded until this is exactly as for PCTL. The other operators will be shortly discussed below. The probability measure for the sets of paths that satisfy ϕ and start in s in CTMC $\mathcal{C}$ is denoted by $Prob^{\mathcal{C}}(s, \phi)$.

Next operator

The probability for each state s to satisfy $X^{[t,t']}\,\Phi$ is defined by:

$$Prob^{\mathcal{C}}(s, X^{[t,t']}\,\Phi) = \left(e^{-E(s)\cdot t} - e^{-E(s)\cdot t'}\right) \cdot \sum_{s' \models_{\mathcal{C}} \Phi} \mathbf{P}(s, s')$$

These probabilities can be computed by multiplying $\mathbf{P}$ with vector $\underline{b}$, where $b(s) = e^{-E(s)\cdot t} - e^{-E(s)\cdot t'}$, if $s \in Sat(\Phi)$ and $b(s) = 0$ otherwise.

Steady state operator

To check whether $s \models_{\mathcal{C}} \mathcal{S}_{\trianglelefteq p}(\Phi)$, first each *bottom strongly connected component* (BSCC) of CTMC $\mathcal{C}$ is computed. A BSCC is a maximal subgraph of $\mathcal{C}$ in which for every pair of vertices s and s' there is a path from s to s' and a path from s' to s and once entered it cannot be left anymore. For each BSCC B containing a Φ state, the following linear equation system is solved:

$$\sum_{\substack{s \in B \\ s \neq s'}} \pi^B(s) \cdot \mathbf{R}(s, s') = \pi^B(s') \cdot \sum_{\substack{s \in B \\ s \neq s'}} \mathbf{R}(s', s) \qquad \text{with } \sum_{s \in B} \pi^B(s) = 1$$

Then, the probabilities to reach each BSCC B from a given state s are computed. State s satisfies $\mathcal{S}_{\trianglelefteq p}(\Phi)$ if:

$$\sum_{B} \left(\Pr\{\text{reach } B \text{ from } s\} \cdot \sum_{s' \in B \cap Sat(\Phi)} \pi^B(s') \right) \trianglelefteq p$$

Time-bounded until operator

Let $\pi^{\mathcal{C}}(s,t)(s')$ denote the probability of being in state s' at time t, under the condition that the CTMC $\mathcal{C}$ is in state s at time 0. CTMC $\mathcal{C}[\psi]$ is defined by the matrix obtained from $\mathcal{C}$ where states satisfying ψ are made absorbing. For formulas of the form $\mathcal{P}_{\trianglelefteq p}(\Phi\ \mathcal{U}^{[t,t']}\ \Psi)$, two cases can be distinguished: $t = 0$ and $t > 0$.

If $t = 0$, the probability measure is defined as:

$$Prob^{\mathcal{C}}(s, \Phi\ \mathcal{U}^{[0,t']}\ \Psi) = \sum_{s' \models \Psi} \pi^{\mathcal{C}[\neg\Phi \vee \Psi]}(s, t')(s')$$

For $t > 0$:

$$Prob^{\mathcal{C}}(s, \Phi\ \mathcal{U}^{[t,t']}\ \Psi) = \sum_{s' \models \Phi} \left(\pi^{\mathcal{C}[\neg\Phi]}(s,t)(s') \cdot \sum_{s'' \models \Psi} \pi^{\mathcal{C}[\neg\Phi \vee \Psi]}(s', t'-t)(s'') \right)$$

The probabilities $\pi^{\mathcal{C}}(s,t)(s')$ can be computed as follows:

$$\underline{\pi^{\mathcal{C}}(s,t)} = \underline{\pi^{\mathcal{C}}(s,0)} \cdot \sum_{k=0}^{\infty} \gamma(k, q \cdot t) \cdot \mathbf{U}^k \tag{2.1}$$

where $\mathbf{U}$ is the probability matrix of the uniformised DTMC $unif(\mathcal{C})$ and $\gamma(k, q \cdot t)$ is the kth Poisson probability with parameter $q \cdot t$.

To compute the transient probabilities numerically, the infinite summation (2.1) is truncated. Given an accuracy ϵ, only the first R_ϵ terms of the summation have to considered. Since the first group of Poisson probabilities are typically very small, the first L_ϵ terms can be neglected. L_ϵ and R_ϵ are called the left and right truncation point, respectively, and can be computed using the Fox-Glynn algorithm [12] as well as the Poisson probabilities. Numerically computing this summation requires R_ϵ matrix-vector multiplications. For $t > 0$, this is needed two times on different transformed CTMCs: first $\mathcal{C}[\neg\Phi \vee \Psi]$ then $\mathcal{C}[\neg\Phi]$.

2.5 Bisimulation equivalence

Lumping is a technique to aggregate the state space of a Markov chain without affecting its performance and dependability measures. It is based on the notion of ordinary lumpability [7]. A slight variant is bisimulation in which it is required in addition that bisimilar states are equally labelled [6].

2.5.1 The discrete-time setting

Definition 12. *Let $\mathcal{D} = (S, \mathbf{P}, L)$ be a DTMC and R an equivalence relation on S. R is a bisimulation on $\mathcal{D}$ if for $(s, s') \in R$:*

$$L(s) = L(s') \text{ and } q(s, C) = q(s', C) \quad \text{for all } C \in S/R,$$

where $q(s, C) = \sum_{s' \in C} \mathbf{P}(s, s') = \mathbf{P}(s, C)$. States s and s' are bisimilar if there exists a bisimulation R that contains (s, s').

Let $[s]_R \in S/R$ denote the equivalence class of s under the bisimulation relation R. For $\mathcal{D} = (S, \mathbf{P}, L)$, the lumped DTMC $\mathcal{D}/R$ is defined by $\mathcal{D}/R = (S/R, \mathbf{P}_R, L_R)$ where $\mathbf{P}_R([s]_R, C) = q(s, C)$ and $L_R([s]_r) = L(s)$. States which belong to the same equivalence class have the same cumulative probability of moving to any equivalence class: $[s]_R = [s']_R \Rightarrow \mathbf{P}_R([s]_R, C) = \mathbf{P}_R([s']_R, C)$.

In [3], it is shown that bisimulation is sound and complete with respect to pCTL*. pCTL* is an extension of PCTL. Bisimulation is also sound and complete with respect to PCTL [6]. This results in the following theorem:

Theorem 1. *Let R be a bisimulation on DTMC $\mathcal{D}$ and s be an arbitrary state in $\mathcal{D}$. Then for all PCTL formulas Φ:*

$$s \models_{\mathcal{D}} \Phi \iff [s]_R \models_{\mathcal{D}/R} \Phi$$

Hence, bisimulation preserves all PCTL formulas. Intuitively, this means every PCTL formula can be checked on the lumped DTMC $\mathcal{D}/R$ instead of on the original DTMC $\mathcal{D}$.

2.5.2 The continuous-time setting

Similar to DTMCs, a bisimulation relation can be defined for CTMCs. The difference is that bisimilar states have the same cumulative rate instead of cumulative probability.

Definition 13. *Let $\mathcal{C} = (S, \mathbf{R}, L)$ be a CTMC and R an equivalence relation on S. R is a bisimulation on $\mathcal{C}$ if for $(s, s') \in R$:*

$$L(s) = L(s') \text{ and } q(s, C) = q(s', C) \quad \text{for all } C \in S/R,$$

where $q(s, C) = \sum_{s' \in C} \mathbf{R}(s, s') = \mathbf{R}(s, C)$. States s and s' are bisimilar if there exists a bisimulation R that contains (s, s').

The notations and definitions for equivalence class and lumped CTMC are similar to the discrete-time setting.

Bisimulation equivalence for CSL is shown in [5]:

Theorem 2. *Let R be a bisimulation on CTMC $\mathcal{C}$ and s be an arbitrary state in $\mathcal{C}$. Then for all CSL formulas Φ:*

$$s \models_{\mathcal{C}} \Phi \iff [s]_R \models_{\mathcal{C}/R} \Phi$$

Hence, also every CSL formula can be checked on the lumped CTMC $\mathcal{C}/R$ instead of on the original CTMC $\mathcal{C}$.

2.6 Lumping algorithm

In [10], an algorithm is presented for the optimal lumping of CTMCs, although it can also be used for the optimal lumping of DTMCs. The algorithm constructs the coarsest lumped Markov chain of a given Markov chain. In this context, coarsest means having the fewest number of equivalence classes. It is based on the partition refinement algorithm of Paige and Tarjan for computing bisimilarity on labelled transition systems [24]. The time complexity is $O(m \log n)$, where m is the number of transitions and n

is the number of states in the Markov chain, and the space complexity is $O(m+n)$.

The algorithm is based on the notion of *splitting*. Let P be a partition of S consisting of *blocks*. Hence, a block is a set of states. Let $[s]_P$ denote the block in partition P containing state s. A *splitter* for a block $B \in P$ is a block $Sp \in P$ which satisfies:

$$\exists s_i, s_j \in B \,.\, q(s_i, Sp) \neq q(s_j, Sp) \tag{2.2}$$

In this case, B can be split into *sub-blocks* $\{B_1, \ldots, B_n\}$ satisfying:

$$\forall s_i, s_j \in B_i \,.\, q(s_i, Sp) = q(s_j, Sp)$$

$$\forall s_i \in B_i, s_j \in B_j \,.\, B_i \neq B_j \,.\, q(s_i, Sp) \neq q(s_j, Sp)$$

Intuitively, a block is split into sub-blocks in which each state has the same cumulative probability/rate to move to a state contained in the splitter.

Pseudocode of the lumping algorithm is given in Algorithm 1. It has as parameters an initial partition P and a transition matrix Q. It returns the transition matrix $Q' = Q_R$ of the lumped Markov chain. Furthermore, the initial partition is refined to the coarsest lumping partition (i. e. the final partition). In case of a DTMC $\mathcal{D} = (S, \mathbf{P}, L)$, we have $Q = \mathbf{P}$ and in case of a CTMC $\mathcal{C} = (S, \mathbf{R}, L)$, we have $Q = \mathbf{R}$. L plays the role of a list of 'potential' splitters. This list should not be confused with the labelling function. Only blocks which can split some or more block according to condition (2.2) are splitters. Initially, every block in the initial partition is considered a potential splitter. In the while loop, procedure SPLIT splits each block $B \in P$ with respect to the potential splitter from L that satisfies condition (2.2). It may also add new potential splitters to L. When L is empty, no more blocks can be split and the transition matrix Q' is constructed according to the definition of the lumped Markov chain in section 2.5.

The pseudocode for procedure SPLIT is given in Algorithm 2. It has as parameters a potential splitter Sp, the partition P and the list of potential splitters L. Line 1 initialises L' and L'' to empty sets. L' contains the set of states which have a transition to a state in Sp. L'' contains the set of blocks which have been split with respect to splitter Sp. Each state s_i has a variable $s_i.sum$ which stores the value of $q(s_i, Sp)$. If there is no transition from s_i to Sp, we have $s_i.sum = 0$. Lines 2–4 initialise these values to zero for each state which has a transition to Sp. Lines 5–8 compute these values according to the definition in section 2.5 and store the states in L'.

Each block B has a binary search tree B_T, which is called the *sub-block tree*. Each node in B_T contains the states $s \in B$ which have the same value of $q(s, Sp)$. Lines 9–13 perform the actual splitting of blocks. The list L'' is also constructed. Each state $s_i \in L'$ is removed from its original block B

and inserted into the corresponding node in the sub-block tree B_T. States which have no transition to a state in Sp will remain in B.

Lines 14–20 update the list of potential splitters L and the partition P. For each block B in L'' all sub-blocks of B are added to L except for the largest sub-block. The largest sub-block can be neglected, because its power of splitting other blocks is maintained by the remaining sub-blocks [1]. However, when the original block already was a potential splitter the largest sub-block cannot be excluded. This strategy is also used in [1]. When no states are remaining in the original block and there is only one sub-block the original block has not been split. If the original block was a potential splitter, the sub-block should also be added as a potential splitter. Line 20 adds the sub-blocks to the partition.

Splay trees

Any implementation of a binary search tree can be used as a sub-block tree. To achieve a $O(m \log n)$ time complexity *splay trees* [29] are used. A splay tree is a self-balancing binary search tree with the additional unusual property that recently accessed elements are quick to access again. It performs basic operations such as insertion, look-up and removal in $O(\log n)$ amortised time. Amortised time is the average time of an operation in a worst-case sequence of operations. All normal operations on a splay tree are combined with one basic operation, called splaying. Splaying the tree for a certain element rearranges the tree so that the element is placed at the root of the tree. This is to be done by first performing a standard binary tree search for the element in question and then use tree rotations in a specific fashion to bring the element to the top.

Algorithm 1 LUMP(P, Q)

$L :=$ blocks of P
while $L \neq \emptyset$ **do**
  $Sp :=$ POP(L)
  SPLIT(Sp, P, L)
$n' :=$ number of blocks in P
allocate $n' \times n'$ matrix Q'
initialise Q' to zero
for every block B_k of P **do**
  $s_i :=$ arbitrary state in B_k
  for every s_j such that $s_i \to s_j$ **do**
    $B_l :=$ block of s_j
    $Q'(B_k, B_l) := Q'(B_k, B_l) + Q(s_i, s_j)$
return Q'

Algorithm 2 SPLIT(Sp, P, L)

$L', L'' := \emptyset$
for every $s_j \in Sp$ **do**
  for every $s_i \to s_j$ **do**
    $s_i.sum := 0$
for every $s_j \in Sp$ **do**
  for every $s_i \to s_j$ **do**
    $s_i.sum := s_i.sum + Q(s_i, s_j)$
    $L' := L' \cup \{s_i\}$
for each $s_i \in L'$ **do**
  $B :=$ block of s_i
  delete s_i from B
  INSERT(B_T, s_i)
  $L'' := L'' \cup \{B\}$
for every $B \in L''$ **do**
  if $B \notin L$ **then**
    $B_l :=$ largest block of $\{B, B_1, \ldots, B_{|B_T|}\}$
    $L := L \cup \{B, B_1, \ldots, B_{|B_T|}\} - \{B_l\}$
  else
    $L := L \cup \{B_1, \ldots, B_{|B_T|}\}$
  $P := P \cup \{B_1, \ldots, B_{|B_T|}\}$

Chapter 3

Implementation of the lumping algorithm

This chapter describes the implementation of the algorithm for the optimal lumping Markov chains into the Markov Reward Model Checker (MRMC).

3.1 The Markov Reward Model Checker

MRMC [20] is a tool for model checking discrete-time and continuous-time Markov reward models. These models are DTMCs or CTMCs equipped with rewards and can be verified using reward extensions of PCTL and CSL. In this study, rewards are not of interest. MRMC also supports the verification of DTMCs and CTMCs without rewards using PCTL and CSL.

The tool supports an easy input format facilitating its use as a backend tool once the Markov chain has been generated. It is a command-line tool written in C and expects at least two input files: a `.tra` file describing the transition matrix and a `.lab` file indicating the state labelling with atomic propositions.

The iterative methods supported by MRMC for solving linear equation systems are the Jacobi and the Gauss-Seidel method. For unbounded until formulas (PCTL or CSL), only the Jacobi method is used. By default, MRMC uses $\epsilon = 10^{-6}$ to determine convergence (see section 2.3.2).

The transition matrix is stored in a slight variant of Compressed Row/Column representation. This sparse matrix representation only stores the non-zero elements of the matrix. Each row in the matrix is stored as a structure which contains a pointer to an array of integers, column indices, and a pointer to an array of double values which are the matrix values. These matrix values are ordered by column index. The number of non-zero elements in a

row is stored in variable *ncols*. In addition, each row has a pointer to an array (*backset*) of row indices which have a transition to this row. This array makes it possible to access the predecessors of a state easily. Self-loops (i. e. the diagonal elements) are stored in a separate variable *diag*. The example below shows a transition matrix A and its Compressed Row/Column representation.

$$A = \begin{bmatrix} 0.5 & 0.5 & 0.0 \\ 0.25 & 0.0 & 0.75 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} = \left[\begin{array}{c} ncols[0] = 1 \\ cols[0] \rightarrow [1] \\ vals[0] \rightarrow [0.5] \\ backset \rightarrow [1] \\ diag = 0.5 \\ \hline ncols[1] = 2 \\ cols[1] \rightarrow [0 \quad 2] \\ vals[1] \rightarrow [0.25 \quad 0.75] \\ backset \rightarrow [0] \\ diag = 0.0 \\ \hline ncols[2] = 0 \\ cols[2] \rightarrow NULL \\ vals[2] \rightarrow NULL \\ backset \rightarrow [1] \\ diag = 1.0 \end{array}\right]$$

3.2 Implementing the lumping algorithm

A description of the lumping algorithm and pseudocode is given in section 2.6. This section uses the same terminology.

3.2.1 Data structures

We implemented a partition as a linked list of block structures. A block has a doubly-linked list of state structures representing the states in that block and it stores the number of states. A doubly-linked list makes insertion and removal of states possible in constant time. MRMC uses bitsets to represent a set of states. A bitset is an array of integers containing at least $|S|$ bits. If bit i is set to 1, state s_i is a member of the bitset. An integer consists of $4 \cdot 8 = 32$ bits, so the number of bytes required for a bitset is $4 \cdot \lceil |S|/32 \rceil$. Using bitsets to store the states in a block requires $4 \cdot \lceil |S|/32 \rceil$ bytes for each block. When using a linked list of state structures, the number of bytes to store these states is fixed, because there is exactly one state structure for each

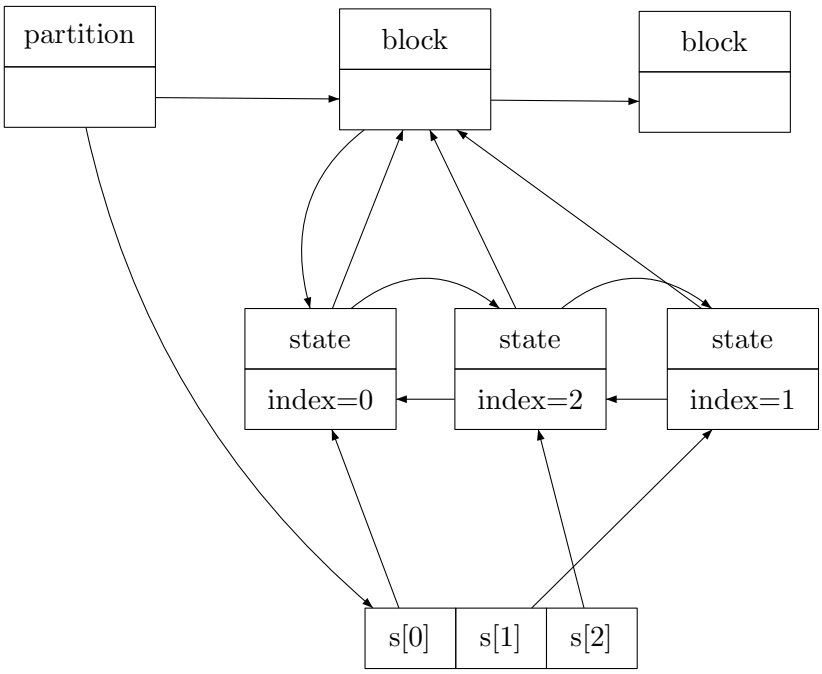

Figure 3.1: Example data structure

state. So, for large numbers of blocks using bitsets requires more memory. Therefore, we used a linked list to store the states in a block.

Each block has two flags (bits) that show its membership in L and L''. A block also has a pointer to its sub-block tree. Each state structure has a pointer to its block. The partition structure also has an array of pointers to state structures. Element $s[i]$ in this array points to the state structure of state s_i. Because a state structure has a pointer to its block, this array makes it easy to access the block of a state.

In figure 3.1 an example is given of a data structure of blocks and states in a partition. A box denotes a structure and an arrow denotes a pointer to a structure. The variables contained in the structures are not shown, except the state index. For sake of readability, only the states contained in the first block are shown.

3.2.2 The initial partition

States in each equivalence class (block) under bisimulation agree on their atomic propositions. Thus, states which have the same combination of atomic propositions should be put into the same block in the initial partition P:

$$\forall s_i, s_j \in B \,.\, L(s_i) = L(s_j) \quad \text{for all } B \in P$$

The number of different combinations of atomic propositions is $2^{|AP|}$. Obviously, the initial partition cannot contain more than $|S|$ blocks.

To determine in which block a state should be put, we used a binary search tree with depth $|AP|$. For each state s_i, we start at the root of this tree. If the first atomic proposition is valid in s_i, we move to the left subtree, otherwise we move to the right subtree. This procedure is repeated for the each atomic proposition until a leaf node is reached. This leaf node has a pointer to a block in which s_i should be put. The tree can be constructed while putting states in the initial partition. So, it is not necessary to build the entire tree in advance. Nodes in the tree which are never accessed are not constructed.

Figure 3.2 shows an example of such a tree. There are two atomic propositions a and b. The node $b \wedge \neg a$ does not exists. So, in this example no state is labelled with $b \wedge \neg a$.

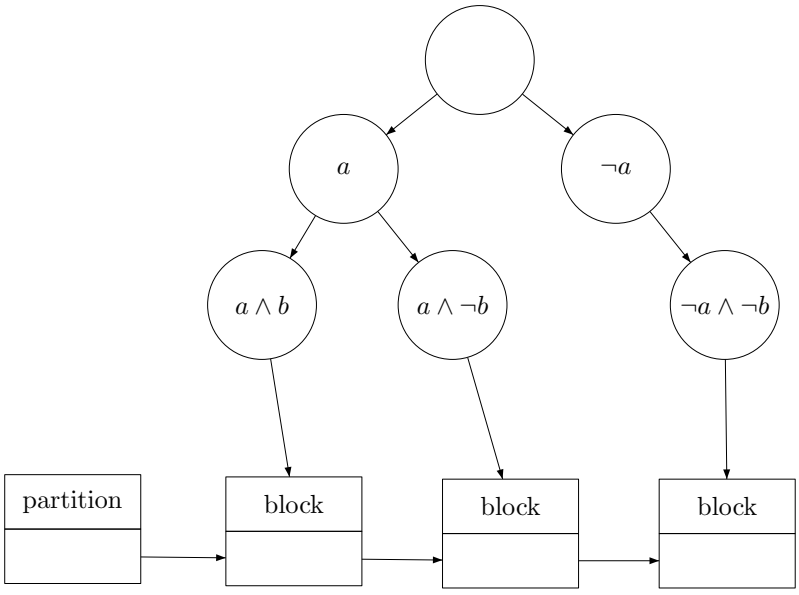

Figure 3.2: Example binary search tree used for creating the initial partition

3.2.3 **Procedure** LUMP

Line 1 (see Algorithm 1 on page 27) of LUMP initialises L. This set is implemented as a linked list. Every item in this list has a pointer to a block. Line 5 counts the number of blocks in the final partition. In the implementation every block is assigned a unique number which corresponds to its row index in the lumped transition matrix. Line 9 chooses an arbitrary state from a block. Our implementation simply takes the first state. Since some model checking algorithms of MRMC require the matrix values to be ordered by column index, each row (i. e. the arrays *cols* and *vals*) of the lumped transition matrix is sorted after it has been filled completely. This is done using an slightly adapted version of quicksort [1].

3.2.4 **Procedure** SPLIT

L' of the SPLIT procedure stores the set of states that have a transition to a state in Sp. It is implemented as a global integer array of size $|S|$. The state indices j of states s_j are stored in this array. A variable to maintain the number of used elements is incremented every time an element is added at line 8. Each state s_i is appended to L' once, if the old value of $q(s_i, Sp)$ is zero.

The values of the cumulative function q are stored in a global array $sum[\,]$. Element $sum[i]$ in this array stores $q(s_i, Sp)$. Lines 2–4 initialises these values to zero for states which have a transition to a state in S. This can be replaced by setting $q(s_i, Sp)$ to zero after state s_i has been inserted into the sub-block tree on line 12. This is allowed because $q(s_i, Sp)$ is not used again after the insert into the tree. The array then only has to be initialised to zero before the first call to SPLIT. This is much faster than iterating through all predecessors.

Because MRMC uses a sparse matrix representation to store the transition matrix line 7 cannot be implemented to take constant time. The row elements are ordered by column so a binary search can be used to access $Q(s_i, s_j)$. This takes $O(\log n)$ time, where n is the number of successor states of s_i, i. e. the number of non-zero elements in row i.

The sub-block tree is implemented as a splay tree. Each tree node contains a pointer to a block structure, which is a sub-block of the original block. A tree node also contains a key equal to $q(s, Sp)$, where s is a state contained in that sub-block. Every time a state is deleted from its original block and inserted into a sub-block, the number of states in the original block and the sub-block is updated. For this state, also the pointer to its block is updated. We used the splay tree implementation from Daniel Sleator's website [1].

[1] http://www.link.cs.cmu.edu/splay/

Lines 14–20 update the list of potential splitters. For each block $B \in L''$ the sub-blocks are added to the list of potential splitters. If B is not (yet) a potential splitter, the largest sub-block can be neglected. Each block has a flag to show its membership in L, which makes it is easy to determine if B is already a potential splitter.

At the end of each call to SPLIT each sub-block tree is destroyed and the sub-blocks are added to the partition. Keeping the sub-blocks in the tree can cause states in the same sub-block with different total outgoing rates/probabilities to another block.

Chapter 4

Bisimulation minimisation and PCTL model checking

4.1 Introduction

This chapter describes experiments to study the effectiveness of bisimulation minimisation for PCTL model checking. We used several case studies from the PRISM website [26]. In these case studies, a probabilistic model of an algorithm or protocol is defined. The probabilistic model checker PRISM [22] is then used to check whether certain PCTL properties hold.

In this study, we used PRISM to build and export a DTMC for these probabilistic models. Using MRMC, we minimised this original model to compute a lumped model. The implementation of the lumping algorithm is described in the previous chapter. When creating the initial partition, only atomic propositions contained in the PCTL property were considered. After lumping, the labelling function was modified such that it corresponded to the lumped DTMC. In our experiments, the time to check the property on the original DTMC is compared to the time to lump and check the property on the lumped DTMC.

For each case study a short description will follow. Then the PCTL properties are explained and finally the results are presented. These results include:

- the number of states and transitions in the original DTMC representing the model;
- the number of blocks (i. e. states) in the lumped DTMC;
- lump equals the time (in milliseconds) to construct the initial partition and lumping the DTMC;

- MC equals the time (in milliseconds) to check the PCTL property;
- the reduction factor of the state space;
- the reduction factor of the runtime (i. e. checking the original DTMC divided by lumping plus checking the lumped DTMC).

Also the time complexity of the lumping algorithm, $O(m \log n)$, where n is the number of states and m is the number of transition in the DTMC, is compared to the actual runtime.

All experiments were conducted on an Intel Pentium 4 2.66 GHz with 1 GB RAM running Linux.

4.2 PCTL properties

To study the effectiveness of bisimulation minimisation for PCTL model checking it is important which kind of properties to consider. Assuming states are labelled with Φ and Ψ model checking $\neg\Phi$, $\Phi \wedge \Psi$, $\Phi \vee \Psi$ and $X\,\Phi$ is straightforward and not computationally expensive. This leaves the bounded and unbounded until operators.

The algorithm for model checking bounded until operators is given in section 2.3.2. The state probabilities are calculated in t iterations. Hence, increasing the bound t yields a longer computation time. Therefore, a realistic time bound with respect to the case study under consideration should be chosen. The worst-case time complexity of model checking a bounded until operator is $O(t \cdot (m + n))$ [13].

Section 2.3.2 describes the algorithm for model checking unbounded until operators. The worst-case time complexity is $O(n^3)$ [5]. Using a backward search, the set of states is partitioned into three subsets subsets U_s, U_f and U_i. If U_i is empty, no linear equation system has to be solved because the solution is already given. U_i is empty if for every state in S_i either no path reaches a state in S_s or all paths reach a state in S_s. For these kind of properties, it is not likely that bisimulation minimisation takes less time than model checking the original DTMC. Therefore, unbounded until properties for which $U_i = \emptyset$ are not considered.

Compared to the time complexity of bisimulation minimisation ($O(m \log n)$), bounded and unbounded until properties are the most interesting properties to consider.

4.3 Case studies

4.3.1 Synchronous Leader Election Protocol

This case study is based on the synchronous leader election protocol in [19]. Given a synchronous ring of N processors, the protocol will elect a leader (a uniquely designated processor) by sending messages around the ring. The protocol proceeds in rounds and is parametrised by a constant $K > 0$. Each round begins by all processors (independently) choosing a random number (uniformly) from $\{1, \ldots, K\}$ as an id. The processors then pass their selected id to all other processors around the ring. If there is a unique id, then the processor with the maximum unique id is elected as the leader, and otherwise all processors begin a new round. The ring is synchronous: there is a global clock and at every time slot a processor reads the message that was sent at the previous time slot (if it exists), makes at most one state transition and then may send at most one message. Each processor knows N.

Properties

The expected number of rounds L to elect a leader depends on N and K. For both $N = 4$ and $N = 5$, we have $L \leq 3$. The number of steps per round is $N + 1$. This corresponds to selecting a random id (one step), and passing it around through the entire ring. In our experiments, the probability of electing a leader within three rounds has been calculated. This can be expressed in PCTL by the path formula:

$$true\, \mathcal{U}^{\leq 3\cdot(N+1)}\, elected$$

Since there is only one atomic proposition, the number of blocks in the initial partition is two: a block for states which are labelled with *elected* and a block for states which are not labelled.

Results

Tables 4.1 and 4.2 show statistics and results for different values of N and K.

For a given N, the number of blocks in the final partition is independent of K. Only one state is labelled with atomic proposition *elected*. This is also the only absorbing state. Many paths starting in the initial state eventually reach this absorbing state. No branching occurs on these paths: each state on such a path has exactly one transition to another state and no transitions to other states. The only branching occurs in the initial state. As K grows, the number of paths reaching the absorbing state also grows. However, the

$N = 4$	original DTMC			lumped DTMC			reduct. factor	
K	states	transitions	MC	blocks	lump	MC	states	time
2	55	70	0.02	10	0.05	0.01	5.5	0.4
4	782	1037	0.4	10	0.5	0.01	78.2	0.8
6	3902	5197	1.8	10	2.1	0.01	390.2	0.9
8	12302	16397	7.0	10	9.0	0.01	1230.2	0.8
10	30014	40013	19	10	25	0.01	3001.4	0.8
12	62222	82957	41	10	52	0.01	6222.2	0.8
14	115262	153677	85	10	100	0.01	11526.2	0.8
16	196622	262157	165	10	175	0.01	19662.2	0.9

Table 4.1: Bisimulation minimisation results for 4 processors

$N = 5$	original DTMC			lumped DTMC			reduct. factor	
K	states	transitions	MC	blocks	lump	MC	states	time
2	162	193	0.1	12	0.1	0.02	13.5	0.9
4	5122	6145	2.8	12	2.9	0.02	426.8	0.9
6	38882	46657	28	12	26	0.02	3240.2	1.1
8	163842	196609	140	12	115	0.02	13653.5	1.2

Table 4.2: Bisimulation minimisation results for 5 processors

length of these paths remains equal. Therefore, all states on these paths at an equal distance from the absorbing state are bisimilar. This explains the constant number of blocks for fixed N. Figure 4.1 shows an example of this situation. States in a dashed box belong to the same equivalence class. State s_{21} is labelled with *elected*.

In most cases, the time to construct the lumped DTMC exceeds the time to model check the original DTMC. The initial state is the only state which has more than one outgoing transition. Thus, only one row in the transition matrix has more than one non-zero element. Since the transition matrix is implemented as a sparse matrix, this results in a relatively low number of multiplications in each iteration when calculating the bounded until property. However, for $N = 5$ and $K = 8$, model checking the original DTMC takes longer than lumping plus model checking the minimised DTMC. In this case the number of states and transitions is less than for example $N = 4$ and $K = 16$, but the bound in the until property is higher, which results in more iterations and therefore a longer computation time.

To compare the actual runtime of the lumping algorithm to its time complexity, the value c has been calculated, where $l = c\, m \log n$ (l denotes the lumping time). For most cases, this results in a nearly constant value of $c \approx 40$. From time complexity theory, we have $cm \log n \in O(m \log n)$. Thus, in this case, the actual runtime is strongly related to the time complexity.

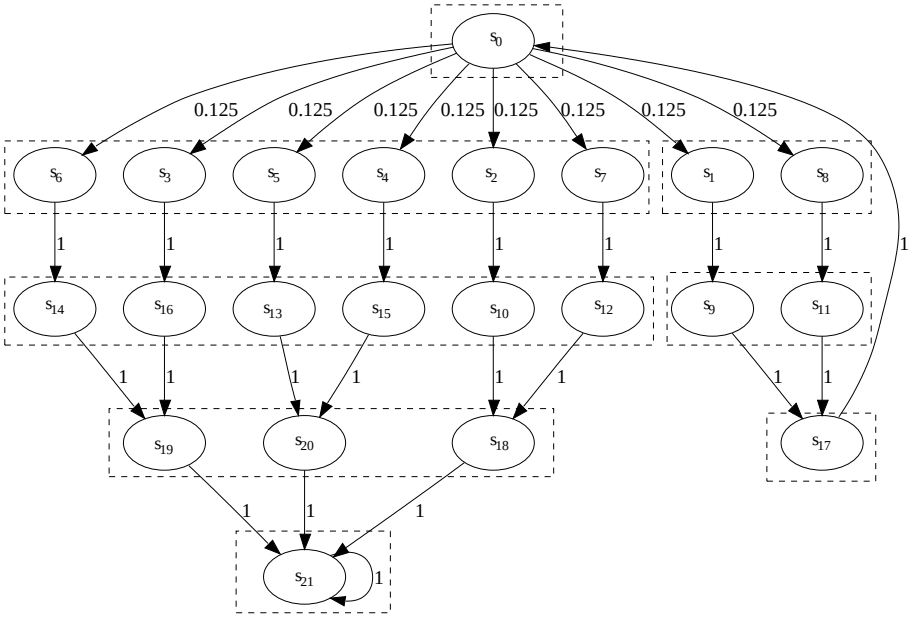

Figure 4.1: Example DTMC for $N = 3$ and $K = 2$

4.3.2 Randomised Self-stabilisation

A self-stabilising protocol for a network of processes is a protocol which, when started from some possibly illegal start state, returns to a legal/stable state without any outside intervention within some finite number of steps.

This case study considers Herman's self stabilising algorithm [16]. The protocol operates synchronously and communication is unidirectional in the ring. In this protocol, the number of processes N in the ring must be odd. The stable states are those where there is exactly one process which possesses a token.

Each process in the ring has a local Boolean variable xi, and there is a token at position i if $xi = x(i-1)$. In a basic step of the protocol, if the current values of xi and $x(i-1)$ are equal, then it makes a (uniform) random choice as to the next value of xi, and otherwise it sets it equal to the current value of $x(i-1)$.

Properties

The expected time to reach a stable state is $N^2/2$ time units [16]. A stable state is a state in which only one process possesses a token. The probability of reaching a stable state within the expected time has been calculated.

Expressed in PCTL by the path formula:

$$true\, \mathcal{U}^{\leq N^2/2}\, stable$$

In the initial partition the number of states labelled *stable* is equal to N.

Results

Table 4.3 shows statistics and results for different number of processes N.

	original DTMC			lumped DTMC			reduct. factor	
N	states	transitions	MC	blocks	lump	MC	states	time
3	8	28	0.01	2	0.02	0.01	4.0	0.3
5	32	244	0.02	4	0.06	0.01	8.0	0.3
7	128	2188	0.2	9	0.5	0.01	14.2	0.4
9	512	19684	2.2	23	5.2	0.05	22.3	0.4
11	2048	177148	50.5	63	105	0.4	32.5	0.5
13	8192	1594324	613	190	1700	3.6	43.1	0.3
15	32768	14348908	7600	612	28000	77	53.5	0.3

Table 4.3: Bisimulation minimisation results for $true\, \mathcal{U}^{\leq N^2/2}\, stable$

We observe that the state space reductions improve with an increase of N. Model checking the original DTMC takes much less time than lumping the DTMC. This can be explained by the fact that the number of transitions is very high compared to the number of states. This makes computing the q value in the lumping algorithm a time consuming procedure, because this value cannot be accessed in constant time (see section 3.2.4).

Similar to the leader election case study, we calculated the value c, where $l = c\, m \log n$. For this case study, c is not constant. As N grows, c seems to grow linearly. Hence, there is a close resemblance between the time complexity and the actual runtime.

4.3.3 Crowds Protocol

The Crowds protocol was developed by Reiter and Rubin [27] to provide users with a mechanism for anonymous Web browsing. The main idea behind Crowds and similar approaches to anonymity is to hide each user's communications by routing them randomly within a group of similar users. Even if a local eavesdropper or a corrupt group member observes a message being sent by a particular user, it can never be sure whether the user is the actual sender, or is simply routing another user's message.

It is assumed that corrupt routers are only capable of observing their local networks. The adversary's observations are thus limited to the apparent source of the message. As the message travels down a (randomly constructed) routing path from its real sender to the destination, the adversary observes it only if at least one of the corrupt members was selected among the routers. The only information available to the adversary is the identity of the crowd member immediately preceding the first corrupt member on the path. It is also assumed that communication between any two crowd members is encrypted by a pairwise symmetric key.

Crowds is designed to provide anonymity for message senders. Under a specific condition on system parameters, Crowds provably guarantees the following property for each routing path: The real sender appears no more likely to be the originator of the message than to not be the originator.

Routing paths in Crowds are set up using the following protocol:

- The sender selects a crowd member at random (possibly itself), and forwards the message to it, encrypted by the corresponding pairwise key.
- The selected router flips a biased coin. With probability $1-pf$, where pf (forwarding probability) is a parameter of the system, it delivers the message directly to the destination. With probability pf, it selects a crowd member at random (possibly itself) as the next router in the path, and forwards the message to it, re-encrypted with the appropriate pairwise key. The next router then repeats this step.

The path from a particular source to a particular destination is set up only once, when the first message is sent. The routers maintain a persistent id for each constructed path, and all subsequent messages follow the established path.

There is no bound on the maximum length of the routing path. For simplicity, instead of modelling each corrupt crowd member separately, a single adversary is modeled who is selected as a router with a fixed probability equal to the sum of selection probabilities of all corrupt members.

Properties

Atomic proposition $observe_i$ denotes the adversary observed crowd member i more than once (i. e. at least twice). Crowd member 0 is the real sender. The following PCTL properties are used to analyse anonymity protection provided by Crowds in the multiple-paths case:

- Eventually the adversary observed the real sender more than once:

$$true\,\mathcal{U}\,observe_0$$

- Eventually the adversary observed someone other than the real sender more than once:

$$true\,\mathcal{U}\,observe,$$

where $observe \equiv \bigvee_{i=1}^{N} observe_i$.

Results

Tables 4.4 and 4.5 show statistics and results for both properties. N is the actual number of honest crowd members and R is the total number of protocol runs to analyse. For $N = 5$ the number of corrupt crowd members is 1, for $N = 10$ the number is 2, for $N = 15$ the number is 3 and for $N = 20$ there are 4 corrupt crowd members.

		original DTMC			lumped DTMC			reduct. factor	
N	R	states	transitions	MC	blocks	lump	MC	states	time
5	3	1198	2038	0.2	41	0.5	0.2	29.2	2.8
5	4	3515	6035	8.2	61	1.7	0.3	57.6	4.1
5	5	8653	14953	35.5	81	4.8	0.5	106.8	6.7
5	6	18817	32677	117	101	11.3	0.6	186.3	9.8
10	3	6563	15143	10.0	41	3.3	0.2	160.1	2.9
10	4	30070	70110	10.5	61	19.5	0.3	493.0	5.3
10	5	111294	261444	480	81	81	0.6	1374.0	5.9
10	6	352535	833015	1770	101	280	0.6	3490.0	6.3
15	3	19228	55948	42	41	16	0.2	469.0	2.6
15	4	119800	352260	355	61	120	0.3	1963.9	3.0
15	5	592060	1754860	2100	81	650	0.4	7309.4	3.2
15	6	2464168	7347928	12100	101	2900	0.6	24397.7	4.2
20	3	42318	148578	93	41	48	0.2	1032.1	1.9
20	4	333455	1183535	890	61	418	0.3	5466.5	2.1
20	5	2061951	7374951	7700	81	2700	0.5	25456.2	2.9

Table 4.4: Bisimulation minimisation results for $true\,\mathcal{U}\,observe_0$

Note that, for both properties, the state space reductions improve with an increase of R and fixed N. Also, lumping plus model checking the lumped DTMC is significantly faster than model checking the original DTMC. The number of iterations to solve the equation systems for both properties varies between 100 and 160, depending on the parameters N and R. This makes solving such a system computationally quite expensive.

		original DTMC			lumped DTMC			reduct. factor	
N	R	states	transitions	MC	blocks	lump	MC	states	time
5	3	1198	2038	3.2	53	0.6	0.3	22.6	3.7
5	4	3515	6035	11	97	2.0	0.5	36.2	4.4
5	5	8653	14953	48	153	6.0	0.9	56.6	6.9
5	6	18817	32677	139	209	14	1.4	90.0	9.0
10	3	6563	15143	24	53	4.6	0.2	123.8	4.9
10	4	30070	70110	190	97	29	0.5	310.0	6.4
10	5	111294	261444	780	153	127	0.9	727.4	6.1
10	6	352535	833015	2640	221	400	1.4	1595.2	6.6
15	3	19228	55948	102	53	23	0.2	362.8	4.4
15	4	119800	352260	790	97	190	0.5	1235.1	4.1
15	5	592060	1754860	4670	153	1020	0.9	3869.7	4.6
15	6	2464168	7347928	20600	221	4180	1.5	11150.1	4.9
20	3	42318	148578	240	53	75	0.2	798.5	3.2
20	4	333455	1183535	2750	97	710	0.5	3437.7	3.9
20	5	2061951	7374951	18600	153	4550	0.9	13476.8	4.1

Table 4.5: Bisimulation minimisation results for $true\,\mathcal{U}\,observe$

Similar to the leader election case study, the value c has been calculated, where $l = c\ m \log n$. For the first property, we have $c \in [17, 25]$, and for the second property $c \in [24, 33]$. Thus, the time complexity is closely related to the actual runtime.

4.3.4 Randomised Mutual Exclusion

This case study is based on Pnueli and Zuck's solution [25] to the well-known mutual exclusion problem. Let $P_1 \ldots P_N$ be N processes that from time to time need to execute a critical section in which at most one of them is allowed. The processes can coordinate their activities by use of a common resource. This solution guarantees at any time t there is at most one process in its critical section phase. It also guarantees if a process tries, then eventually it enters the critical section.

Properties

The state probabilities for P_1 entering the critical section first have been calculated. This can be expressed in PCTL by the path formula:

$$notEnter_1\,\mathcal{U}\,enter_1$$

Where $enter_i$ means P_i enters the critical section and $notEnter_i \equiv \bigwedge_{j \neq i}^{N} \neg enter_j$.

Combining several atomic propositions into one atomic proposition generally yields a coarser final partition.

Results

Table 4.6 shows statistics and results for N processes.

	original DTMC			lumped DTMC			reduct. factor	
N	states	transitions	MC	blocks	lump	MC	states	time
3	2368	8272	3.0	1123	8.0	1.6	2.1	0.3
4	27600	123883	78	5224	192	19	5.3	0.4
5	308800	1680086	980	18501	2880	120	16.7	0.3
6	3377344	21514489	11700	-	$> 10^7$	-	-	-

Table 4.6: Bisimulation minimisation results for $notEnter_1 \,\mathcal{U}\, enter_1$

Lumping the DTMC takes significantly more time than model checking the original DTMC. The number of transitions in the original DTMC is relatively high, making lumping more computationally expensive. On the other hand, the number of iterations to solve the linear equation system is quite low. These numbers vary between 60 and 80, depending on the number of processes.

For $N = 6$, lumping the DTMC was not completed within several hours. To compare the time complexity to the actual runtime of the lumping algorithm, more results have to be available.

4.4 Conclusion

The effectiveness of bisimulation minimisation for PCTL model checking has been studied. The case studies on synchronous leader election and randomised self-stabilisation have been used to check bounded until properties. For some configurations of the synchronous leader election protocol, lumping plus model checking the minimised DTMC takes less time than model checking the original DTMC. This suggests there is a lower time bound above which bisimulation minimisation is effective. However, it is hard to predict this bound exactly for concrete cases. For all other configurations as well as the randomised self-stabilisation protocol, model checking the original DTMC takes less time than lumping and model checking the lumped DTMC.

The case studies on randomised mutual exclusion and the Crowds protocol have been used to check unbounded until properties. Lumping plus model checking the lumped DTMC is faster than model checking the original DTMC for the Crowds protocol. For randomised mutual exclusion,

this is not the case. Randomised mutual exclusion requires a relatively low number of iterations to solve the linear equation system in comparison to the Crowds protocol. This could explain the fact that bisimulation minimisation is effective for the Crowds protocol, but not for randomised mutual exclusion. It should be noted that MRMC implements the Jacobi method to solve the linear equation system for PCTL unbounded until formulas. This method generally converges slower than for example the Gauss-Seidel method [30]. Using the Gauss-Seidel method could improve the runtime of model checking unbounded until properties.

The time complexity of the lumping algorithm is $O(m \log n)$, where n is the number of states and m is the number of transition in the DTMC. Except for the mutual exclusion case study, this time complexity has been compared to the actual runtime. For most cases, the actual runtime is closely related to the time complexity.

The experiments have shown that in some cases bisimulation minimisation is effective for PCTL properties with (at least) one until operator. For most cases it was not effective. This does not imply that bisimulation minimisation can play no role in PCTL model checking. It is possible to check several properties on a lumped DTMC. To do this, the DTMC has to be lumped by considering the atomic propositions contained in all properties to be checked. These properties can then be checked on a (possibly) much smaller DTMC and thus require less computation time.

Predicting in advance whether bisimulation minimisation is effective for PCTL model checking is not easy. For a parametrised model (e. g. the synchronous leader election protocol with N processors), a strategy could be to consider small cases first. If bisimulation minimisation is effective for these cases, it is likely to be effective for larger cases.

The experiments also showed that lumping can reduce the number of states by several orders of magnitude. Huge reductions for DTMCs consisting of several millions of states have been accomplished. Model checking the lumped DTMC is much faster than model checking the original DTMC. It can be concluded that bisimulation minimisation is a good state space reduction technique for DTMCs.

Chapter 5

Formula-dependent lumping for PCTL model checking

5.1 Introduction

This chapter considers formula-dependent lumping for PCTL until formulas, which might lead to more important state space reductions. This can be explained using the notion of F bisimulation [5]. Instead of labelling states with atomic propositions, each state is labelled with formulas from a set F that are valid in that state. So, the bisimulation relation in chapter 4 can be viewed as an AP bisimulation, which is essentially formula-independent lumping.

First, we present F bisimulation and bisimulation equivalence for PCTL. Then, we describe which PCTL properties are checked and how we can lump the DTMC for these properties. Finally, the case studies and results are presented.

5.2 Bisimulation equivalence

In [5], F bisimulation and bisimulation equivalence is defined for CTMCs and CSL. We define this for DTMCs and PCTL in a similar way.

Definition 14. *Let $\mathcal{D} = (S, \mathbf{P}, L)$ be a DTMC, F a set of PCTL formulas, and R an equivalence relation on S. R is an F bisimulation on $\mathcal{D}$ if for $(s, s') \in R$:*

$$L_F(s) = L_F(s') \text{ and } \mathbf{P}(s, C) = \mathbf{P}(s', C) \quad \text{for all } C \in S/R,$$

where $L_F(s) = \{\Phi \in F \mid s \models \Phi\}$. States s and s' are F bisimilar if there exists an F bisimulation R that contains (s, s').

Definition 15. *Let $PCTL_F$ denote the smallest set of PCTL formulas that includes F and is closed under all PCTL operators.*

Theorem 3. *Let R be an F bisimulation on DTMC $\mathcal{D} = (S, \mathbf{P}, L)$ and s and s' states in S. Then,*

1. *For all $PCTL_F$ formulas Φ, $(s, s') \in R \Rightarrow \; s \models_{\mathcal{D}} \Phi \leftrightarrow s' \models_{\mathcal{D}} \Phi$.*
2. *For all $PCTL_F$ path formulas ϕ,*

$$(s, s') \in R \Rightarrow \Pr(\{\sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models_{\mathcal{D}} \phi\}) = \Pr(\{\sigma \in Path^{\mathcal{D}}(s') \mid \sigma \models_{\mathcal{D}} \phi\}).$$

Proof. The proof is adapted from [3] and goes by induction on the length of the formula, where formulas in F are of length 1. To avoid problems with until formulas, we transform formulas of the form $\Phi \, \mathcal{U}^{\leq t} \, \Psi$ such that $\Phi \cap \Psi = \emptyset$. This can be done easily, as $\Phi \, \mathcal{U}^{\leq t} \, \Psi \Leftrightarrow (\Phi \wedge \neg\Psi) \, \mathcal{U}^{\leq t} \, \Psi$.

The only state formulas of length 1 are the formulas in F. By definition of F bisimulation, the labels of bisimilar states agree. Therefore, for state formulas of length 1 the theorem holds. For path formala of length 1, this is trivial.

The induction hypothesis is: The theorem holds for all state formulas of length at most k and for all path formulas of length at most k.

Let the induction hypothesis hold for formulas of length at most k. We start with the proof for state formulas. Let Φ be a state formula of length $k+1$.

- $\Phi = \neg\Phi_1$ or $\Phi = \Phi_1 \wedge \Phi_2$, where Φ_1 and Φ_2 are state formulas: Follows directly from the induction hypothesis.
- $\Phi = \mathcal{P}_{\trianglelefteq p}(\phi)$, where ϕ is a path formula. From the induction hypothesis, we know that the probability measure of the set of states starting in s which satisfy ϕ equals the probability measure of the set of states starting in s' which satisfy ϕ. Thus, $s \models_{\mathcal{D}} \mathcal{P}_{\trianglelefteq p}(\phi) \leftrightarrow s' \models_{\mathcal{D}} \mathcal{P}_{\trianglelefteq p}(\phi)$

Now, we show the proof for path formulas. Let ϕ be a path formula of length $k+1$.

- $\phi = X \, \Phi$, where Φ is state formula. By definition,

$$\Pr(\{\sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models X \, \Phi\}) = \sum_{s' \models_{\mathcal{D}} \Phi} \mathbf{P}(s, s').$$

  According to the induction hypothesis, bisimilar states agree on the same state formulas. Since states in $C \in S/R$ are bisimilar, we can rewrite this as:

$$\Pr(\{\sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models X \, \Phi\}) = \sum_{C \models_{\mathcal{D}/R} \Phi} \mathbf{P}(s, C)$$

By definition of F bisimulation, bisimilar states s and s' have the same cumulative probability of moving to any other equivalence class: $\forall C \in S/R\ \mathbf{P}(s, C) = \mathbf{P}(s', C)$. Therefore, the probability measures are equal:

$$\Pr(\{\sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models_{\mathcal{D}} X\,\Phi\}) = \Pr(\{\sigma \in Path^{\mathcal{D}}(s') \mid \sigma \models_{\mathcal{D}} X\,\Phi\})$$

- $\phi = \Phi\,\mathcal{U}^{\leq t}\,\Psi$, where Φ and Ψ are state formulas. We define the set of paths B_n^s starting in s that satisfy ϕ after exactly n steps as:

$$B_n^s = \{\sigma \in Path^{\mathcal{D}}(s) \mid \forall\ 0 \leq i < n(\sigma[i] \models_{\mathcal{D}} \Phi) \wedge \sigma[n] \models_{\mathcal{D}} \Psi\}$$

Since the sets B_i^s are disjoint, the probability measure of paths starting in s is now given by:

$$\Pr(\{\sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models_{\mathcal{D}} \phi\}) = \sum_{i=0}^{t} \Pr(B_i^s)$$

By induction on i it can be seen that $\forall i\ \Pr(B_i^s) = \Pr(B_i^{s'})$ [3]. From this follows $\sum_{i=0}^{t} \Pr(B_i^s) = \sum_{i=0}^{t} \Pr(B_i^{s'})$. Thus:

$$\Pr(\{\sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models_{\mathcal{D}} \phi\}) = \Pr(\{\sigma \in Path^{\mathcal{D}}(s') \mid \sigma \models_{\mathcal{D}} \phi\})$$

- $\phi = \Phi\,\mathcal{U}\,\Psi$, where Φ and Ψ are state formulas. This formula is equivalent to $\Phi\,\mathcal{U}^{\leq \infty}\,\Psi$. Thus, the probability measure is given by:

$$\Pr(\{\sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models_{\mathcal{D}} \phi\}) = \sum_{i=0}^{\infty} \Pr(B_n^s)$$

Note that the sum converges as Pr is a probability measure. Similar to the bounded until case, we have $\sum_{i=0}^{\infty} \Pr(B_i^s) = \sum_{i=0}^{\infty} \Pr(B_i^{s'})$ [3]. Therefore, we can conclude:

$$\Pr(\{\sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models_{\mathcal{D}} \phi\}) = \Pr(\{\sigma \in Path^{\mathcal{D}}(s') \mid \sigma \models_{\mathcal{D}} \phi\})$$

We can conclude that the theorem holds for any PCTL_F formula of length $k > 0$. □

Thus, we can check each PCTL formula $\Phi \in \mathrm{PCTL}_F$ on the lumped DTMC $\mathcal{D}/R$ instead of on the original DTMC $\mathcal{D}$, where R is a F bisimulation.

5.3 PCTL properties

To check bounded until formulas $\phi = \mathcal{P}_{\trianglelefteq p}(\Phi\ \mathcal{U}^{\leq t}\ \Psi)$, the set of states S is partitioned into three subsets S_s, S_f and S_i (see section 2.3.2):

$$\begin{aligned} S_s &= \{s \in S \mid s \models \Psi\} \\ S_f &= \{s \in S \mid s \models \neg\Phi \wedge \neg\Psi\} \\ S_i &= \{s \in S \mid s \models \Phi \wedge \neg\Psi\} \end{aligned}$$

Formula ϕ can be checked using an F bisimulation with $F = \{\Psi, \neg\Phi \wedge \neg\Psi, \Phi \wedge \neg\Psi\}$. First, we have to show that $\phi \in \mathrm{PCTL}_F$. It is easy to see that $\psi = \mathcal{P}_{\trianglelefteq p}((\Phi \wedge \neg\Psi)\ \mathcal{U}^{\leq t}\ \Psi) \in \mathrm{PCTL}_F$. Now, we show that $\phi \Leftrightarrow \psi$. It is sufficient to show that both formulas agree on their subsets S_s, S_f and S_i. This is easy to see for S_s and S_i. For formula ψ we have:

$$\begin{aligned} S_f &= Sat(\neg(\Phi \wedge \neg\Psi) \wedge \neg\Psi) \\ &= Sat((\neg\Phi \vee \Psi) \wedge \neg\Psi) \\ &= Sat((\neg\Phi \wedge \neg\Psi) \vee (\Psi \wedge \neg\Psi)) \\ &= Sat(\neg\Phi \wedge \neg\Psi) \end{aligned}$$

Therefore, both formulas agree on S_s, S_f and S_i. Because $\phi \Leftrightarrow \psi$, we can also check ϕ using the F bisimulation mentioned above.

The initial partition corresponding to $F = \{\Psi, \neg\Phi \wedge \neg\Psi, \Phi \wedge \neg\Psi\}$ is $P = \{S_s, S_f, S_i\}$. It is possible to optimise by collapsing the states in S_s and S_f into two single absorbing states s_s and s_f, respectively. Then, we can lump with initial partition $P' = \{\{s_s\}, \{s_f\}, S_i\}$. In general, lumping with P' instead of P yields a coarser partition. To avoid the construction of transition matrix $\mathbf{P}''$ in which states from S_s and S_f are collapsed into absorbing states s_s and s_f, the lumping algorithm has been modified to be able to omit blocks from partitioning. Blocks which are omitted cannot be split, but will still be considered as potential splitters. To omit a block B, lines 2–8 of the SPLIT procedure of the lumping algorithm have been modified to skip states in B. In this case, blocks S_s and S_f are to be omitted while lumping with P. This saves time and memory, because $\mathbf{P}''$ does not need to be constructed explicitly.

In case of formula $\mathcal{P}_{\trianglelefteq p}(true\ \mathcal{U}^{\leq t}\ \Psi)$, the set S_f is empty yielding initial partition $P = \{S_s, S \setminus S_s\}$. This is essentially an AP bisimulation, which is covered in chapter 4. A slight difference is that Ψ states can be collapsed into a single absorbing state prior to lumping. We do not consider these kinds of formulas here. Therefore, most case studies from chapter 4 cannot be used.

The unbounded until formula $\mathcal{P}_{\trianglelefteq p}(\Phi\ \mathcal{U}\ \Psi)$ can also be checked using an F bisimulation. Two initial partition are possible: $P = \{S_s, S_f, S_i\}$ and

$P_U = \{U_s, U_f, U_i\}$. Sets U_f and U_s are constructed by extending S_f and S_s which implies $U_i \subseteq S_i$ (see section 2.3.2). Similar to formula-dependent lumping for bounded until formulas, it is possible to optimise by collapsing states in U_s and U_f into two single absorbing states u_s and u_f, respectively: $P'_U = \{\{u_s\}, \{u_f\}, U_i\}$. Set U_i may contain significantly less states than S_i. Since U_f and U_s can be omitted, less states have to be considered for partitioning in case of P'_U. Therefore, we will use P'_U as initial partition. States in U_s satisfy $\mathcal{P}_{\geq 1}(\Phi\, \mathcal{U}\, \Psi)$, states in U_f satisfy $\mathcal{P}_{\leq 0}(\Phi\, \mathcal{U}\, \Psi)$ and states in U_i satisfy $\mathcal{P}_{<1}(\Phi\, \mathcal{U}\, \Psi) \wedge \mathcal{P}_{>0}(\Phi\, \mathcal{U}\, \Psi)$. So, we have an F bisimulation, where $F = \{\mathcal{P}_{\geq 1}(\Phi\, \mathcal{U}\, \Psi), \mathcal{P}_{\leq 0}(\Phi\, \mathcal{U}\, \Psi), \mathcal{P}_{<1}(\Phi\, \mathcal{U}\, \Psi) \wedge \mathcal{P}_{>0}(\Phi\, \mathcal{U}\, \Psi)\}$. Similar to the bounded until case, it can be seen that $\exists \psi \in \mathrm{PCTL}_F$. $\mathcal{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}\, \Psi) \Leftrightarrow \psi$:

$$\begin{aligned} \psi &= \mathcal{P}_{\trianglelefteq p}\Big(\big(\mathcal{P}_{<1}(\Phi\, \mathcal{U}\, \Psi) \wedge \mathcal{P}_{>0}(\Phi\, \mathcal{U}\, \Psi)\big)\, \mathcal{U}\, \mathcal{P}_{\geq 1}(\Phi\, \mathcal{U}\, \Psi)\Big) \\ &\Leftrightarrow \mathcal{P}_{\trianglelefteq p}\Big(\big(\Phi \wedge \neg\Psi\big)\, \mathcal{U}\, \Psi\Big) \\ &\Leftrightarrow \mathcal{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}\, \Psi) \end{aligned}$$

It is easy to see that $\psi \in \mathrm{PCTL}_F$, so we can check $\mathcal{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}\, \Psi)$ using the F bisimulation.

A possible optimisation when checking bounded until formulas is to partition the state space into subsets S_s, U_f and $S \setminus (S_s \cup U_f)$. There exists no path from a state in U_f to a state in S_s. Therefore, for any state $s \in U_f$ we have $s \models \mathcal{P}_{\leq 0}(\Phi\, \mathcal{U}^{\leq t}\, \Psi)$. Thus, states in U_f can be made absorbing, which could yield less computation time. We cannot use the U_s instead of S_s, because states in U_s satisfy $\mathcal{P}_{\geq 1}(\Phi\, \mathcal{U}\, \Psi)$ but not necessarily $\mathcal{P}_{\geq 1}(\Phi\, \mathcal{U}^{\leq t}\, \Psi)$. This approach can also be used when formula-dependent lumping for bounded until formulas. Then we take as initial partition $P = \{S_s, U_f, S \setminus (S_s \cup U_f)\}$. In the following, we check bounded until formulas for the workstation cluster and cyclic server polling system case studies. These DTMCs are *irreducible*, i. e. for every pair of states s and s' there exists a path from s to s' and a path from s' to s. Hence, for these case studies we have $U_f = S_f$. Therefore, this approach is not used in this chapter.

5.4 Case studies

Like in chapter 4, several case studies have been used to study the effectiveness of formula-dependent lumping for PCTL model checking. The difference is that the DTMC is not lumped a priori by only considering atomic propositions, but lumping is integrated into the model checking procedures of until operators. For each until operator contained in the property to check, the DTMC is lumped using an initial partition mentioned above. Therefore, instead of presenting the lumping time, the computation time

of the adapted model checking procedure is presented (in milliseconds), denoted lump+MC.

5.4.1 Randomised Mutual Exclusion

See section 4.3.4 for a description of this case study and the PCTL formula.

Results

Tables 5.1 show the results of formula-dependent lumping for N processes.

	original DTMC			lumped DTMC		reduct. factor	
N	states	transitions	MC	blocks	lump+MC	states	time
3	2368	8272	3.0	233	3.2	10.2	0.9
4	27600	123883	78	785	48	35.2	1.6
5	308800	1680086	980	2159	680	143.0	1.6
6	3377344	21514489	11700	554135	39800	6.1	0.3

Table 5.1: Results for $notEnter_1 \,\mathcal{U}\, enter_1$

Comparing these results with table 4.6 shows the number of blocks is significantly less in case of formula-dependent lumping. For $N = 4$ and $N = 5$, formula-dependent lumping plus model checking the lumped DTMC is also faster than model checking the original DTMC as well as formula-independent lumping plus model checking the lumped DTMC. For $N = 6$, formula-dependent lumping completed within a reasonable amount of time in contrast to formula-independent lumping (see section 4.3.4).

5.4.2 Workstation Cluster

This case study considers a dependable cluster of workstations [15]. Two sub-cluster are connected via a backbone connection. Each sub-cluster consists of N workstations, connected in a star topology with a central switch that provides the interface to the backbone. Each of the components of the system (workstation, switch, backbone) can break down. There is a single repair unit that takes care of repairing failed components. The model is represented as a CTMC.

Properties

To check PCTL properties on the workstation cluster, the uniformised DTMC of the CTMC is constructed. The time to construct this uniformised DTMC is not considered in the results below. Atomic propositions *minimum* and

premium are defined. Proposition *minimum* corresponds to the minimum quality of service (QoS) provided: at least $\frac{3}{4}N$ workstations have to be operational and these workstations have to be connected to each other via operational switches. For *premium* quality of service at least N workstations have to be operational, with the same connectivity constraints. N is the number of workstations in each sub-cluster, so the total number of workstations is $2N$.

The following PCTL path formulas are checked:

- Within k steps the QoS is turned from *minimum* to *premium*:

$$minimum\, \mathcal{U}^{\leq k}\, premium$$

  We take the bound to be $k = q \cdot t$, where q is the uniformisation rate. This value $q \cdot t$ is equal to the expected number of steps in the uniformised DTMC corresponding to the state probabilities at time t, where t is a time instant in the original CTMC. The uniformisation rate is 51 and $t = 10$ is a reasonable amount of time in the CTMC of the workstation cluster. So, we have $k = 510$.

- The QoS turns from *minimum* to *premium*:

$$minimum\, \mathcal{U}\, premium$$

Results

Tables 5.2 and 5.3 show the results of formula-dependent lumping for N workstations. The last column of table 5.2 shows the number of blocks when using an AP bisimulation like we did in chapter 4. Obviously, for the second property these numbers are the same, since we are lumping with respect to the same atomic propositions.

	original DTMC			lumped DTMC		reduct. factor		$F = AP$
N	states	transitions	MC	blocks	lump+MC	states	time	blocks
2	276	1396	1.8	30	0.5	9.2	3.6	147
4	820	4436	4.6	52	1.1	15.8	4.2	425
8	2772	15604	17	239	5.7	11.6	3.0	1413
16	10132	58292	140	917	29	11.0	4.8	5117
32	38676	225076	603	3599	170	10.7	3.6	19437
64	151060	884276	2340	14267	960	10.6	2.4	75725
128	597012	3505204	9250	56819	5400	10.5	1.7	298893
256	2373652	13957172	38100	226787	36600	10.5	1.1	1187597

Table 5.2: Results for $minimum\, \mathcal{U}^{\leq 510}\, premium$

	original DTMC			lumped DTMC		reduct. factor	
N	states	transitions	MC	blocks	lump+MC	states	time
2	276	1396	0.2	30	0.3	9.2	0.67
4	820	4436	0.5	52	0.7	15.8	0.71
8	2772	15604	2.1	239	3.3	11.6	0.64
16	10132	58292	20.5	917	21.0	11.0	0.98
32	38676	225076	114	3599	124	10.7	0.92
64	151060	884276	570	14267	750	10.6	0.76
128	597012	3505204	3850	56819	4870	10.5	0.79
256	2373652	13957172	33000	226787	37200	10.5	0.89

Table 5.3: Results for $minimum\,\mathcal{U}\,premium$

Table 5.2 shows formula-dependent lumping yields a significantly smaller DTMC than formula-independent lumping. Note that the reduction factor of the state space is quite constant. Formula-dependent lumping plus model checking the lumped DTMC for the first property is faster than checking the original DTMC. For the second property, checking the original DTMC is slightly faster. This can be explained by means of the number of iterations to solve the linear equation system. As N increases, the number of iterations also increases. This number is at most 325, for $N = 256$. Thus, the number of iterations is significantly lower than k. Hence, the computation time of the unbounded until property is less.

We observe that, for a given N, the number of blocks for both properties is equal. So unfortunately, lumping with initial partition $P'_U = \{\{u_s\}, \{u_f\}, U_i\}$ does not lead to a coarser partition than lumping with initial partition $P' = \{\{s_s\}, \{s_f\}, S_i\}$.

5.4.3 Cyclic Server Polling System

This case study is based on a cyclic server polling system [18]. It consists of one polling server which handles N stations. Each station has a buffer. The polling server polls the stations in a cyclic manner. If a station has a full buffer, the station is served by the server. After this, the buffer is empty again and the server continues polling the stations. The model is represented as a CTMC.

Properties

To check PCTL properties on the cyclic server polling system, the uniformised DTMC of the CTMC is constructed. The time to construct this uniformised DTMC is not considered in the results below.

The following PCTL path formulas are checked:

- Within k steps station 1 is served before all other stations:

$$notServe_1 \, \mathcal{U}^{\leq t} \, serve_1$$

  We take $k = q \cdot t$, like in the previous case study. The uniformisation rate is 202 and $t = 5$, so we have $k = 1010$.

- Station 1 is served before all other stations:

$$notServe_1 \, \mathcal{U} \, serve_1$$

Atomic propositions $notServe_i$ is defined as: $notServe_i \equiv \bigwedge_{j \neq i}^{N} \neg serve_j$.

In the original case study, the authors check the formula $\neg serve_2 \, \mathcal{U} \, serve_1$. We find the formula $notServe_1 \, \mathcal{U} \, serve_1$ more intuitive, since this expresses whether station 1 is served first, instead of only earlier than station 2.

Results

Tables 5.4 and 5.5 show the results of formula-dependent lumping for N stations.

	original DTMC			lumped DTMC		reduct. factor	
N	states	transitions	MC	blocks	lump+MC	states	time
4	96	368	1.4	19	0.4	5.1	3.5
5	240	1040	4.2	26	0.7	9.2	6.0
6	576	2784	10	34	1.2	16.9	8.3
7	1344	7168	25	43	2.0	31.3	12.5
8	3072	17920	62	53	4.0	58.0	15.5
9	6912	43776	190	64	9.4	108.0	20.2
10	15360	104960	575	76	22	202.1	26.1
11	33792	247808	1310	89	51	379.7	25.7
12	73728	577536	3050	103	120	715.8	25.4
13	159744	1331200	7250	118	287	1353.8	25.3
14	344064	3039232	16900	134	730	2567.6	23.2
15	737280	6881280	39000	151	1590	4882.6	24.5

Table 5.4: Results for $notServe_1 \, \mathcal{U}^{\leq 1010} \, serve_1$

Formula-independent lumping does not lead to any state space reductions. Then, the number of blocks in the final partition equals the number of states of the original DTMC. So, each block contains only one state. In contrast, formula-dependent lumping does lead to significant reductions. Note that the state space reductions improve with an increase of N.

	original DTMC			lumped DTMC		reduct. factor	
N	states	transitions	MC	blocks	lump+MC	states	time
4	96	368	2.1	12	0.9	8	2.3
5	240	1040	4.9	15	1.1	16	4.5
6	576	2784	11	18	1.4	32	7.9
7	1344	7168	24	21	2.1	64	11.4
8	3072	17920	52	24	2.9	128	17.9
9	6912	43776	180	27	5.7	256	31.6
10	15360	104960	610	30	11	512	55.5
11	33792	247808	1580	33	25	1024	63.2
12	73728	577536	3460	36	55	2048	62.9
13	159744	1331200	7300	39	119	4096	61.3
14	344064	3039232	15400	42	260	8192	59.2
15	737280	6881280	32100	45	580	16384	55.3

Table 5.5: Results for $notServe_1 \,\mathcal{U}\, serve_1$

Also, the number of blocks in case of the second property is significantly less compared to the first property. So, for this case study, lumping with initial partition $P'_U = \{\{u_s\}, \{u_f\}, U_i\}$ does lead to a significantly coarser partition than lumping with initial partition $P' = \{\{s_s\}, \{s_f\}, S_i\}$. Finally, for both properties, formula-dependent lumping plus model checking the lumped DTMC is faster than model checking the original DTMC.

5.5 Conclusion

The effectiveness of formula-dependent lumping for PCTL bounded and unbounded until formulas has been studied. An advantage of formula-dependent lumping is that some blocks in the initial partition can be omitted yielding a shorter lumping time. A disadvantage is that the lumped DTMC cannot be reused for model checking formulas in which the until formula is not contained. The DTMC is lumped with respect to a specific until formula. This is the main difference with formula-independent lumping.

Three case studies have been used: randomised mutual exclusion, a workstation cluster and a cyclic server polling system. It was shown that formula-dependent lumping can lead to large state space reductions. As expected, formula-dependent lumping can lead to a significantly smaller state space than formula-independent lumping. For the cyclic server polling system, formula-independent lumping does not lead to a state space reduction at all, whereas formula-dependent lumping does.

For several cases, formula-dependent lumping plus model checking the lumped DTMC takes significantly less time than model checking the original DTMC.

Chapter 6

Bisimulation minimisation and CSL model checking

6.1 Introduction

This chapter considers bisimulation minimisation and CSL model checking. Bisimulation minimisation may be even more effective for CSL than for PCTL model checking. For very large models, it could be that the original CTMC can be fully represented in memory, but that model checking cannot be carried out because of excessive memory requirements (e. g. constructing the uniformised DTMC). Lumping could require less memory than model checking the original CTMC. In this case, model checking is possible on the lumped CTMC.

Similar to chapter 4, we used several case studies from the PRISM website, but now we are checking CSL properties on CTMC case studies. The implementation of the LUMP procedure has been slightly modified to be able to use a more efficient sparse matrix data structure. The lumping algorithm has not been changed though, since it can be used for lumping DTMCs as well as CTMCs. We present the results in the same manner as in chapter 4.

We also compare bisimulation minimisation to symmetry reduction using a case study. This technique will be explained in section 6.3.

6.2 CSL properties

Any CSL formula that does not refer to time can be checked using PCTL model checking techniques. However, in this chapter we are interested in CSL model checking on CTMCs. Therefore, we do not consider unbounded

until formulas in this chapter. Checking $X^I\,\Phi$ formulas is straightforward and requires only one matrix-vector multiplication.

For time-bounded until formulas of the form $\mathcal{P}_{\trianglelefteq p}(\Phi\,\mathcal{U}^{[t,t']}\,\Psi)$, two cases can be distinguished: $t = 0$ and $t > 0$. For $t > 0$, transient analysis on different uniformised Markov chains is needed twice, instead of once in case of $t = 0$. The overall time complexity of model checking bounded until formulas is $O(m \cdot q \cdot t')$, where m is the number of transitions in the CTMC and q is the uniformisation rate. For model checking steady-state operators, the time complexity is $O(n^3)$, where n is the number of states in the CTMC.

Bisimulation minimisation takes $O(m \log n)$ time. Compared to the time complexity of model checking time-bounded until and steady-state properties, these properties are the most interesting to study the effectiveness of bisimulation minimisation on CSL model checking. The $\mathcal{U}^{[t,t']}$ operator with $t > 0$ is especially interesting, because it is more complex and time consuming to check than the case where $t = 0$.

6.3 Symmetry reduction

In [23], an approach is presented for applying symmetry reduction to probabilistic model checking in PRISM. Similar to bisimulation minimisation, symmetry reduction allows verification to be performed on a reduced model instead of the original model. The difference is that bisimulation minimisation can be applied to any model, whereas symmetry reduction can only be applied to symmetric models. The peer-to-peer case study from [23] is used to compare these techniques.

Symmetry reduction is a way of exploiting the presence of replication in a model. It can be applied to models consisting of N symmetric components of which any pair can be exchanged without any effect on the behaviour of the system (i. e. the parallel composition of N identical components). The global state s of the model consists of the local states s_i of the N components, $s = (s_1, \cdots, s_N)$. Each global state has a unique representative. States having the same representative are equivalent and will be collapsed into one single state. For instance, consider the case of four symmetric processes, each with two local states A and B. The states (A, B, A, A) and (A, A, B, A) are equivalent, since their representative is $(A = 3, B = 1)$. The reduced model is constructed from the symbolic representation (i. e. using MTBDDs) of the original model. Provided that atomic propositions are preserved, model checking can be done safely on the reduced model.

6.4 Case studies

6.4.1 Workstation Cluster

See section 5.4.2 for a description of this case study and the atomic propositions used in the CSL properties.

Properties

The following CSL properties have been checked:

- In the long run, premium QoS will be delivered with a probability of at least 0.7:
$$\mathcal{S}_{>0.7}(premium)$$
- The probability that QoS drops below minimum quality within 85 time units is less than 0.1:
$$\mathcal{P}_{<0.1}(true\, \mathcal{U}^{\leq 85}\, \neg minimum)$$
- The probability of going from minimum QoS to premium QoS within 20 to 40 time units without violating the minimum QoS constraint along the way is less than 0.99:
$$\mathcal{P}_{<0.99}(minimum\, \mathcal{U}^{[20,40]}\, premium)$$

Results

Table 6.1 shows model checking the lumped CTMC takes more time than model checking the original CTMC.

	original CTMC			lumped CTMC			reduct. factor	
N	states	transitions	MC	blocks	lump	MC	states	time
2	276	1120	0.3	147	0.9	0.2	1.9	0.3
4	820	3616	1.0	425	2.8	21	1.9	0.04
8	2772	12832	3.6	1413	12	130	2.0	0.03
16	10132	48160	21	5117	64	770	2.0	0.03
32	38676	186400	114	19437	290	215	2.0	0.2
64	151060	733216	730	75725	1360	1670	2.0	0.2
128	597012	2908192	6500	298893	5900	14900	2.0	0.3
256	2373652	11583520	103000	1187597	25400	175000	2.0	0.5

Table 6.1: Bisimulation minimisation results for $\mathcal{S}_{>0.7}(premium)$

Checking steady-state operators requires solving a linear equation system for each BSCC (see section 2.4.2). In this case, there is a single BSCC consisting of all states. MRMC uses the Gauss-Seidel method or the Jacobi method to solve this linear equation system. When using the Jacobi method to solve the linear equation system, it does not converge within 1.000.000 iterations. For this specific property, the Gauss-Seidel method converges significantly slower for the lumped CTMC.

This can be explained by means of the subdominant eigenvalue λ_2 of the Gauss-Seidel iteration matrix. We can write $\mathbf{Q}^T = \mathbf{D} - (\mathbf{L} + \mathbf{U})$ with $\mathbf{Q} = \mathbf{R} - diag(\underline{E})$. $\mathbf{D}$ denotes the diagonal of $\mathbf{Q}^T$ and $\mathbf{L}$ and $\mathbf{U}$ denote the lower and upper triangular half, respectively. Now, the Gauss-Seidel iteration matrix is given by $(\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}$. The closer λ_2 is to $\lambda_1 = 1$, the slower the convergence [14].

Using Maple, we were able to compute these eigenvalues for $N \leq 8$. These values are presented in table 6.2 as well as the number of Gauss-Seidel iterations NoI. We see that λ_2 is significantly closer to 1 in case of the lumped CTMC. This could explain the slow convergence. Note that for $N = 2$, the subdominant eigenvalue of the lumped CTMC is not that close to 1. The difference in number of iterations is also less than for the other cases.

Note that the time to check the lumped CTMC for $N = 32$ is less than the time to check the lumped CTMC for $N = 16$. This is because the number of iterations executed to solve the linear equation system is significantly larger for $N = 16$.

	original CTMC		lumped CTMC	
N	NoI	λ_2	NoI	λ_2
2	9	0.0665	12	0.111
4	10	0.0954	818	0.987
8	11	0.156	1520	0.993
16	14	-	2182	-
32	21	-	101	-

Table 6.2: Subdominant eigenvalues and number of Gauss-Seidel iterations

We also observe that for $N < 128$, the lumping time alone exceeds the time to check the original CTMC.

For the two time-bounded until properties, in almost all cases, lumping plus model checking the lumped CTMC is faster than model checking the original CTMC (see tables 6.3 and 6.4). The only exception is $N = 256$ in case of the third property. Unfortunately, we could not check the case $N = 512$ because model checking the corresponding sparse matrix requires too much memory.

	original CTMC			lumped CTMC			reduct. factor	
N	states	transitions	MC	blocks	lump	MC	states	time
2	276	1120	18.0	147	0.9	9.2	1.9	1.8
4	820	3616	34.0	425	2.8	19.0	1.9	1.6
8	2772	12832	104	1413	12.5	56.0	2.0	1.5
16	10132	48160	1000	5117	69	220	2.0	3.5
32	38676	186400	4200	19437	324	2190	2.0	1.7
64	151060	733216	16400	75725	1460	9900	2.0	1.4
128	597012	2908192	68000	298893	6830	48300	2.0	1.2
256	2373652	11583520	279000	1187597	33200	227000	2.0	1.1

Table 6.3: Bisimulation minimisation results for $\mathcal{P}_{<0.1}(true\, \mathcal{U}^{\leq 85}\, \neg minimum)$

	original CTMC			lumped CTMC			reduct. factor	
N	states	transitions	MC	blocks	lump	MC	states	time
2	276	1120	7.2	147	0.9	3.7	1.9	1.6
4	820	3616	14.7	425	2.8	7.7	1.9	1.4
8	2772	12832	49.0	1413	12.7	25.8	2.0	1.3
16	10132	48160	470	5117	69	100	2.0	2.8
32	38676	186400	2190	19437	328	1130	2.0	1.5
64	151060	733216	8550	75725	1480	5170	2.0	1.3
128	597012	2908192	35200	298893	6860	24700	2.0	1.1
256	2373652	11583520	144000	1187597	33400	116000	2.0	0.9

Table 6.4: Bisimulation minimisation results for $\mathcal{P}_{<0.99}(minimum\, \mathcal{U}^{[20,40]}\, premium)$

Note that, although using three different sets of atomic propositions, the number of blocks in the lumped CTMC is equal for all three properties. Also, the state space reduction factor is quite constant as N increases.

6.4.2 Cyclic Server Polling System

See section 5.4.3 for a description of this case study.

Properties

The following CSL properties have been checked:

- Station 1 will be polled within t time units with probability $\trianglelefteq p$:

$$\mathcal{P}_{\trianglelefteq p}(true\, \mathcal{U}^{\leq t}\, poll_1)$$

- In the long run, station 1 is waiting for the server with probability $\trianglelefteq p$:

$$\mathcal{S}_{\trianglelefteq p}(waiting_1)$$

- In the long run, station 1 is idle with probability $\trianglelefteq p$:

$$\mathcal{S}_{\trianglelefteq p}(idle_1)$$

- Station 1 is served before station 2 with probability $\trianglelefteq p$:

$$\mathcal{P}_{\trianglelefteq p}(\neg serve_2\, \mathcal{U}\, serve_1)$$

- All stations will be full within $[t, t']$ time units with probability $\trianglelefteq p$:

$$\mathcal{P}_{\trianglelefteq p}(true\, \mathcal{U}^{[t,t']}\, full)$$

Results

Except for the last property, lumping the CTMC did not result in any reductions. The number of blocks in the final partition equals the number of states in the original CTMC. So, each block contains only one state. This is also the case for the property in section 5.4.3.

Table 6.5 shows statistics and results for the last CSL property. We observe that the number of blocks in the lumped CTMC is significantly less than the number of states in the original CTMC. Lumping without considering the state labelling (i. e. with initial partition $P = \{S\}$) results in the same number of blocks. Note that the state space reduction factor slightly increases as N increases. Also, the lumping time plus the model checking time on the lumped CTMC is significantly less than the time to check the original CTMC.

	original CTMC			lumped CTMC			reduct. factor	
N	states	transitions	MC	blocks	lump	MC	states	time
4	96	272	5.5	24	0.2	1.5	4	3.3
5	240	800	16.0	48	0.4	3.1	5	4.6
6	576	2208	41.0	96	1.1	6.5	6	5.4
7	1344	5824	99.0	192	3.2	14.5	7	5.6
8	3072	14848	250	384	10	32	8	6.0
9	6912	36864	820	768	42	62	9	7.9
10	15360	89600	2090	1536	135	128	10	7.9
11	33792	214016	4700	3072	396	280	11	7.0
12	73728	503808	10900	6144	1080	770	12	5.9
13	159744	1171456	26100	12288	2870	1910	13	5.5
14	344064	2695168	64500	24576	7660	5800	14	4.8
15	737280	6144000	148000	49152	19100	16400	15	4.2

Table 6.5: Bisimulation minimisation results for $\mathcal{P}_{\trianglelefteq p}(true\,\mathcal{U}^{[5,10]}\,full)$

6.4.3 Tandem Queueing Network

This case study is based on a CTMC model of a tandem queueing network presented in [17]. The network consists of two sequentially composed queues, both having the same capacity.

Properties

The following CSL properties are interesting to verify:

- The tandem network becomes fully occupied within t time units with probability $\trianglelefteq p$:
$$\mathcal{P}_{\trianglelefteq p}(true\,\mathcal{U}^{\leq t}\,full)$$
- Queue i of the tandem network becomes fully occupied within t time units with probability $\trianglelefteq p$:
$$\mathcal{P}_{\trianglelefteq p}(true\,\mathcal{U}^{\leq t}\,full_i)$$
- The probability of leaving a situation where the second queue is fully occupied within t time units meets the bound $\trianglelefteq p$:
$$\mathcal{P}_{\trianglelefteq p}(full_2\,\mathcal{U}^{\leq t}\,\neg full_2)$$

Results

Lumping the CTMC for the CSL properties mentioned above did not result in a reduction of the CTMC. Even without considering the state labelling, the number of blocks in the final partition equals the number of states of the original CTMC. A possible explanation could be different exit rates. As $\mathbf{R}(s, C) = \mathbf{P}(s, C) \cdot E(s)$, the condition on the cumulative rates (see section 2.5) can be reformulated as:

$$\mathbf{P}(s, C) = \mathbf{P}(s', C) \text{ for all } C \in S/R \quad \text{and} \quad E(s) = E(s')$$

Thus, bisimilar states must have the same exit rate. If all states would have a different exit rate, then no pair of states can be bisimilar. However, this is not the case for the CTMC of the tandem queueing network.

In the embedded DTMC of the CTMC the exit rates are not of concern, because $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ for all states s. So, lumping the embedded DTMC only considers the cumulative *probabilities*. However, when lumping the embedded DTMC for this case study, the number of blocks in the final partition also equals the original number of states. This shows that different exit rates are not the problem when lumping the CTMC of the tandem queueing network.

6.4.4 Simple Peer-To-Peer Protocol

This case study describes a simple peer-to-peer (P2P) protocol based on BitTorrent. The model contains a set of clients trying to download a file that has been partitioned into K blocks. Initially, there is one client that has already obtained all of the blocks and N additional clients with no blocks. Each client can download a block from any of the others, but they can only attempt four concurrent downloads for each block.

In [23], this case study is used to illustrate the performance of symmetry reduction. To compare bisimulation minimisation and symmetry reduction, we check the same CSL property.

Property

The probability that all clients have downloaded all blocks within 0.5 time-units has been calculated. This is expressed by the CSL path formula:

$$true \, \mathcal{U}^{\leq 0.5} \, done$$

Results

We considered the cases $2 \leq N \leq 4$, whereas in [23] they consider the cases $4 \leq N \leq 7$. This is because for $N > 4$, the transition matrix becomes too large for PRISM to export. We used the prototype extension of PRISM with symmetry reduction to reduce the state space for these cases.

The last three columns of table 6.6 show the size of the symmetry-reduced state space, the time to construct this reduced state space on our machine (in milliseconds), and the time to check the reduced state space using MRMC.

	original CTMC		lumped CTMC			symm. reduced CTMC		
N	states	MC	blocks	lump	MC	states	reduce	MC
2	1024	5.6	56	1.4	0.3	528	12	2.9
3	32768	410	252	170	1.3	5984	100	59
4	1048576	22000	792	10200	1080	52360	360	820

Table 6.6: Bisimulation minimisation and symmetry reduction results

We observe that bisimulation minimisation results in larger state space reductions than symmetry reduction. However, the time to construct the symmetry-reduced state space is significantly less than lumping as N increases. This is to be expected, since the lumping algorithm is far more complex. The results also show lumping plus model checking the lumped CTMC takes significantly less time than model checking the original CTMC.

6.5 Conclusion

To study the effectiveness of bisimulation minimisation on CSL model checking, we considered several well-known case studies from literature: a workstation cluster, a cyclic server polling system and a tandem queueing network.

For the workstation cluster, we checked a steady state property and two time-bounded until properties. For the polling system, we checked one time-bounded until property. Model checking the steady state property on the lumped CTMC takes more time than model checking the original CTMC. The linear equation system corresponding to the steady state probabilities on the lumped CTMC converges significantly slower. More details can be found in section 6.4.1. For the time-bounded until properties, lumping plus model checking the lumped CTMC takes less time in almost all cases.

For certain other properties for the polling system as well as the tandem queueing network, bisimulation minimisation did not lead to a reduction of the CTMC. These results show bisimulation minimisation is not always a good state space reduction technique.

To conclude, bisimulation minimisation can be effective for model checking CSL, but it very much depends on the specific model and properties under consideration. Like for DTMCs, it is also possible to construct a lumped CTMC on which several properties can be checked by considering all atomic propositions contained in these properties. Then, these properties can be checked on a (possibly) much smaller CTMC.

Chapter 7

Formula-dependent lumping for CSL model checking

7.1 Introduction

Similar to chapter 5, we can lump the CTMC for a specific CSL formula using an F bisimulation. First, we present the definition of F bisimulation and bisimulation equivalence for CSL presented in [5]. Then, we describe which CSL properties are checked and how the CTMC is lumped for these properties. Finally, we present the case studies and the results. The modified lumping algorithm from chapter 5 was used to study the effectiveness of formula-dependent lumping for CSL model checking.

7.2 Bisimulation equivalence

We recall the definitions of F bisimulation and CSL_F formulas from [5].

Definition 16. *Let $\mathcal{C} = (S, \mathbf{R}, L)$ be a CTMC, F a set of CSL formulas, and R an equivalence relation on S. R is an F bisimulation on $\mathcal{C}$ if for $(s, s') \in R$:*

$$L_F(s) = L_F(s') \text{ and } \mathbf{R}(s, C) = \mathbf{R}(s', C) \quad \text{for all } C \in S/R,$$

where $L_F(s) = \{\Phi \in F \mid s \models \Phi\}$. States s and s' are F bisimilar if there exists an F bisimulation R that contains (s, s').

Definition 17. *Let CSL_F denote the smallest set of CSL formulas that includes F and is closed under all CSL operators.*

The following theorem allows us to check CSL_F formulas on the lumped CTMC $\mathcal{C}/R$ instead of on the original CTMC $\mathcal{C}$, where R is an F bisimulation.

Theorem 4 ([5]). *Let R be an F bisimulation on CTMC $\mathcal{C} = (S, \mathbf{R}, L)$ and s a state in S. Then,*

1. *For all CSL_F formulas Φ, $s \models_{\mathcal{C}} \Phi \leftrightarrow [s]_R \models_{\mathcal{C}/R} \Phi$.*

2. *For all CSL_F path formulas ϕ, $Prob^{\mathcal{C}}(s, \phi) = Prob^{\mathcal{C}/R}([s]_R, \phi)$.*

In particular, F bisimilar states satisfy the same CSL_F formulas.

7.3 CSL properties

We only consider time-bounded until formulas. Unbounded until formulas are not considered, because the model checking procedure is exactly as for PCTL. In chapter 5, some PCTL unbounded until formulas are checked on the uniformised DTMC of a CTMC. Since CSL unbounded until formulas can be checked on the embedded DTMC as well as the uniformised DTMC, formula-dependent lumping for these CSL unbounded until formulas will lead to similar results. Formulas of the form $true\, \mathcal{U}^I\, \Psi$ are also not considered.

For formulas of the form $\mathcal{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}^{[t,t']}\, \Psi)$, we distinguish two cases: $t = 0$ and $t > 0$. For the case $t = 0$, we need to identify Ψ states and $(\neg\Phi \vee \Psi)$ states. By definition, the formula is satisfied in Ψ states and the $(\neg\Phi \vee \Psi)$ states can be made absorbing (see section 2.4.2). As $Sat(\neg\Phi \vee \Psi) = Sat(\Psi) \cup Sat(\neg\Phi \wedge \neg\Psi)$, we can use an F bisimulation with $F = \{\Psi, \neg\Phi \wedge \neg\Psi, \Phi \wedge \neg\Psi\}$. Similar to PCTL bounded until formulas, it is easy to see that $\mathcal{P}_{\trianglelefteq p}(\Phi\, \mathcal{U}^{[0,t']}\, \Psi) \Leftrightarrow \mathcal{P}_{\trianglelefteq p}((\Phi \wedge \neg\Psi)\, \mathcal{U}^{[0,t']}\, \Psi) \in \mathrm{CSL}_F$ (see section 5.3). Thus, we can check this formula using the F bisimulation mentioned above. Like in chapter 5, prior to lumping we collapse Ψ states and $(\neg\Phi \wedge \neg\Psi)$ states into absorbing states s_s and s_f, respectively. So, we will use $P = \{Sat(\Phi \wedge \neg\Psi), \{s_s\}, \{s_f\}\}$ as initial partition.

This F bisimulation cannot be used without modification for the case $t > 0$. As can be seen in section 2.4.2, the computation is performed on two different (transformed) Markov chains. We need to identify all Φ states which is not possible with this F bisimulation, because the $(\Phi \wedge \Psi)$ states are included in the set of Ψ states. The probability measure of the set of $(\Phi \wedge \Psi)$ states is needed in the first part of the computation (i. e. the second sum). Thus, we distinguish $(\Phi \wedge \Psi)$ and $(\neg\Phi \wedge \Psi)$ states. In both parts of the computation $\neg\Phi$ states can be made absorbing. We have $Sat(\neg\Phi) = Sat(\neg\Phi \wedge \Psi) \cup Sat(\neg\Phi \wedge \neg\Psi)$. Let R be a $\{\Phi \wedge \neg\Psi, \Phi \wedge \Psi, \neg\Phi \wedge \Psi, \neg\Phi \wedge \neg\Psi\}$ bisimulation.

Then, can write this more formally:

$$
\begin{aligned}
&Prob^{\mathcal{C}}(s, \Phi\, \mathcal{U}^{[t,t']}\, \Psi) \\
&= \sum_{s' \vDash \Phi} \left(\pi^{\mathcal{C}[\neg\Phi]}(s, s', t) \cdot \sum_{s'' \vDash \Psi} \pi^{\mathcal{C}[\neg\Phi \vee \Psi]}(s', s'', t' - t) \right) \\
&= \sum_{[s']_R \vDash \Phi} \left(\pi^{\mathcal{C}[\neg\Phi]/R}([s]_R, [s']_R, t) \cdot \sum_{[s'']_R \vDash \Psi} \pi^{(\mathcal{C}[\neg\Phi]/R)[\Psi]}([s']_R, [s'']_R, t' - t) \right)
\end{aligned}
$$

We optimise by collapsing $(\neg\Phi \wedge \Psi)$ states and $(\neg\Phi \wedge \neg\Psi)$ states into two absorbing states $s_{s'}$ and s_f, respectively. Thus, the initial partition is $P = \{Sat(\Phi \wedge \neg\Psi), Sat(\Phi \wedge \Psi), \{s_{s'}\}, \{s_f\}\}$. Only for the first part of the computation, after lumping we make $(\Phi \wedge \Psi)$ states absorbing additionally.

Similar to formula-dependent lumping for PCTL, we can optimise by disregarding Φ states which never reach a Ψ state. Then, we make states satisfying $\mathcal{P}_{\leq 0}(\Phi\ \mathcal{U}\ \Psi)$ absorbing instead of $(\neg\Phi \wedge \neg\Psi)$ states. This optimisation can lead to more absorbing states yielding less computation time. However, the case studies we consider all have irreducible CTMCs. Hence, for these CTMCs we have $Sat(\mathcal{P}_{\leq 0}(\Phi\ \mathcal{U}\ \Psi)) = Sat(\neg\Phi \wedge \neg\Psi)$. So, we will not use this optimisation.

7.4 Case studies

7.4.1 Workstation Cluster

See section 5.4.2 for a description of this case study and the atomic propositions used in the CSL properties.

Properties

The following CSL properties have been checked:

- The probability of going from minimum QoS to premium QoS within 40 time units without violating the minimum QoS constraint along the way is less than 0.99:

$$\mathcal{P}_{<0.99}(minimum\, \mathcal{U}^{\leq 40}\, premium)$$

- The probability of going from minimum QoS to premium QoS within 20 to 40 time units without violating the minimum QoS constraint along the way is less than 0.99:

$$\mathcal{P}_{<0.99}(minimum\, \mathcal{U}^{[20,40]}\, premium)$$

Results

The results from tables 7.1 and 7.2 show that formula-dependent lumping plus model checking the lumped CTMC takes less time than model checking the original CTMC. Note that the number of blocks is larger in case of the second property.

	original CTMC			lumped CTMC		reduct. factor	
N	states	transitions	MC	blocks	lump+MC	states	time
2	276	1120	5.5	37	2.2	7.5	2.3
4	820	3616	9.4	65	4.0	12.6	2.4
8	2772	12832	36.0	239	16.3	11.6	2.2
16	10132	48160	360	917	70.0	11.0	5.1
32	38676	186400	1860	3599	300	10.7	6.2
64	151060	733216	7200	14267	1810	10.6	4.0
128	597012	2908192	29700	56819	9300	10.5	3.2
256	2373652	11583520	121000	226787	45700	10.5	2.6

Table 7.1: Results for $\mathcal{P}_{<0.99}(minimum\, \mathcal{U}^{\leq 40}\, premium)$

	original CTMC			lumped CTMC		reduct. factor	
N	states	transitions	MC	blocks	lump+MC	states	time
2	276	1120	7.0	70	3.6	3.9	1.9
4	820	3616	14.5	131	7.0	6.3	2.1
8	2772	12832	49.0	386	24.0	7.2	2.0
16	10132	48160	480	1300	96.0	7.8	5.0
32	38676	186400	2200	4742	430	8.2	5.1
64	151060	733216	8500	18082	2550	8.4	3.3
128	597012	2908192	33700	70586	12800	8.5	2.6
256	2373652	11583520	143000	278890	60900	8.5	2.3

Table 7.2: Results for $\mathcal{P}_{<0.99}(minimum\, \mathcal{U}^{[20,40]}\, premium)$

Compared to the results in section 6.4.1, formula-dependent lumping leads to greater state space reductions than formula-independent lumping. Also, formula-dependent lumping for the second property is faster than formula-independent lumping.

7.4.2 Cyclic Server Polling System

See section 5.4.3 for a description of this case study and the atomic propositions used in the CSL properties.

Properties

The following CSL properties have been checked:

- Within 10 time units station 1 is served before all other stations:

$$notServe_1 \, \mathcal{U}^{\leq 10} \, serve_1$$

- Within 5 to 10 time units station 1 is served before all other stations:

$$notServe_1 \, \mathcal{U}^{[5,10]} \, serve_1$$

Results

	original CTMC			lumped CTMC		reduct. factor	
N	states	transitions	MC	blocks	lump+MC	states	time
4	96	272	3.7	35	2.0	2.7	1.9
5	240	800	10.0	56	3.7	4.3	2.7
6	576	2208	27.5	84	6.1	6.9	4.5
7	1344	5824	65.0	120	9.9	11.2	6.6
8	3072	14848	160	165	15.0	18.6	10.7
9	6912	36864	480	220	27.0	31.4	17.8
10	15360	89600	1410	286	50.0	53.7	28.2
11	33792	214016	3170	364	101	92.8	31.4
12	73728	503808	7350	455	213	162.0	34.5
13	159744	1171456	17100	560	485	285.3	35.3
14	344064	2695168	40270	680	1100	506.0	36.6
15	737280	6144000	94600	816	2590	903.5	36.5

Table 7.3: Results for $notServe_1 \, \mathcal{U}^{\leq 10} \, serve_1$

Tables 7.3 and 7.4 show the results for both properties. Interesting to see is that formula-dependent lumping leads to a significant reduction of the state space, whereas formula-independent lumping does not lead to any reductions. The number of blocks is larger in case of the second property. Also, formula-dependent lumping plus model checking the lumped CTMC takes significantly less time than model checking the original CTMC. Note that both reduction factors increase as N increases.

	original CTMC			lumped CTMC		reduct. factor	
N	states	transitions	MC	blocks	lump+MC	states	time
4	96	272	3.9	45	3.4	2.1	1.1
5	240	800	11.0	76	5.0	3.2	2.2
6	576	2208	28.0	119	8.6	4.8	3.3
7	1344	5824	70.0	176	14.0	7.6	5.0
8	3072	14848	170	249	22.0	12.3	7.7
9	6912	36864	530	340	37.0	20.3	14.3
10	15360	89600	1540	451	66.0	34.1	23.3
11	33792	214016	3480	584	126	57.9	27.6
12	73728	503808	8130	741	251	99.5	32.4
13	159744	1171456	18800	924	528	172.9	35.6
14	344064	2695168	44500	1135	1170	303.1	38.0
15	737280	6144000	104000	1376	2720	535.8	38.2

Table 7.4: Results for $notServe_1\, \mathcal{U}^{[5,10]}\, serve_1$

7.4.3 Tandem Queueing Network

See section 6.4.3 for a description of this case study and the atomic propositions used in the CSL properties.

Property

The probability of leaving a situation where the second queue is fully occupied within 0.5 time units meets the bound $\trianglelefteq p$:

$$\mathcal{P}_{\trianglelefteq p}(full_2\, \mathcal{U}^{\leq 0.5}\, \neg full_2)$$

Results

	original CTMC			lumped CTMC		reduct. factor	
c	states	transitions	MC	blocks	lump+MC	states	time
15	496	1619	0.5	32	0.4	15.5	1.3
31	2016	6819	1.7	64	1.0	31.5	1.7
63	8128	27971	8.0	128	3.3	63.5	2.4
127	32640	113283	110	256	12.8	127.5	8.6
255	130816	455939	570	512	44.5	255.5	12.8
511	523776	1829379	2700	1024	179	511.5	15.1
1023	2096128	7328771	13500	2048	680	1023.5	19.9

Table 7.5: Results for $\mathcal{P}_{\trianglelefteq p}(full_2\, \mathcal{U}^{\leq 0.5}\, \neg full_2)$

Similar to the cyclic server polling system case study, formula-dependent lumping leads to a significant reduction of the state space, whereas formula-independent lumping does not. We observe from table 7.5 that both reduction factors increase as c increases. Also, formula-dependent lumping plus model checking the lumped CTMC is faster than than model checking the original CTMC.

7.5 Conclusion

We have studied the effectiveness of formula-dependent lumping for CSL time-bounded until formulas. Several case studies from chapter 6 concerning bisimulation minimisation (i. e. formula-independent lumping) have been used.

Our experiments showed that formula-dependent lumping can lead to greater state space reductions than formula-independent lumping. In some cases (cyclic server polling system and tandem queueing network), formula-independent lumping does not lead to any reductions, whereas formula-dependent lumping does reduce the state space significantly. Also, in our experiments, formula-dependent lumping plus model checking the lumped CTMC takes less time than model checking the original CTMC. We can conclude that formula-dependent lumping can be an even better state space reduction technique than formula-independent lumping.

Chapter 8

Conclusion

In this thesis, we studied the effectiveness of bisimulation minimisation for probabilistic model checking. The algorithm we used for computing the minimised model is the optimal lumping algorithm from [10]. We implemented this algorithm efficiently into the model checker MRMC. Using case studies from PRISM, we conducted several experiments. In these experiments, we verified a PCTL or CSL property on the original, non-minimised model as well as on the lumped model (i. e. the minimised model). For PCTL, we considered bounded and unbounded until properties and for CSL, we considered steady-state properties and time-bounded until properties.

Our experiments showed that bisimulation minimisation can significantly reduce DTMCs and CTMCs consisting of millions of states. For many DTMC models, the state space has been reduced by several orders of magnitude. However, for certain CTMC case studies, bisimulation minimisation did not lead to a reduction of the CTMC. Also, in several cases, the time to lump the model plus the time to verify the property on the lumped model is less than the time to verify the original model.

To lump the model, only atomic propositions contained in the property were considered. It is also possible to verify more than one property on the lumped model. In order to do so, the model has to be lumped by considering the atomic propositions contained in all properties to be verified. These properties can then be verified on a (possibly) much smaller model.

We also compared bisimulation minimisation to symmetry reduction. Using a CTMC case study, we showed that bisimulation minimisation can result into larger state space reductions. As expected, symmetry reduction is significantly faster than bisimulation minimisation since the lumping algorithm is more complex.

Formula-dependent lumping is a minimisation technique where the model is lumped for a specific PCTL or CSL property. Our experiments showed

that formula-dependent lumping can lead to larger state space reductions. In most cases, formula-dependent lumping plus verifying the property takes less time than verifying the original model.

For several cases, bisimulation minimisation results in faster runtimes. It should be noted that we only considered properties for which it is most likely that using bisimulation minimisation is faster than verifying the original model. For many properties, it is faster to verify the original model. However, we are not only interested in the runtime of model checking. The size of the state space is also of great importance. We showed that bisimulation minimisation can significantly reduce the state space. Therefore, we can conclude that bisimulation minimisation is a good state space reduction technique.

Future work

Bisimulation minimisation can be easily extended to reward model checking [4, 2]. The initial partition should be modified such that each state in an equivalence class also has equal state reward. Bisimulation minimisation may be even more effective for reward model checking because, in general, model checking reward models is more complex and time consuming.

We only considered DTMCs and CTMCs. The model checker PRISM also supports PCTL model checking on Markov Decision Processes (MDPs). MDPs extend DTMCs by allowing both probabilistic and non-deterministic behaviour. Bisimulation minimisation could also be an interesting technique for minimising MDPs [28].

Lumping requires to fully store the original model. Reducing the size of state space of the original model allows minimising larger models. MRMC uses a sparse matrix representation. It stores for each state a list of successor states and a list of predecessor states. The lumping algorithm only needs the list of predecessor states. Only storing the predecessor states can substantially reduce the size of the state space of the original model. Furthermore, we can speed up the computation of the cumulative probabilities or rates by associating the matrix values with the predecessors instead of the successors.

Bibliography

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, 1974.

[2] S. Andova, H. Hermanns, J.-P. Katoen. Discrete-time rewards model-checked. In K. G. Larsen, P. Niebert (editors) *Formal Modelling and Analysis of Timed Systems (FORMATS 2003)*, LNCS 2791, pp. 88–103, Springer, Berlin, 2003.

[3] A. Aziz, V. Singhal, F. Balarin. It usually works: The temporal logic of stochastic systems. In P. Wolper (editor) *Proceedings 7th International Conference on Computer Aided Verification (CAV'95)*, LNCS 939, pp. 155–165, Springer, Berlin, 1995.

[4] C. Baier, B. R. Haverkort, H. Hermanns, J.-P. Katoen. Automated performance and dependability evaluation using model checking. In M. Calzarossa, S. Tuccin (editors) *Performance Evaluation of Complex Systems: Techniques and Tools*, LNCS 2459, pp. 261–289, Springer, Berlin, 2002.

[5] C. Baier, B. R. Haverkort, H. Hermanns, J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, IEEE Computer Society, New York, 2003.

[6] C. Baier, H. Hermanns, J.-P. Katoen, V. Wolf. Comparative branching-time semantics for Markov chains. In R. M. Amadio, D. Lugiez (editors) *Proceedings 14th International Conference on Concurrency Theory*, LNCS 2761, pp. 492–508, Springer, Berlin, 2003.

[7] P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, 31:59–75, 1994.

[8] E. M. Clarke, E. A. Emerson, A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[9] E. M. Clarke, O. Grumberg, D. A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.

[10] S. Derisavi, H. Hermanns, W. H. Sanders. Optimal state-space lumping in Markov chains. *Information Processing Letters*, 87(6):309–315, 2003.

[11] K. Fisler, M. Y. Vardi. Bisimulation minimization in an automata-theoretic verification framework. In G. Gopalakrishnan, P. J. Windley (editors) *Proceedings Second International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, LNCS 1522, pp. 115–132, Springer, Berlin, 1998.

[12] B. Fox, P. Glynn. Computing Poisson probabilities. *Communications of the ACM*, 31(4):440–445, 1988.

[13] H. Hansson, B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[14] B. R. Haverkort. *Performance of Computer Communication Systems*. John Wiley & Sons, New York, 1998.

[15] B. R. Haverkort, H. Hermanns, J.-P. Katoen. On the use of model checking techniques for dependability evaluation. In *Proceedings 19th IEEE Symposium on Reliable Distributed Systems*, pp. 228–237, IEEE Computer Society, New York, 2000.

[16] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35(2):63–67, 1990.

[17] H. Hermanns, J. Meyer-Kayser, M. Siegle. Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains. In B. Plateau, W. Stewart, M. Silva (editors) *Proceedings 3rd International Workshop on the Numerical Solution of Markov Chains*, pp. 188–207, Prensas Universitarias de Zaragoza, 1999.

[18] O. Ibe, K. Trivedi. Stochastic Petri net models of polling systems. *IEEE Journal on Selected Areas in Communications*, 8(9):1649–1657, 1990.

[19] A. Itai, M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88:60–97, 1990.

[20] J.-P. Katoen, M. Khattri, I. S. Zapreev. A Markov reward model checker. In *Proceedings Second International Conference on the Quantitative Evaluation of Systems (QEST 2005)*, pp. 243–244, IEEE Computer Society, New York, 2005.

[21] J.-P. Katoen, M. Z. Kwiatkowska, G. Norman, D. Parker. Faster and symbolic CTMC model checking. In L. de Alfaro, S. Gilmore (editors) *Process Algebra and Probabilistic Methods*, LNCS 2165, pp. 23–38, Springer, Berlin, 2001.

[22] M. Kwiatkowska, G. Norman, D. Parker. PRISM: Probabilistic symbolic model checker. In T. Field, P. G. Harrison, J. T. Bradley, U. Harder (editors) *Proceedings 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, LNCS 2324, pp. 200–204, Springer, Berlin, 2002.

[23] M. Kwiatkowska, G. Norman, D. Parker. Symmetry reduction for probabilistic model checking. In T. Ball, R. Jones (editors) *Proceedings 18th International Conference on Computer Aided Verification (CAV'06)*, Springer-Verlag, To appear, 2006.

[24] R. Paige, R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal On Computing*, 16(6):973–989, 1987.

[25] A. Pnueli, L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.

[26] PRISM case studies. www.cs.bham.ac.uk/~dxp/prism/casestudies.

[27] M. Reiter, A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[28] R. Segala, N. A. Lynch. Probabilistic simulations for probabilistic processes. In B. Jonsson, J. Parrow (editors) *Proceedings of the 5th International Conference on Concurrency Theory*, LNCS 836, pp. 481–496, Springer, Berlin, 1994.

[29] D. D. Sleator, R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32:652–686, 1985.

[30] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.