



Lehrstuhl für Informatik 2
Software Modeling and Verification



Diploma Thesis

Symbolic Bisimulation Minimization of Markov Models

Christian Dehnert

October 20, 2011

First Referee:

Prof. Dr. Ir. Joost-Pieter Katoen

Second Referee:

Prof. Dr. Gerhard Lakemeyer

Additionally supervised by:

Dr. David Parker (University of Oxford)

Acknowledgements.

First of all, I want to thank Joost-Pieter Katoen for his advice and support during this diploma thesis. Also, I want to thank him, Marta Kwiatkowska and Dave Parker for giving me the opportunity to spend two months at the University of Oxford. There, fruitful discussions with Dave helped inspire me, for which I am very grateful.

Of course, I extend my thanks to all the people I got to know in Oxford for making it an amazing and unforgettable time.

Furthermore, I give my thanks to Alberto Griggio of the MathSat development team for providing fast and reliable support.

I want to thank Benedikt Brütsch for using his expertise regarding SIGREF to familiarize me with its internals.

As it must have been difficult for my friends to listen to my thesis-related talk all the time, I want to thank them for their patience and willingness to think about my problems despite their sometimes uninitiated minds.

Last but not least, I want to express my gratitude towards my family for always supporting my decisions and in particular my studies.

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den 20. Oktober 2011

Christian Dehnert

Zusammenfassung

Probabilistisches Model Checking ist eine Technik, um Systeme, die probabilistische Verhaltensweisen aufzeigen, bezüglich einer Spezifikation zu verifizieren. Um Systeme mit besonders vielen Zuständen behandeln zu können, benutzt man beispielsweise symbolische Datenstrukturen, wie etwa binäre Entscheidungsdiagramme (BED), oder diverse Minimierungstechniken. Eine dieser Techniken ist die Bisimulationsminimierung, die Zustände zusammenfasst, die sich äquivalent verhalten. Wir zeigen zwei Verfahren auf, die den Bisimulationsquotienten mit Hilfe von symbolischen Datenstrukturen berechnen. Während die erste auf BED-Manipulationen basiert, formuliert die zweite die Berechnung als eine Serie von Anfragen an einen SAT-solver für die lineare Integer-Arithmetik. Für beide Methoden untersuchten wir mit Fallstudien die Effektivität bezüglich des Zeitaufwands als auch des Speicherverbrauchs. Wir konnten zeigen, dass große Zustandsraumverkleinerungen erreicht werden, die aber gewöhnlich nicht zu einer verbesserten Gesamtlaufzeit führen. Darüber hinaus konnten wir mittels der SAT-basierten Methode Systeme verifizieren, deren BED-Repräsentation zu groß für eine direkte Analyse waren.

Abstract

Probabilistic model checking is a technique to verify systems that exhibit probabilistic behavior with respect to a specification. Symbolic data structures, such as Binary Decision Diagrams (BDD), and various minimization techniques are used to enable treatment of systems with a large state space. One of these methods is bisimulation minimization, which lumps states that behave equivalently. We present two methods that compute the bisimulation quotient using symbolic data structures. While the first one is based on BDD manipulations, the second formulates the computation as a series of queries to a SAT solver for the linear integer arithmetic. For both methods we conducted case studies to evaluate the effectiveness with respect to time and memory requirements. We were able to achieve significant state space reductions, which usually not result in a decreasing total runtime, however. Furthermore, using the SAT-based method, we were able to verify systems whose BDD representation sizes rendered direct analyses infeasible.

Contents

1. Introduction	1
1.1. Verification	1
1.2. State Space Explosion Problem	3
1.3. Outline	4
2. Preliminaries	5
2.1. Markov Models	5
2.1.1. Markov Chains	6
2.1.2. Markov Decision Processes	8
2.1.3. Reward Extensions	12
2.2. Probabilistic Temporal Logics	12
2.3. (Strong) Probabilistic Bisimulation	15
2.4. PRISM	21
2.4.1. The PRISM modeling language	22
2.5. Preliminaries of the BDD-based approach	27
2.5.1. General	27
2.5.2. Formalism	28
2.5.3. State Space Representation Using MTBDDs	29
2.5.4. Sigref: A Symbolic Bisimulation Tool	29
3. BDD-based bisimulation minimization	31
3.1. Creating the SIGREF Input	32
3.1.1. The System-MTBDD	32
3.1.2. The Initial Partition	33
3.2. Computing the Quotient	34
3.2.1. General	34
3.2.2. Symbolic Implementation	35
3.3. Case Studies	40
3.3.1. Shared Coin Protocol (MDP)	40
3.3.2. Firewire (MDP)	42
3.3.3. Peer-To-Peer Protocol (CTMC)	42
3.3.4. Wireless Network (MDP)	42
3.3.5. Synchronous Leader Election Protocol (DTMC)	45
3.3.6. Zeroconf Protocol (MDP)	45
3.3.7. Crowds Protocol (DTMC)	49
3.4. Analysis	49

3.4.1. Bisimulation in General	49
3.4.2. BNI vs. BFI	51
3.4.3. Bisimulation and Symmetry Reduction	51
3.4.4. Symbolic vs. Explicit	52
3.5. Conclusion	52
4. SMT-based bisimulation minimization	55
4.1. Meaning of Weakest Preconditions	56
4.2. Idea	57
4.3. Algorithm	59
4.3.1. Computing the Bisimulation	59
4.3.2. Extracting the Quotient	62
4.4. Prototypical Implementation	65
4.4.1. General	65
4.4.2. Optimizations	65
4.5. Case Studies	67
4.5.1. Synchronous Leader Election Protocol (DTMC)	68
4.5.2. Crowds Protocol (DTMC)	68
4.6. Analysis	71
4.6.1. Performance	71
4.6.2. Problems and Improvements	71
4.7. Conclusion	72
5. Conclusion	73
5.1. Summary	73
5.2. Future Work	74
Appendix	77
A. Case Studies for BDD-based Bisimulation	77
A.1. General	77
A.2. Properties	77
A.3. Tables	80
B. Case Studies for SMT-based Bisimulation	89
B.1. General	89
B.2. Properties	89
B.3. Tables	90
C. Reformulation of the Crowds Protocol	91
Bibliography	93

List of Figures

1.1.	A schematic of the model checking approach [1].	2
2.1.	An example DTMC.	7
2.2.	Two processes and their alleged joint-behavior DTMC.	9
2.3.	The joint-behaviour MDP of the two processes.	11
2.4.	Model checking using bisimulation minimization.	20
2.5.	Schematic of bisimulation computation via partition refinement.	21
2.6.	The probabilistic program P_{Ex}	25
2.7.	The reachable state space of $\llbracket P_{Ex} \rrbracket$	26
3.1.	Integration of SIGREF into the PRISM process chain.	32
3.2.	The form of the partition BDD \mathcal{P}^Π for $\Pi = \{B_1, \dots, B_j\}$	35
3.3.	The form of the signature BDDs.	36
3.4.	The set \mathcal{K}_v for $v = r _{\mathbb{S}=s}$	38
3.5.	Shared Coin Protocol results for BDD-based bisimulation.	41
3.6.	Firewire results for BDD-based bisimulation.	43
3.7.	Peer-To-Peer Protocol results for BDD-based bisimulation.	44
3.8.	WLAN results for BDD-based bisimulation.	46
3.9.	Synchronous Leader Election Protocol results for BDD-based bisimulation.	47
3.10.	Zeroconf Protocol results for BDD-based bisimulation.	48
3.11.	Crowds Protocol results for BDD-based bisimulation.	50
4.1.	The probabilistic program $P_{Ex, \sim}$	64
4.2.	The quotient DTMC $\mathcal{D}_{Ex, \sim} = \llbracket P_{Ex, \sim} \rrbracket$	64
4.3.	The partition tree T_π	66
4.4.	Synchronous Leader Election Protocol results for SMT-based bisimulation.	69
4.5.	Crowds Protocol results for SMT-based bisimulation.	70

1. Introduction

Begin at the beginning and go on till you come to the end: then stop.

Lewis Carroll - *Alice's Adventures in Wonderland*

This chapter introduces (probabilistic) model checking as a mathematically rigorous technique to verify systems against a specification. We point out one of the central problems involved and give an overview over already proposed countermeasures before we end this chapter with an outline of this thesis.

1.1. Verification

In times of ever more complex system designs, it becomes increasingly difficult and often even impossible to manually analyze systems. Yet, for certain systems, e.g. safety-critical ones or ones for which the repair of previously undetected errors may be very cost-intensive, it is crucial to determine whether they behave correctly with respect to the requirements that they are intended to fulfill. Thus, the need for comprehensive, automated analysis techniques arises. An obvious choice for this is testing: in order to increase confidence in the design, prototypes may be built and tested extensively. This, however, only reveals bugs, but cannot prove their absence, as the test cases usually do not cover every possible execution of the system.

Aiming to counter this shortcoming, formal verification was proposed as a mathematically rigorous technique to prove or disprove the correctness of a given system with respect to a given specification. The two main approaches are model checking [1] and logical inference [2, 3]. While logical inference uses theorem provers to reason about the system, model checking amounts to an exhaustive search through all possible executions of the system, the so-called *state space*, and proves (or disproves) that none of these violate a requirement of the given specification. In this thesis, we focus on model checking as the technique of choice.

Figure 1.1 provides an overview of the general approach. Given an informal description of a system to be verified and a set of informal requirements, both must be formalized first to enable rigorous treatment. For requirements this is usually a formulation in a temporal logic, e.g. CTL or LTL [1]. These allow for the formal specification of (qualitative) properties such as "the system will never reach a bad state" or "at any

point in time, the system is able to return to its initial state”. The system model usually is a labeled transition system, i.e. comprises the states of the system and the possible transitions between these states. Then, both formal models are passed to a software tool, the *model checker*, which checks the system model against the properties. If the system fulfills all of them, the model checker reports *satisfied* to confirm the system’s compliance with the provided requirements. If, on the other hand, a property is violated, a counterexample, i.e. a violating execution of the system, can be returned to provide the designer with insightful feedback about how to improve the design of the system. Furthermore, after the formalization stage, model checking is a push-button technique as it is fully automated. Moreover, it can support the whole development cycle of the system through the various design stages by refining the accuracy of the model accordingly. Consequently, there is an increasing interest and support for model checking by industry, e.g. through various research projects such as the COMPASS¹ project funded by the European Space Agency (ESA).

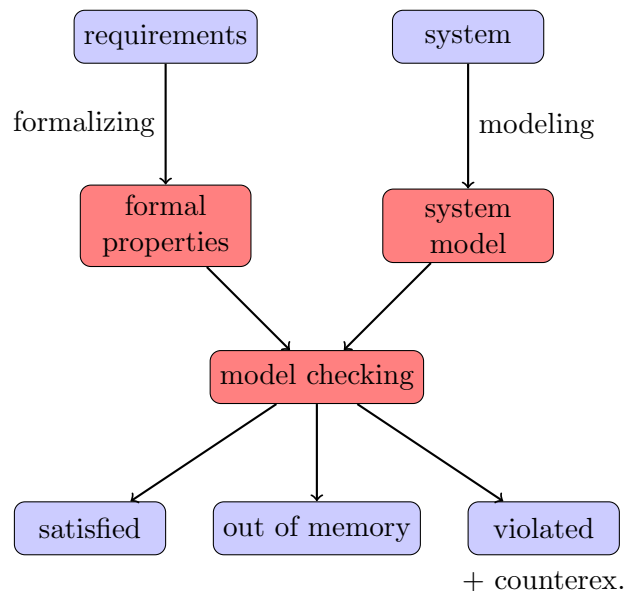


Figure 1.1.: A schematic of the model checking approach [1].

Whilst ”normal” qualitative model checking has proven to be remarkably successful in some settings, others are inherently probabilistic and are consequently only amenable to qualitative model checking in a very restricted way. Consider, for example, a hardware circuit that uses components that fail with a given probability. Now, trying to prove that the complete circuit will never produce wrong results will inevitably fail, as there

¹Correctness, Modeling And Performance of Aerospace Systems.
<http://compass.informatik.rwth-aachen.de>

is a non-zero probability for this scenario, but clearly it is not the case that the system will fail in any case. So the only information derivable by conventional model checking is that the system *might* fail, but does not do so in any case, giving no meaningful insight. To overcome this deficit, probabilistic model checking extends conventional model checking with quantitative aspects, such as probabilities and - to a certain degree - time. Using these techniques and the corresponding models, it is possible to verify whether the aforementioned system will stay intact with a probability greater than, say, 95%. So unlike before, the designer can either decide that the proven probability is high enough or take appropriate measures to increase the probability, e.g. introduce redundancy to some components in order to reduce failure rates. In this thesis, the focus lies entirely on models exhibiting quantitative behavior.

1.2. State Space Explosion Problem

Unfortunately, despite the many advantages of model checking, it suffers from problems that often prevent real-world models from being successfully verified. Most notably this is due to the state space explosion problem, meaning that the state space of the system becomes too large to be stored in memory (indicated by the corresponding "outcome" of the model checking process in Figure 1.1). State-of-the-art model checkers can process state spaces up to 10^8 to 10^9 states using an explicit representation of the states in memory and 10^{20} up to even 10^{476} (for very specific systems) using so-called *symbolic* data structures that exploit similarity in the state space [1].

In particular, in the context of probabilistic model checking, where computationally expensive numerical operations dominate the runtime, it may also well be that state spaces that can be stored in memory are still too large to be analyzed, because the analysis simply takes too long to be feasible. In order to stem this problem many techniques have been proposed, such as bisimulation minimization [4], probabilistic counterexample-guided abstraction-refinement [5], game-based abstraction-refinement [6] and assume-guarantee verification [7].

(Probabilistic) bisimulation minimization exploits redundancies in the state space by lumping states that behave "in the same way" according to some defined measure. This is done in a way that simultaneously preserves the temporal properties of interest of the original system. Hence, the original system can be minimized and the sometimes significantly smaller quotient system can be used to verify the original system. Katoen et al. present a survey [4] of the effectiveness of bisimulation minimization in a probabilistic setting using an explicit state space representation and conclude that in many cases the application of this technique is worthwhile. This thesis aims to analyze the effectiveness of bisimulation minimization in a symbolic setting as well as to develop a new approach for computing the bisimulation quotient symbolically without constructing the state space beforehand.

1.3. Outline

The thesis is divided as follows. In chapter 2 we establish the theoretical background and setting, before we present and evaluate a bisimulation minimization procedure based on the manipulation of Binary Decision Diagrams (BDD) in chapter 3. In chapter 4 we develop a novel approach that centers on solving instances of a variant of the satisfiability problem (SAT) as a means to extract the quotient system without building the full system first and evaluate its effectiveness. Finally, chapter 5 summarizes the results of the two different approaches and gives an overview about possible future work.

2. Preliminaries

So be sure when you step, step with care and great tact. And remember that life's a great balancing act. And will you succeed? Yes! You will, indeed! (98 and $\frac{3}{4}$ percent guaranteed) Kid, you'll move mountains.

Dr. Seuss - *Oh, the Places You'll Go!*

This chapter introduces Markov models, the logics used to specify properties and the connection thereof to bisimulation minimization. We describe the modeling language of PRISM, a probabilistic model checker, and establish the specifics of the BDD-based bisimulation minimization approach.

2.1. Markov Models

Markov models are similar to transition systems in that they comprise states and transitions between these states, but unlike them transitions are equipped with randomness information. Put differently, in each state, the successor state is determined by a stochastic choice. In particular, it is important that the stochastic choice in each state only depends on the state itself and not on the path that was taken to reach that state, a property that is referred to as the *Markov property*, which gives rise to the term *Markov models*.

Markov models can be classified according to their ability to model non-determinism and time. Time may be measured in discrete time steps or continuously, depending on the accuracy of timed behavior that the model is required to reflect. Non-determinism is an important means to model the possible concurrency of certain actions in absence of any information about which is scheduled first. Table 2.1 gives an overview of the available formalisms: while discrete-time Markov Chains (DTMCs) and their continuous-time counterpart continuous-time Markov Chains (CTMCs) do not allow for the modeling of non-determinism, Markov Decision Processes (MDPs) and continuous-time Markov Decision Processes (CTMDPs) do. As CTMDPs are only included for completeness reasons and will not be considered further in this thesis, only the other three models will be formally defined.

	deterministic	non-deterministic
discrete-time	DTMC	MDP
continuous-time	CTMC	CTMDP

Table 2.1.: An overview over available markov models.

2.1.1. Markov Chains

Let 2^A denote the power set of A and $Dist_S$ the set of discrete probability distributions over a set S , i.e. $Dist_S = \{\mu : S \rightarrow [0, 1] \mid \sum_{s \in S} \mu(s) = 1\}$.

Definition 2.1 (Discrete-Time Markov Chain (DTMC)). A discrete-time Markov Chain is a tuple

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

where

- S is a countable, non-empty set of states,
- $\mathbf{P} : S \times S \rightarrow [0, 1]$ is the transition probability function that assigns to each pair $(s, s') \in S \times S$ of states the probability $\mathbf{P}(s, s')$ of moving from state s to s' in one step such that $\mathbf{P}(s, \cdot) \in Dist_S$,
- $s_{init} \in S$ is the initial state,
- AP is a set of atomic propositions,
- $L : S \rightarrow 2^{AP}$ is the labeling function that assigns the possibly empty set of atomic propositions $L(s)$ to a state $s \in S$.

□

Let $\mathcal{D} = (S_{\mathcal{D}}, \mathbf{P}_{\mathcal{D}}, s_{init}^{\mathcal{D}}, AP_{\mathcal{D}}, L_{\mathcal{D}})$ be a DTMC. In each state $s \in S_{\mathcal{D}}$, the successor state is chosen according to a single discrete probability distribution given by $\mathbf{P}_{\mathcal{D}}(s, \cdot)$. Given $T \subseteq S_{\mathcal{D}}$, we denote by $\mathbf{P}_{\mathcal{D}}(s, T)$ the probability to enter any state in T in one step from $s \in S_{\mathcal{D}}$, i.e. $\mathbf{P}_{\mathcal{D}}(s, T) = \sum_{t \in T} \mathbf{P}_{\mathcal{D}}(s, t)$.

A path in \mathcal{D} is an infinite sequence of states $\pi = s_0 s_1 s_2 \dots \in S^{\omega}$ such that $s_i \in S_{\mathcal{D}}$ and $\mathbf{P}_{\mathcal{D}}(s_i, s_{i+1}) > 0$ for all $i \in \mathbb{N}$. With $\pi[i]$ we refer to the $(i + 1)$ -th state of π , i.e. $\pi[i] = s_i$, and with $Paths_{\mathcal{D}}$ and $Paths_{\mathcal{D}}(s)$ we denote the set of all paths in \mathcal{D} and the set of all paths that start in $s \in S_{\mathcal{D}}$, respectively. More formally, $Paths_{\mathcal{D}}(s) = \{\pi \in Paths_{\mathcal{D}} \mid \pi = s_0 s_1 s_2 \dots \text{ and } s_0 = s\}$. It can be shown that \mathcal{D} induces a unique probability measure $Pr_{\mathcal{D}}$ on $Paths_{\mathcal{D}}$ that associates with each measurable subset $T \subseteq Paths_{\mathcal{D}}$ the probability $Pr_{\mathcal{D}}(T)$ [1].

In the literature, the definition of Markov models sometimes differs from the above in that they allow for an initial distribution $\iota_{init} : S \rightarrow [0, 1]$ instead of just one initial state. However, this is no generalization as this case can be easily transformed into an equivalent DTMC with only one initial state. Let $\mathcal{D} = (S_{\mathcal{D}}, \mathbf{P}_{\mathcal{D}}, \iota_{init}^{\mathcal{D}}, AP_{\mathcal{D}}, L_{\mathcal{D}})$ be a DTMC with an initial distribution $\iota_{init}^{\mathcal{D}} \in Dist_S$, then $\bar{\mathcal{D}} = (S_{\mathcal{D}} \cup \{s_{init}^{\mathcal{D}}\}, \bar{\mathbf{P}}_{\mathcal{D}}, s_{init}^{\mathcal{D}}, AP_{\mathcal{D}}, L_{\mathcal{D}})$ for a state $s_{init}^{\mathcal{D}} \notin S_{\mathcal{D}}$ with

$$\bar{\mathbf{P}}_{\mathcal{D}}(s, s') = \begin{cases} \iota_{init}^{\mathcal{D}}(s') & \text{if } s = s_{init}^{\mathcal{D}} \\ \mathbf{P}_{\mathcal{D}}(s, s') & \text{if } s, s' \neq s_{init}^{\mathcal{D}} \\ 0 & \text{otherwise} \end{cases}$$

is an equivalent Markov chain with only one initial state. Thus, only considering models with a single initial state throughout this thesis is no restriction.

Example 2.2.

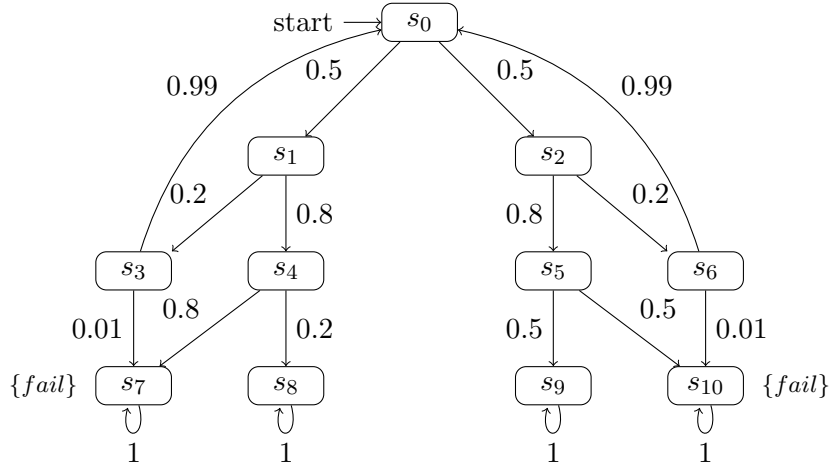


Figure 2.1.: An example DTMC.

Figure 2.1 shows a DTMC $\mathcal{D} = (S_{\mathcal{D}}, \mathbf{P}_{\mathcal{D}}, s_{init}^{\mathcal{D}}, AP_{\mathcal{D}}, L_{\mathcal{D}})$ with

- $S_{\mathcal{D}} = \{s_i \mid 1 \leq i \leq 10\}$,
- $s_{init}^{\mathcal{D}} = s_0$,
- $AP_{\mathcal{D}} = \{fail\}$,
- $L_{\mathcal{D}}(s_i) = \{fail\}$ iff $i \in \{7, 10\}$, and, amongst others,
- $\mathbf{P}_{\mathcal{D}}(s_0, s_1) = 0.5$, $\mathbf{P}_{\mathcal{D}}(s_2, s_6) = 0.2$, and $\mathbf{P}_{\mathcal{D}}(s_7, s_7) = 1$.

Additionally, $\pi_1 = s_0 s_1 s_4 (s_7)^\omega \in \text{Paths}_{\mathcal{D}}$ is a path in \mathcal{D} with $\pi[2] = s_4$. The probability of the set of paths that start with $s_0 s_1 s_4 s_7$

$$T = \{\pi \in \text{Paths}_{\mathcal{D}} \mid \pi[0] = s_0, \pi[1] = s_1, \pi[2] = s_4, \pi[3] = s_7\} = \{\pi_1\}$$

is given by $\text{Pr}_{\mathcal{D}}(T) = 0.5 \cdot 0.8 \cdot 0.8 = 0.32$. \square

CTMCs now extend this concept by substituting rates $\mathbf{R}(s, s')$ for the probabilities $\mathbf{P}(s, s')$ with the meaning, that in a state s the transition to s' is taken *within t time units* with a probability of $1 - e^{-\mathbf{R}(s, s') \cdot t}$. In other words, there is a probability to stay in a state that decreases with progressing time, before one of the outgoing transitions to another state is triggered.

Definition 2.3 (Continuous-Time Markov Chain (CTMC)). A continuous-time Markov Chain is a tuple

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

where

- S, s_{init}, AP and L are the same as for DTMCs (see Definition 2.1), and
- $\mathbf{R} : S \times S \rightarrow \mathbb{R}_{\geq 0}$ is the transition rate function that assigns the rate $\mathbf{R}(s, s')$ of moving from state s to s' to each pair $(s, s') \in S \times S$ of states.

\square

Let $\mathcal{C} = (S_{\mathcal{C}}, \mathbf{R}_{\mathcal{C}}, s_{init}^{\mathcal{C}}, AP_{\mathcal{C}}, L_{\mathcal{C}})$ be a CTMC. As for DTMCs, $\mathbf{R}_{\mathcal{C}}(s, T)$ lifts the transition rate function to sets $T \subseteq S_{\mathcal{C}}$ of states. To account for the sojourn times in the states, a path now has to include these. Hence, a (timed) path in \mathcal{C} is a sequence of states with intermediate sojourn times $\pi = s_0 t_1 s_1 t_2 s_2 \dots \in (S_{\mathcal{C}} \times \mathbb{R}_{\geq 0})^\omega$ such that $s_i \in S_{\mathcal{C}}$ and $\mathbf{R}_{\mathcal{C}}(s_i, s_{i+1}) > 0$ for all $i \in \mathbb{N}$. Analogously to DTMCs, $\pi[i]$ refers to the $(i + 1)$ -th state in π , $\text{Paths}_{\mathcal{C}}$ denotes the set of all timed paths in \mathcal{C} , $\text{Paths}_{\mathcal{C}}(s)$ the paths that start in $s \in S_{\mathcal{C}}$ and $\text{Pr}_{\mathcal{C}}$ the uniquely induced probability measure over timed paths. Additionally, for a non-time-divergent path $\pi = s_0 t_0 s_1 t_1 s_2 \dots \in \text{Paths}_{\mathcal{C}}$ and $t \in \mathbb{R}_{\geq 0}$, we denote with $\pi@t$ the state that the path is in after t time units, i.e. $\pi@t = s[i]$ where i is such that $\sum_{0 \leq j < i} t_j \leq t$ and $\sum_{0 \leq j \leq i} t_j > t$.

2.1.2. Markov Decision Processes

Markov chains do not support the modeling of non-determinism. In particular, note that having multiple possible successor states is not to be confused with non-determinism. While states in Markov chains may have several successors, all choices are required to be equipped with randomness information, thus explicitly specifying the likelihood of the respective choice. This is in contrast to non-determinism, which models choice in total absence of any information about which transition is scheduled. For example, consider the two probabilistic processes depicted at the top of Figure 2.2(a), which are to be executed concurrently on a single processing unit. It is tempting to assume

that it is equally likely in each step that the next action of one or the other process is scheduled, resulting in the Markov chain depicted in Figure 2.2(b). But, clearly, making this assumption is not entirely accurate provided that there is no information about the scheduling order of the processing unit. There may be more adverse schedulers that dramatically lower the probability of one of the processes finishing in time, e.g. by scheduling all steps of the other process first.

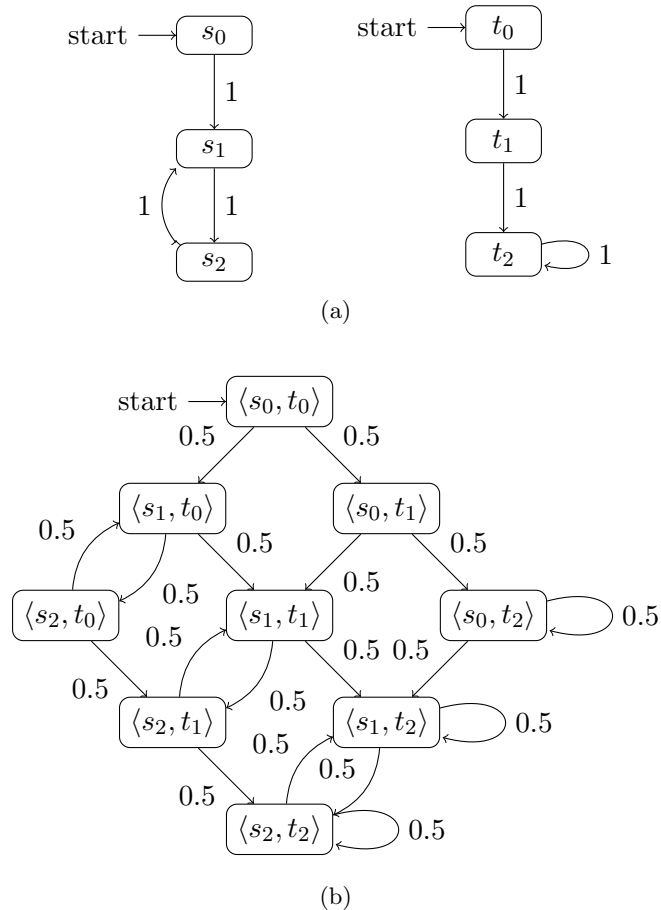


Figure 2.2.: Two processes and their alleged joint-behavior DTMC.

To account for non-determinism in probabilistic systems, Markov Decision Processes (MDPs) have been proposed. Intuitively, an MDP is an extension of a DTMC in the following sense: in each state of an MDP, there is a nondeterministic choice among so-called *actions*. Each of these actions is associated with a discrete probability distribution over states. Consequently, selecting an action a and thereby omitting the other actions resolves the non-determinism in a state, selects a unique discrete probability distribution that defines the possible successor states and, thus, effectively reduces this state to a state of a DTMC.

Definition 2.4 (Markov Decision Process (MDP)). A (discrete-time) Markov Decision Process is a tuple

$$\mathcal{M} = (S, Act, \mathbf{P}, s_{init}, AP, L)$$

where

- S , s_{init} , AP and L are the same as for DTMCs (see Definition 2.1),
- Act is a set of actions, and
- $\mathbf{P} : S \times Act \times S \rightarrow [0, 1]$ is the transition probability function that assigns to each tuple $(s, a, s') \in S \times Act \times S$ the probability $\mathbf{P}(s, a, s')$ of moving from state s to s' with action a such that for all $s \in S$ and $a \in Act : \sum_{s' \in S} \mathbf{P}(s, a, s') \in \{0, 1\}$.

□

Let $\mathcal{M} = (S_{\mathcal{M}}, Act_{\mathcal{M}}, \mathbf{P}_{\mathcal{M}}, s_{init}^{\mathcal{M}}, AP_{\mathcal{M}}, L_{\mathcal{M}})$ be an MDP. An action $a \in Act_{\mathcal{M}}$ is called *enabled* in a state $s \in S_{\mathcal{M}}$ if $\sum_{s' \in S_{\mathcal{M}}} \mathbf{P}_{\mathcal{M}}(s, a, s') = 1$. As usual, $\mathbf{P}(s, a, T)$ lifts the transition probability function to sets of states $T \subseteq S_{\mathcal{M}}$ in the canonical way. Let $Act_{\mathcal{M}}(s)$ denote the set of all enabled actions in s . Note that \mathcal{M} can be considered a DTMC if in each state there is exactly one action enabled, i.e. $|Act_{\mathcal{M}}(s)| = 1$ for all $s \in S_{\mathcal{M}}$, whereas every DTMC can be understood as an MDP without non-deterministic choices. A path in \mathcal{M} is an alternating sequence of states and actions $\pi = s_0 a_0 s_1 a_1 s_2 \dots \in (S_{\mathcal{M}} \times Act)^{\omega}$ such that $s_i \in S_{\mathcal{M}}$, $a_i \in Act(s_i)$ and $\mathbf{P}(s_i, a_i, s_{i+1}) > 0$ for all $i \in \mathbb{N}$ and, as before, $\pi[i]$ identifies s_i , $Paths_{\mathcal{M}}$ denotes the set of paths in \mathcal{M} and $Paths_{\mathcal{M}}(s)$ those paths that start in $s \in S_{\mathcal{M}}$. Additionally, for $s \in S$ let $Dist_{\mathcal{M}}(s) = \{\mu \in Dist_S \mid \text{there exists } a \in Act(s) \text{ such that } \mathbf{P}_{\mathcal{M}}(s, a, \cdot) = \mu\}$ denote the set of probability distributions of actions that are enabled in s . Figure 2.3 depicts the MDP that accurately models the concurrent execution of the processes of Figure 2.2. Appearing similar to the DTMC, the MDP makes no assumption about the likelihood that either of the processes is scheduled next, but provides a non-deterministic choice in each state instead.

To be able to measure probabilities of paths in MDPs, it is necessary to resolve all non-deterministic choices, which is done by *schedulers*.

Definition 2.5 (Scheduler for an MDP). Let $\mathcal{M} = (S, Act, \mathbf{P}, s_{init}, AP, L)$ be an MDP. A scheduler $\sigma : S^+ \rightarrow Act$ for \mathcal{M} is a function that maps each finite sequence of states $\omega = s_0 s_1 s_2 \dots s_n \in S^+$ to an action that is enabled in the last state of that sequence, i.e. $\sigma(\omega) \in Act_{\mathcal{M}}(s_n)$.

A scheduler σ is called *memoryless* if the choice of the next action only depends on the last state of the given state sequence, i.e. if σ can be represented as a function $\sigma : S \rightarrow Act$. □

The reason as to why it suffices that a scheduler σ maps state sequences instead of path fragments to actions is that the action names can be reconstructed using σ itself: for $\omega = s_0 s_1 s_2 \dots s_n \in S^+$ we retrieve the corresponding path fragment as $\pi_{\omega}^{\sigma} = s_0 \sigma(s_0) s_1 \sigma(s_0 s_1) \dots \sigma(s_0 \dots s_{n-1}) s_n$.

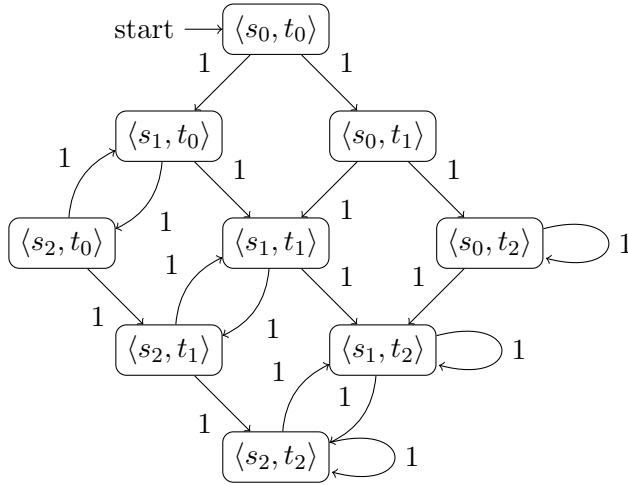


Figure 2.3.: The joint-behaviour MDP of the two processes.

Because a scheduler for an MDP resolves all non-deterministic choices by selecting exactly one available action, an MDP \mathcal{M} and a scheduler σ for \mathcal{M} induce a DTMC \mathcal{M}_σ . Intuitively, this DTMC reflects the behavior of \mathcal{M} when all non-deterministic choices are governed by σ .

Definition 2.6 (Induced DTMC). Let $\mathcal{M} = (S, Act, \mathbf{P}, s_{init}, AP, L)$ be an MDP and σ a scheduler for \mathcal{M} . Then \mathcal{M} and σ induce the DTMC

$$\mathcal{M}^\sigma = (S^+, \mathbf{P}_\sigma, s_{init}, AP, L')$$

such that for all $\omega = s_0 s_1 s_2 \dots s_n \in S^+$

- $\mathbf{P}_\sigma(\omega, \omega s_{n+1}) = \mathbf{P}(s_n, \sigma(\omega), s_{n+1})$, and
- $L'(\omega) = L(s_n)$.

□

Note that \mathcal{M}^σ in turn induces the probability measure $Pr_{\mathcal{M}^\sigma}$ on $Paths_{\mathcal{M}^\sigma}$, which we will from now on also denote with $Pr_{\mathcal{M}}^\sigma$.

In the sequel, for a Markov model \mathcal{M} with state space S , atomic propositions AP and a state $s \in S$, we will denote by \mathcal{M}_s the Markov model that results from \mathcal{M} by making s the (only) initial state. The corresponding probability measures are $Pr_{\mathcal{M}}^s$ if \mathcal{M} is a DTMC or CTMC, and $Pr_{\mathcal{M}}^{s, \sigma}$ if \mathcal{M} is an MDP and σ is a scheduler for \mathcal{M} . DTMCs and CTMCs are considered to be *finite*, if S and AP are finite. For MDPs, we additionally require Act to be finite. Often it is useful to identify the probability and rate functions of Markov models with matrices. For DTMCs and CTMCs these are the $S \times S$ matrices

$(\mathbf{P}(s, s'))_{s, s' \in S}$ and $(\mathbf{R}(s, s'))_{s, s' \in S}$, respectively. For a finite MDP \mathcal{M} with the set of actions $Act = \{a_1, \dots, a_n\}$, we identify $\mathbf{P}_{\mathcal{M}}$ with the $n \cdot S \times S$ matrix

$$\begin{pmatrix} \mathbf{P}_{a_1}(s, s') \\ \vdots \\ \mathbf{P}_{a_n}(s, s') \end{pmatrix}_{s, s' \in S}$$

where $\mathbf{P}_{a_1}(s, s') = \mathbf{P}_{\mathcal{M}}(s, a_1, s')$ for $s, s' \in S$.

Analogously to the relationship between DTMCs and CTMCs, there is a continuous-time counterpart of MDPs, not surprisingly called continuous-time Markov Decision Processes (CTMDPs). As they will not be considered further, they are not formally defined here.

2.1.3. Reward Extensions

Markov models can be equipped with an additional quantitative measure. This so-called *reward function* assigns a reward, a non-negative real number, to each state of the Markov model. This induces a quantitative measure, which will allow us to reason over the accumulated rewards along paths. The combination of a Markov model with a reward function, a Markov reward model (MRM), is formally defined as follows.

Definition 2.7 (Markov Reward Model). A Markov reward model (DMRM) is a tuple $\mathcal{M}_r = (\mathcal{M}, r)$ where

- \mathcal{M} is a Markov model with state space S ,
- $r : S \rightarrow \mathbb{R}_{\geq 0}$ is a function that assigns non-negative rewards to states.

□

If for an MRM $\mathcal{M}_r = (\mathcal{M}, r)$ the Markov model \mathcal{M} is a DTMC or an MDP, we will speak of a discrete-time Markov reward model (DMRM). Likewise, if \mathcal{M} is a CTMC, \mathcal{M}_r is a continuous-time Markov reward model (CMRM).

In a DMRM, the reward $r(s)$ is "gained" whenever a state s is left, i.e. a transition from s is taken. This contrasts CMRMs, where the reward for the visit of a state s is $r(s) \cdot t$ where t is the time spent in that state. These definitions are easily lifted to path fragments, which enables specific temporal logics to reason over the rewards gained during system executions.

2.2. Probabilistic Temporal Logics

To allow for the mathematically rigorous treatment that is verification, it is indispensable to not only formalize the model to be verified, but also the requirements. For qualitative model checking, this is usually done by formulating the requirement in a temporal logic, such as, e.g., CTL or LTL, which permit the formulation of qualitative

properties in mathematically precise manner. Typical properties include "there does not exist an execution that leads to a bad state" or "whenever the system has reached an unsafe state, the system will eventually perform a security-shutdown".

For Markov models, however, these logics are only partially applicable as they do not take the quantitative aspects into consideration. Although there is a probabilistic interpretation of LTL, we will now focus on branching time logics for Markov models that are more closely related to CTL, namely Probabilistic Computation Tree Logic (PCTL) for DTMCs and MDPs and Continuous Stochastic Logic (CSL) for CTMCs. Just like for CTL, their syntax is defined in two parts, that refer to each other: there are state formulae, whose truth values can be evaluated for states only, and path formulae, which are interpreted over paths. Because we take a state-based view of Markov models, we require the overall formula Φ to be a state formula, such that the task of model checking a Markov model \mathcal{M} with initial state s_{init} against Φ amounts to determine whether the initial state satisfies Φ . We will now present the formal definitions for the syntax and satisfaction relation for each of the two logics.

Definition 2.8 (Syntax of PCTL). The syntax of PCTL state formulae over a set of atomic propositions AP is given by the following rules:

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \mathbb{P}_J(\varphi)$$

where $a \in AP$, φ is a PCTL path formula and $J = [a, b] \subseteq [0, 1]$ with $a, b \in \mathbb{Q}$.

PCTL path formulae are composed according to the grammar:

$$\varphi ::= \mathbf{X}\Phi \mid \Phi_1 \mathbf{U} \Phi_2 \mid \Phi_1 \mathbf{U}^{\leq k} \Phi_2$$

where Φ, Φ_1 and Φ_2 are state formulae and $k \in \mathbb{N}$. □

Compared to CTL, the only new operator is $\mathbb{P}_J(\varphi)$. Given a state s , it asserts that the probability mass of the paths that start in s and satisfy the path formula φ must lie in the interval J . So φ can be thought of as a criterion that precisely defines a (measurable) set of paths and \mathbb{P}_J establishes bounds on the probability of this set of paths. We will now define the satisfaction relation of PCTL formulae formally.

Definition 2.9 (Satisfaction Relation of PCTL for DTMCs). Let $\mathcal{D} = (S, \mathbf{P}, s_{init}, AP, L)$ be a DTMC and $s \in S$ be a state. Then the satisfaction relation of state formulae is given by

$$\begin{aligned} s &\models true \\ s &\models a \quad \text{iff } a \in L(s) \\ s &\models \neg\Phi \quad \text{iff } s \not\models \Phi \\ s &\models \Phi \wedge \Psi \quad \text{iff } s \models \Phi \text{ and } s \models \Psi \\ s &\models \mathbb{P}_J(\varphi) \quad \text{iff } Pr_{\mathcal{D}}(s \models \varphi) \in J, \end{aligned}$$

where $Pr_{\mathcal{D}}(s \models \varphi) = Pr_{\mathcal{D}}^s(\{\pi \in Paths_{\mathcal{D}}(s) \mid \pi \models \varphi\})$.

For a given path $\pi = s_0 s_1 s_2 \dots \in Paths_{\mathcal{D}}$, the satisfaction relation of path formulae is defined by

$$\begin{aligned}
\pi \models \mathbf{X} \Phi & \quad \text{iff} \quad \pi[1] \models \Phi \\
\pi \models \Phi \mathbf{U} \Psi & \quad \text{iff} \quad \exists j \geq 0. (\pi[j] \models \Psi \wedge \forall 0 \leq k < j. (\pi[k] \models \Psi)) \\
\pi \models \Phi \mathbf{U}^{\leq k} \Psi & \quad \text{iff} \quad \exists 0 \leq j \leq k. (\pi[j] \models \Psi \wedge \forall 0 \leq k < j. (\pi[k] \models \Psi)).
\end{aligned}$$

We will write $\mathcal{D} \models \Phi$ if $s_{init} \models \Phi$ for a state formula Φ . \square

Example 2.10. Reconsider Example 2.2. The formula

$$\Phi = \mathbb{P}_{\leq 0.5}(\text{true } \mathbf{U} \text{ fail})$$

asserts that the probability to eventually reach a state in which the proposition *fail* holds is at most 0.5. \square

For MDPs the definition must also incorporate the various schedulers. The $\mathbb{P}_J(\varphi)$ operator now requires the probability mass of all paths satisfying φ to be in the interval J for *every possible resolution of non-determinism*, i.e. for all schedulers.

Definition 2.11 (Satisfaction Relation of PCTL for MDPs). Let $\mathcal{M} = (S, Act, \mathbf{P}, s_{init}, AP, L)$ be an MDP and $s \in S$ be a state. The satisfaction relation is the same as for DTMCs (see Definition 2.9) except for

$$s \models \mathbb{P}_J(\varphi) \quad \text{iff} \quad Pr_{\mathcal{M}}^{\sigma}(s \models \varphi) \in J \text{ for all schedulers } \sigma \text{ for } \mathcal{M},$$

where $Pr_{\mathcal{M}}^{\sigma}(s \models \varphi) = Pr_{\mathcal{M}}^{s, \sigma}(\{\pi \in Paths_{\mathcal{M}}(s) \mid \pi \models \varphi\})$. Accordingly, we write $\mathcal{M} \models \Phi$ if $s_{init} \models \Phi$ for a state formula Φ . \square

PCTL can be used to express properties of discrete-time Markov models, but it does not, however, account for the continuous-time aspects of CTMCs. An appropriate logic for this purpose is CSL.

Definition 2.12 (Syntax of CSL). The syntax of CSL state formulae over a set of atomic propositions AP is given by the following rules:

$$\Phi ::= \text{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid \mathbb{P}_J(\varphi) \mid \mathbb{S}_J(\Phi)$$

where $a \in AP$, φ is a CSL path formula and $J = [a, b] \subseteq [0, 1]$ with $a, b \in \mathbb{Q}$.

CSL path formulae are composed according to the following grammar:

$$\varphi ::= \mathbf{X}^{[t_1, t_2]} \Phi \mid \Phi_1 \mathbf{U}^{[t_1, t_2]} \Phi_2$$

where Φ, Φ_1 and Φ_2 are state formulae and $t_1 \in \mathbb{R}_{\geq 0}, t_2 \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ such that $t_1 \leq t_2$ or $t_2 = \infty$. \square

The following definition formally captures the satisfaction relation of CSL formulae for CTMCs.

Definition 2.13 (Satisfaction Relation of CSL for CTMCs). Let $\mathcal{C} = (S, \mathbf{R}, s_{init}, AP, L)$ be a CTMC and $s \in S$ a state. Then the satisfaction relation of state formulae is given by

$$\begin{aligned}
 s &\models \text{true} \\
 s &\models a && \text{iff } a \in L(s) \\
 s &\models \neg\Phi && \text{iff } s \not\models \Phi \\
 s &\models \Phi \wedge \Psi && \text{iff } s \models \Phi \text{ and } s \models \Psi \\
 s &\models \mathbb{P}_J(\varphi) && \text{iff } Pr_{\mathcal{C}}(s \models \varphi) \in J \\
 s &\models \mathbb{S}_J(\Phi) && \text{iff } \lim_{t \rightarrow \infty} Pr_{\mathcal{C}}(s + t \models \Phi) \in J,
 \end{aligned}$$

where

- $Pr_{\mathcal{C}}(s \models \varphi) = Pr_{\mathcal{C}}^s(\{\pi \in Paths_{\mathcal{C}}(s) \mid \pi \models \varphi\})$, and
- $Pr_{\mathcal{C}}(s + t \models \Phi) = Pr_{\mathcal{C}}^s(\{\pi \in Paths_{\mathcal{C}}(s) \mid \pi @ t \models \Phi\})$.

For a given path $\pi = s_0 t_0 s_1 t_1 \dots \in Paths_{\mathcal{C}}$, the satisfaction relation of path formulae is defined by

$$\begin{aligned}
 \pi &\models \mathbf{X}^I \Phi && \text{iff } \pi[1] \models \Phi \text{ and } t_0 \in I \\
 \pi &\models \Phi \mathbf{U}^I \Psi && \text{iff } \exists t \in I. (\pi @ t \models \Psi \wedge \forall 0 \leq t' < t. (\pi @ t' \models \Psi)).
 \end{aligned}$$

□

PCTL and CSL suffice to express various properties of interest of DTMCs, CTMCs and MDPs, but they alone are not suitable to express properties of MRMs. For this purpose, there exist reward extensions of both logics, namely Probabilistic Reward Computation Tree Logic (PRCTL) and Continuous Stochastic Reward Logic (CSRL) for PCTL and CSL, respectively. For details, we refer to [8, 9].

Given a Markov model \mathcal{M} with state space $S_{\mathcal{M}}$ and a formula Φ , the model checking problem is to determine whether $\mathcal{M} \models \Phi$. If this is the case, the model is said to fulfill the property, otherwise the model violates it.

If F is a set of state formulae of the logic $X \in \{PCTL, CSL, PRCTL, CSRL\}$ and $s \in S_{\mathcal{M}}$, we denote by X_F the smallest set of formulae that contains F and is closed under the logical connectives. $Sat_F(s)$ is the subset of formulae $F' \subseteq F$ that are satisfied by s , formally $Sat_F(s) = F' = \{\Phi \in F \mid s \models \Phi\}$. In particular, note that X_{AP} coincides with X . In addition, we let $Sat_{\Phi}(\mathcal{M}) = \{s \in S_{\mathcal{M}} \mid s \models \Phi\}$ be the set of all states that satisfy Φ and $Sat_F(\mathcal{M}) = \{s \in S_{\mathcal{M}} \mid s \in Sat_{\Phi}(\mathcal{M}) \text{ for all } \Phi \in F\}$ be the set of states that satisfy all formulae in F .

2.3. (Strong) Probabilistic Bisimulation

The state space explosion problem often impairs the possibility to model check realistic systems. But at the same time there often is a certain redundancy in these models, for example originating from multiple copies of certain components. Hence, it is very desirable to remove that redundancy, but still maintaining all logical properties of interest. One well-known approach for this is bisimulation minimization [1]. The idea is to lump states of the original system that exhibit the same (stepwise) behavior to new blocks. These blocks form the state space of the minimized system.

Example 2.14. Consider the states s_3 and s_6 in the DTMC of Example 2.2. Both behave equivalently in the sense that the probability to be in a state satisfying the proposition *fail* after the next step is 0.01 and the probability to return to the initial state is 0.99. \square

We will now formalize what it means for states to exhibit the same stepwise behavior by means of equivalence relations. Given a set S and an equivalence relation $\sim \subseteq S \times S$, we denote by $[s]_\sim$ the equivalence class of $s \in S$ and with S/\sim the set of all equivalence classes of S under \sim , i.e. $[s]_\sim = \{s' \in S \mid s \sim s'\}$ and $S/\sim = \{[s]_\sim \mid s \in S\}$. For a function $\mu : S \rightarrow \mathbb{R}$, we let $[\mu]_\sim : S/\sim \rightarrow \mathbb{R}$ be the quotient function with respect to \sim , which is given by $[\mu]_\sim([s]_\sim) = \sum_{s' \in [s]_\sim} \mu(s')$. Two functions μ, μ' are called equivalent with respect to \sim , denoted by $\mu \equiv_\sim \mu'$, if their corresponding quotient functions coincide, i.e. if $[\mu]_\sim = [\mu']_\sim$.

Definition 2.15 (Strong Probabilistic Bisimulation on Markov Models). Let $\mathcal{D} = (S_{\mathcal{D}}, \mathbf{P}_{\mathcal{D}}, s_{init}^{\mathcal{D}}, AP_{\mathcal{D}}, L_{\mathcal{D}})$ be a DTMC and F be a set of PCTL state formulae. A (strong) probabilistic F -bisimulation on \mathcal{D} is an equivalence relation $\sim^F \subseteq S_{\mathcal{D}} \times S_{\mathcal{D}}$ such that for each $(s_1, s_2) \in \sim^F$

- $Sat_F(s_1) = Sat_F(s_2)$, and
- $\mathbf{P}_{\mathcal{D}}(s_1, \cdot) \equiv_{\sim^F} \mathbf{P}_{\mathcal{D}}(s_2, \cdot)$

Let $\mathcal{C} = (S_{\mathcal{C}}, \mathbf{R}_{\mathcal{C}}, s_{init}^{\mathcal{C}}, AP_{\mathcal{C}}, L_{\mathcal{C}})$ be a CTMC and F be a set of CSL state formulae. A (strong) probabilistic F -bisimulation on \mathcal{C} is an equivalence relation $\sim^F \subseteq S_{\mathcal{C}} \times S_{\mathcal{C}}$ such that for each $(s_1, s_2) \in \sim^F$

- $Sat_F(s_1) = Sat_F(s_2)$, and
- $\mathbf{R}_{\mathcal{C}}(s_1, \cdot) \equiv_{\sim^F} \mathbf{R}_{\mathcal{C}}(s_2, \cdot)$

Let $\mathcal{M} = (S_{\mathcal{M}}, Act_{\mathcal{M}}, \mathbf{P}_{\mathcal{M}}, s_{init}^{\mathcal{M}}, AP_{\mathcal{M}}, L_{\mathcal{M}})$ be an MDP and F be a set of PCTL state formulae. A (strong) probabilistic F -bisimulation on \mathcal{M} is an equivalence relation $\sim^F \subseteq S_{\mathcal{M}} \times S_{\mathcal{M}}$ such that for each $(s_1, s_2) \in \sim^F$

- $Sat_F(s_1) = Sat_F(s_2)$, and
- for all $a_1 \in Act(s_1)$ there exists $a_2 \in Act(s_2)$ with $\mathbf{P}_{\mathcal{M}}(s_2, a_2, \cdot) \equiv_{\sim^F} \mathbf{P}_{\mathcal{M}}(s_1, a_1, \cdot)$.

\square

So, for DTMCs and CTMCs two states s_1 and s_2 behave equivalently, if they satisfy the same formulae of F and agree on the outgoing probabilities with respect to each equivalence class. For MDPs, the second criterion is altered such that it demands the existence of an equivalent distribution with respect to \sim^F in $\mu' \in Dist_{\mathcal{M}}(s_2)$ for every distribution $\mu \in Dist_{\mathcal{M}}(s_1)$.

Note, that for $F = AP$, $Sat_F(s_1) = Sat_F(s_2)$ holds if and only if $L(s_1) = L(s_2)$ for all of the three model types. In the sequel, if $F = AP$ we omit F altogether and, likewise,

we call a strong probabilistic (F -)bisimulation a probabilistic (F -)bisimulation or just (F -)bisimulation, as we do not consider other bisimulation types.

For a Markov model \mathcal{M} with state space $S_{\mathcal{M}}$ and $s_1, s_2 \in S_{\mathcal{M}}$, s_1 and s_2 are called F -bisimilar, denoted by $s_1 \sim_{\mathcal{M}}^F s_2$, if there exists an F -bisimulation \sim^F such that $(s_1, s_2) \in \sim^F$. Note, that this implies that $\sim_{\mathcal{M}}^F$ itself is an equivalence relation.

Given an F -bisimulation \sim^F , the state space of the quotient system \mathcal{M}/\sim^F consists of the equivalence classes of \sim^F . Because all states in each equivalence class agree on their stepwise behavior with respect to S/\sim , the resulting system is of the same type as the original system.

Formally:

Definition 2.16 (Bisimulation Quotient). Let $\mathcal{D} = (S_{\mathcal{D}}, \mathbf{P}_{\mathcal{D}}, s_{init}^{\mathcal{D}}, AP_{\mathcal{D}}, L_{\mathcal{D}})$ be a DTMC, F a set of PCTL state formulae and \sim^F an F -bisimulation on \mathcal{D} . Then

$$\mathcal{D}/\sim^F = (S_{\mathcal{D}}/\sim^F, \mathbf{P}'_{\mathcal{D}}, [s_{init}^{\mathcal{D}}]_{\sim^F}, AP_{\mathcal{D}}, L'_{\mathcal{D}})$$

with

- $L'_{\mathcal{D}}([s]_{\sim^F}) = Sat_F(s)$, and
- $\mathbf{P}'_{\mathcal{D}}([s]_{\sim^F}, \cdot) = [\mathbf{P}_{\mathcal{D}}(s, \cdot)]_{\sim^F}$

is the quotient DTMC. Note that this is well-defined, as for $s_1 \sim^F s_2$ it is guaranteed that $Sat_F(s_1) = Sat_F(s_2)$ and $\mathbf{P}_{\mathcal{D}}(s_1, \cdot) \equiv_{\sim^F} \mathbf{P}_{\mathcal{D}}(s_2, \cdot)$.

Let $\mathcal{C} = (S_{\mathcal{C}}, \mathbf{R}_{\mathcal{C}}, s_{init}^{\mathcal{C}}, AP_{\mathcal{C}}, L_{\mathcal{C}})$ be a CTMC, F a set of CSL state formulae and \sim^F an F -bisimulation on \mathcal{C} . Then

$$\mathcal{C}/\sim^F = (S_{\mathcal{C}}/\sim^F, \mathbf{R}'_{\mathcal{C}}, [s_{init}^{\mathcal{C}}]_{\sim^F}, AP_{\mathcal{C}}, L'_{\mathcal{C}})$$

with

- $L'_{\mathcal{C}}([s]_{\sim^F}) = Sat_F(s)$, and
- $\mathbf{R}'_{\mathcal{C}}([s]_{\sim^F}, \cdot) = [\mathbf{R}_{\mathcal{C}}(s, \cdot)]_{\sim^F}$

is the quotient CTMC. As for DTMCs, this is well-defined for similar reasons.

Let $\mathcal{M} = (S_{\mathcal{M}}, Act_{\mathcal{M}}, \mathbf{P}_{\mathcal{M}}, s_{init}^{\mathcal{M}}, AP_{\mathcal{M}}, L_{\mathcal{M}})$ be an MDP, F a set of PCTL state formulae and \sim^F an F -bisimulation on \mathcal{M} . Then

$$\mathcal{M}/\sim^F = (S_{\mathcal{M}}/\sim^F, Act_{\mathcal{M}}, \mathbf{P}'_{\mathcal{M}}, [s_{init}^{\mathcal{M}}]_{\sim^F}, AP_{\mathcal{M}}, L'_{\mathcal{M}})$$

with

- $L'_{\mathcal{M}}([s]_{\sim^F}) = Sat_F(s)$, and
- there exists $a \in Act_{\mathcal{M}}$ with $\mathbf{P}'_{\mathcal{M}}([s]_{\sim^F}, a, \cdot) = \mu'$ if and only if $\mu' = [\mu]_{\sim^F}$ for some $\mu \in Dist_{\mathcal{M}}(s)$.

is the quotient MDP. □

For DTMCs and CTMCs this definition is straightforward in the sense that all states in an equivalence class $[s]_{\sim}$ have exactly the same stepwise behavior with respect to \sim^F . This uniquely fixes the behavior of the state corresponding to $[s]_{\sim^F}$, which simply mimics the behavior of all contained states. For MDPs, however, this situation is slightly more involved. Let $[Dist_{\mathcal{M}}(s)]_{\sim^F} = \{[\mu]_{\sim^F} \mid \mu \in Dist_{\mathcal{M}}(s)\}$ be the set of available quotient distributions in a state s . While Definition 2.15 guarantees, that for $s \sim^F s'$ the sets $[Dist_{\mathcal{M}}(s)]_{\sim^F}$ and $[Dist_{\mathcal{M}}(s')]_{\sim^F}$ coincide, it does not ensure that both states associate the probability distributions with the same action labels. Hence, the quotient system cannot keep the connection between action labels and distributions, because different states might interpret action labels differently. In essence, this means that the quotient system is not uniquely determined by the above definition, as it leaves open which of the action labels is associated with which distribution. This is justified by the fact that we only consider state-based rewards in PRCTL and properties are thus indifferent to the concrete names of the actions, but only depend on the set of available distributions in each state.

Finally, we can transfer the above definitions to Markov reward models in a canonical way.

Definition 2.17 (Strong Probabilistic F -bisimulation on Markov Reward Models). Let $\mathcal{M}_r = (\mathcal{M}, r)$ be a Markov reward model with state space S and F be a set of state formulae of *PRCTL* if \mathcal{M} is a DTMC or MDP and of *CSRL* if \mathcal{M} is a CTMC. Then $\sim^F \subseteq S \times S$ is an F -bisimulation on \mathcal{M}_r if

- \sim^F is an F -bisimulation on \mathcal{M} , and
- for any $s_1, s_2 \in S$ with $(s_1, s_2) \in \sim^F$ we require $r(s_1) = r(s_2)$.

□

The additional (second) condition on the reward model ensures that no two states are said to behave equivalently, if they provide a different reward upon a visit to them. Intuitively, this prevents states from getting merged in the quotient system, if they have a different influence on the reward measure of a path passing through the respective states, because otherwise the state corresponding to their equivalence class could not correctly capture their reward behavior. We will use previously introduced denotations, such as, e.g., $\sim_{\mathcal{M}}$ and $\sim_{\mathcal{M}}^F$, also for Markov reward models.

Also, the quotient system of a Markov reward model is defined in a straightforward manner:

Definition 2.18 (Bisimulation Quotient of Markov Reward Models). Let $\mathcal{M}_r = (\mathcal{M}, r)$ be an MRM with state space S and \sim^F an F -bisimulation on \mathcal{M}_r . Then

$$\mathcal{M}_r / \sim^F = (\mathcal{M} / \sim^F, r / \sim^F)$$

where $r / \sim^F: S / \sim^F \rightarrow \mathbb{R}_{\geq 0}$ with $r / \sim^F([s]_{\sim^F}) = r(s)$, is the quotient MRM. □

Note that r/\sim^F is well defined because of the second condition of Definition 2.17.

As the quotient system is smaller than the original system for non-trivial bisimulations, it is desirable to solve the model checking problem on the quotient system and draw conclusions about the original system. Aziz et al. [10] showed that all PCTL formulae are preserved under bisimulation on DTMCs. As similar results hold for MDPs, CSL and bisimulation on CTMCs [12], for PRCTL on DMRMs, and for CSRL on CM-RMs [4], we can formulate the following preservation theorem.

Theorem 2.19 (Preservation of PRCTL/CSRL under probabilistic bisimulation). *Let \mathcal{M}_r be a DMRM with state space $S_{\mathcal{M}}$, F a set of PRCTL state formulae, $\Phi_{\mathcal{M}_r}$ a PRCTL _{F} state formula and \sim^F a probabilistic F -bisimulation on \mathcal{M}_r . Then for all $s, s' \in S_{\mathcal{M}}$ with $s \sim^F s'$, it holds that*

$$s \models \Phi_{\mathcal{M}_r} \iff s' \models \Phi_{\mathcal{M}_r}.$$

In particular, this implies

$$\mathcal{M}_r \models \Phi_{\mathcal{M}_r} \iff \mathcal{M}/\sim \models \Phi_{\mathcal{M}_r}.$$

Accordingly, let \mathcal{C}_r be a CMRM with state space $S_{\mathcal{C}}$, F a set of CSRL state formulae, $\Phi_{\mathcal{C}_r}$ a CSRL _{F} state formula and \sim^F a probabilistic F -bisimulation on \mathcal{C}_r . Then for all $s, s' \in S_{\mathcal{C}}$ with $s \sim^F s'$, it holds that

$$s \models \Phi_{\mathcal{C}_r} \iff s' \models \Phi_{\mathcal{C}_r}.$$

As before, this implies

$$\mathcal{C}_r \models \Phi_{\mathcal{C}_r} \iff \mathcal{C}/\sim \models \Phi_{\mathcal{C}_r}.$$

□

Note that these preservation results hold for any probabilistic F -bisimulation, in particular for $\sim^F_{\mathcal{M}_r}$ and $\sim_{\mathcal{M}_r}$ on a Markov reward model \mathcal{M}_r . Summing up, an F -bisimulation quotient preserves the logic fragment, that only uses formulae in F as subformulae. If $F = AP$ then all formulae are preserved.

Hence, given an MRM \mathcal{M}_r and a formula Φ , the model checking problem may be solved by first computing an F -bisimulation where F is chosen such that $\Phi \in PRCTL_F$ or $\Phi \in CSRL_F$, respectively, computing the corresponding quotient and then checking this usually smaller system against Φ . Figure 2.4 illustrates this approach.

This thesis will focus on the part of this process chain in the dashed box, namely the computation of an appropriate probabilistic F -bisimulation. Needless to say, for any MRM \mathcal{M}_r with state space $S_{\mathcal{M}}$, $\sim = \{(s, s) \mid s \in S_{\mathcal{M}}\}$ is a probabilistic bisimulation. This naive choice produces a quotient that is as big as the original system, but as the overall goal is to minimize the original system, it is obviously desirable to compute a coarser bisimulation. This implies two things: first, given a formula Φ , it is preferable to choose a small set F such that $\Phi \in PRCTL_F$ or $\Phi \in CSRL_F$, respectively, because this reduces the number of states that cannot be F -bisimilar according the first requirements

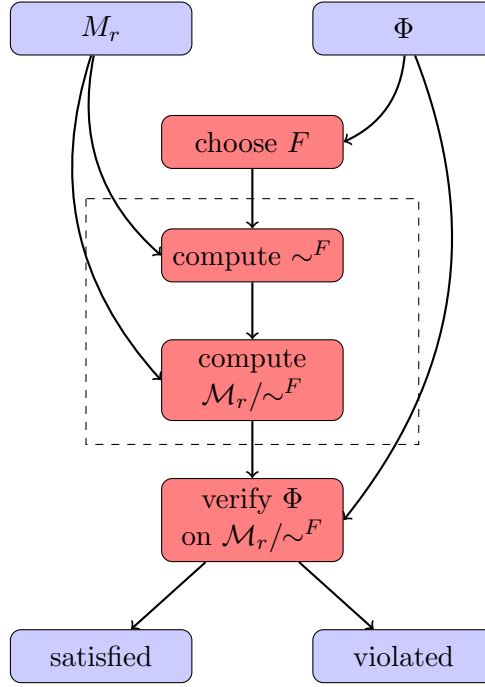


Figure 2.4.: Model checking using bisimulation minimization.

of Definition 2.15. And secondly, it is advantageous to find an F -bisimulation \sim^F that is as close to $\sim_{\mathcal{M}_r}^F$ as possible. Put differently, the target F -bisimulation should not assign states s_1 and s_2 to different equivalence classes if there exists an F -bisimulation that relates those states, i.e. $s_1 \sim_{\mathcal{M}_r}^F s_2$. Both considerations justify the usual computational approach, which is depicted in Figure 2.5.

The idea is to start with an initially coarse partition Π_{init} of the state space and then refining the blocks of that partition until no more refinement is needed. Each refinement step chooses a block B that is inconsistent, i.e. contains (at least) two states that do not exhibit the same stepwise behavior with respect to the current partition. This is "repaired" by splitting B into sub-blocks B_1, \dots, B_n such that the states in each of these sub-blocks behave equivalently. This procedure needs to be reiterated, because there may be more blocks that were inconsistent before and the splitting of B might lead to other blocks becoming inconsistent in the process. More concretely, it needs to be repeated until there is no inconsistent block remaining.

This approach leads to the coarsest F -bisimulation that respects the given initial partition Π_{init} . Assuming that Π_{init} is chosen canonically, i.e. contains only inclusion-maximal blocks such that every pair of states in each block agrees on the formulae of F that hold in these states, formally $\Pi_{init} = \Pi_F$ with

$$\Pi_F = \{\Gamma \subseteq S_{\mathcal{M}} \mid \forall s, s' \in \Gamma : Sat_F(s) = Sat_F(s') \wedge \nexists s'' \in \bar{\Gamma} : Sat_F(s) = Sat_F(s'')\}$$

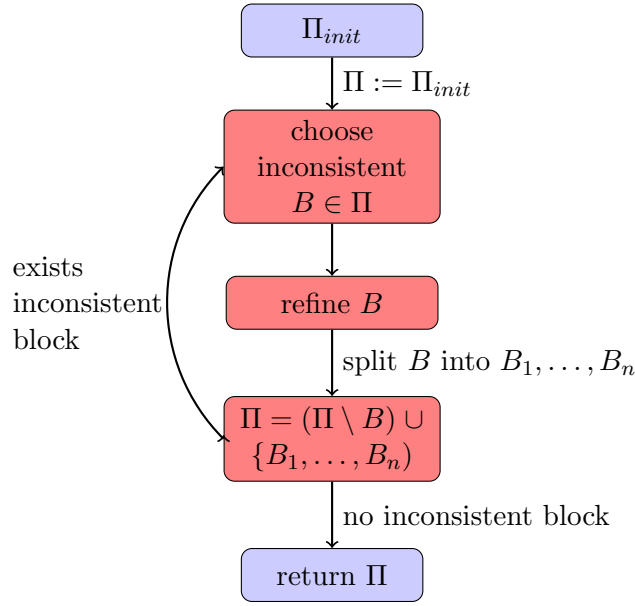


Figure 2.5.: Schematic of bisimulation computation via partition refinement.

where $\bar{\Gamma} = S_{\mathcal{M}} \setminus \Gamma$, it therefore computes $\sim_{\mathcal{M}_r}^F$. Given $b \in \{0, 1\}$, we let

$$\Phi^b = \begin{cases} \Phi & \text{if } b = 1 \\ \neg\Phi & \text{if } b = 0 \end{cases}$$

and for $F = \{\Phi_1, \dots, \Phi_n\}$

$$B(F) = \left\{ \bigwedge_{i=1}^n \Phi_i^{b_i} \mid (b_1, \dots, b_n) \in \{0, 1\}^n \right\}.$$

Then, the formulae in $B(F)$ capture exactly the blocks of Π_F in the sense that an inclusion-maximal set of states satisfies $\Phi \in B(F)$ if and only if it is an element of Π_F , formally

$$\Pi_F = \{Sat_{\Phi}(\mathcal{M}) \mid \Phi \in B(F)\}.$$

2.4. PRISM

This thesis aims to develop and evaluate techniques for the computation of probabilistic F -bisimulations in the framework of the PRISM¹ model checker [11]. Mainly implemented in Java with supplementary C/C++ libraries, it is a widely known state-of-the-art probabilistic model checker, initially developed at the University of Birmingham and

¹<http://www.prismmodelchecker.org>

now maintained by the University of Oxford. It provides a variety of features, some of which are particularly useful in the setting of our work. First, it features different model checking engines that build upon different representations in memory. In particular, it supports Binary Decision Diagrams (BDDs), which will be formally introduced later as a symbolic representation of the model to verify. Secondly, it has a high-level input language used to describe its models. This will play an important role in our second approach, because it crucially depends on reasoning at the language level. Finally, it implements advanced abstraction techniques, such as symmetry reduction.

We will now present a somewhat simplified version of the PRISM modeling language and define the corresponding semantics in terms of Markov (reward) models.

2.4.1. The PRISM modeling language

Let $Expr_{Var}$ denote the set of expressions over the variables contained in the finite set Var , which are typed according to a function

$$VarType : Var \rightarrow \{int[a, b] \mid a, b \in \mathbb{N} \text{ and } a \leq b\} \cup \{bool\}$$

as either $int[a, b]$ (integer with values between a and b , inclusively) or $bool$ (Boolean) and let $BExpr_{Var} \subseteq Expr_{Var}$ denote the set of Boolean expressions in $Expr_{Var}$. Given a set Var of variables with types $VarType$,

$$\begin{aligned} AP_{Var} &= \{x = c \mid x \in Var, VarType(x) = int[a, b], c \in [a, b]\} \\ &\cup \{x \mid x \in Var, VarType(x) = bool\} \end{aligned}$$

denotes the set of all atomic propositions over Var . In particular, note that this set is finite if Var is finite. We will, however, use shortcuts such as, for example, $x < y$ for the appropriate Boolean combination of formulae in AP_{Var} .

With $low(x)$ and $high(x)$ we denote the lower and upper bound, respectively, for a variable x , formally

$$low(x) = \begin{cases} a & \text{if } VarType(x) = int[a, b] \\ 0 & \text{if } VarType(x) = bool \end{cases}$$

and

$$high(x) = \begin{cases} b & \text{if } VarType(x) = int[a, b] \\ 1 & \text{if } VarType(x) = bool. \end{cases}$$

Furthermore, we let $b(Var) = \bigwedge_{v \in Var} low(v) \leq v \wedge v \leq high(v)$ be a Boolean expression that enforces the appropriate bounds for all variables. A valuation of the variables in Var is a function $\nu : Var \rightarrow \mathbb{N}$ that respects the types, i.e. for $x \in Var$ we require $\nu(x) \in [low(x), high(x)]$.

Then, we let

$$\Sigma(Var) = \{\nu \mid \nu \text{ is a valuation of } Var\}$$

be the set of all valuations over Var . We will often denote elements of $\Sigma(Var)$ with s , because this set will form the state space of the Markov model that is the semantics of

the PRISM model. For $e \in Expr_{Var}$ and $s \in \Sigma(Var)$, let $\llbracket e \rrbracket_s$ be the value of e in s . If $b \in BExpr_{Var}$ then we write $s \models b$ if and only if $\llbracket b \rrbracket_s = 1$ and we denote by $\llbracket b \rrbracket$ the set of all valuations that assign the truth value 1 to b , i.e. $\llbracket b \rrbracket = \{s \in \Sigma(Var) \mid s \models b\}$. Note that an $s \in \Sigma(Var)$ induces an expression $b(s) = \prod_{v \in Var} v = s(v) \in BExpr_{Var}$ that holds true only in s , i.e. $\llbracket b(s) \rrbracket = \{s\}$.

Definition 2.20 (Assignment). An *assignment* over a set of variables Var is a function $E : Var \rightarrow Expr_{Var}$ that comply with the respective types of the variables. \square

For a valuation $s \in \Sigma(Var)$ and an assignment E over Var , we write $s \xrightarrow{E} s'$ if and only if for all $v \in Var$ we have $\llbracket v \rrbracket_{s'} = \llbracket E(v) \rrbracket_s$ with the intuition that s is transformed into s' by updating the values of all variables according to E .

Definition 2.21 (Weakest Precondition). Given an expression $e \in Expr_{Var}$ and an assignment E over Var , $e[Var/E]$ denotes the expression that results from simultaneously replacing each occurrence of each variable $v \in Var$ in e by $E(v)$. $e[Var/E]$ is often referred to as the *weakest precondition* of e with respect to E and written $WP(e, E)$. \square

Definition 2.22 (Guarded Command). A guarded command

$$c = (a, g, (r_1, E_1), \dots, (r_n, E_n))$$

over a set of variables Var and a set of actions Act consists of

- an action $a \in Act$,
- a guard $g \in BExpr_{Var}$ and
- randomness information $r_i \in \mathbb{R}_{>0}$ associated with assignments E_i over Var for $1 \leq i \leq n$.

\square

The syntactical representation of c is

$$[a] g \longrightarrow r_1 : Var' = E_1 + \dots + r_n : Var' = E_n$$

where $Var' = E_i$ is short for a conjunction of entities of the form $v' = e$ where $v \in Var$ and $e = E_i(v)$.

Intuitively, a guarded command can be executed in every state that satisfies its guard. If it is executed, the i -th assignment is carried out with probability or rate r_i . For a guarded command $c = (a, g, (r_1, E_1), \dots, (r_n, E_n))$ and valuations $s, s' \in \Sigma(Var)$, we let $c(s, s') = \{i \mid s \xrightarrow{E_i} s'\}$ be the set of indices whose corresponding updates transform s into s' . Additionally, let a_c denote its action and g_c its guard.

Definition 2.23 (PRISM reward model). A PRISM reward model over a set of variables Var is a (possibly empty) set

$$Rew \subseteq BExpr_{Var} \times \mathbb{R}_{\geq 0}$$

such that for all $(b_1, r_1), (b_2, r_2) \in Rew$ and $s \in \Sigma(Var)$

$$s \models b_1 \wedge s \models b_2 \quad \Rightarrow \quad (b_1, r_1) = (b_2, r_2).$$

□

The condition imposed on the elements of a PRISM reward model Rew guarantees that no state is assigned multiple different rewards. For a PRISM reward model Rew , we let $AF(Rew) = \{b \mid (b, r) \in Rew \text{ for some } r \in \mathbb{R}_{\geq 0}\}$.

Now, a probabilistic program combines a set of typed variables, a set of guarded commands and a PRISM reward model in the following way.

Definition 2.24. A probabilistic program

$$P = (ModelType, Var, VarType, init, Act, Comm, Rew)$$

consists of

- a model type $ModelType \in \{dtmc, ctmc, mdp\}$,
- a finite set Var of variables,
- a mapping $VarType : Var \rightarrow \{int[a, b] \mid a, b \in \mathbb{N} \text{ and } a < b\} \cup \{bool\}$ of variables to types,
- an expression $init = b(s)$ for some $s \in \Sigma(Var)$,
- a finite set of actions Act ,
- a finite set of guarded commands $Comm$ over Var and Act such that for every two guarded commands $c_1, c_2 \in Comm$, $c_1 \neq c_2$ implies $a_{c_1} \neq a_{c_2}$, and
- a PRISM reward model Rew over Var .

If $ModelType = dtmc$ or $ModelType = mdp$, we require

$$c = (a, g, (p_1, E_1), \dots, (p_n, E_n)) \in Comm \Rightarrow \sum_{1 \leq i \leq n} p_i = 1$$

and if $ModelType = dtmc$ or $ModelType = ctmc$ we additionally impose that for each $s \in \Sigma(Var)$ there is at most one $c \in Comm$ with $s \models g_c$. □

Example 2.25. Let P_{Ex} be the probabilistic program depicted in Figure 2.6.

It models a probabilistic algorithm over the variables $Var_{Ex} = \{c, h, f, r\}$ whose bounds expression $b(Var_{Ex})$ is given by

$$b(Var_{Ex}) = 1 \leq c \wedge c \leq 4 \wedge \bigwedge_{v \in Var \setminus \{c\}} 0 \leq v \wedge v \leq 1.$$

c	:	int[1, 4] init 1;
h	:	bool init false;
f	:	bool init false;
r	:	bool init false;
[<i>coin</i>] $c = 1$ \longrightarrow 0.5 : $c' = c + 1 \wedge h' = \neg h$ + 0.5 : $c' = c + 1 \wedge h' = h$;		
[<i>process</i>] $c = 2$ \longrightarrow 0.2 : $c' = c + 1 \wedge f' = \neg f$ + 0.8 : $c' = c + 1 \wedge f' = f$;		
[<i>return1</i>] $c = 3 \wedge h \wedge \neg f$ \longrightarrow 0.2 : $c' = c + 1 \wedge r' = 0 \wedge f' = 1$ + 0.8 : $c' = c + 1 \wedge r' = 1$;		
[<i>return2</i>] $c = 3 \wedge \neg h \wedge \neg f$ \longrightarrow 0.5 : $c' = c + 1 \wedge r' = 0 \wedge f' = 1$ + 0.5 : $c' = c + 1 \wedge r' = 1$;		
[<i>restart</i>] $c = 3 \wedge f$ \longrightarrow 0.99 : $c' = 1 \wedge h' = 0 \wedge f' = 0 \wedge r' = 0$ + 0.01 : $c' = c + 1$;		
[<i>done</i>] $c = 4$ \longrightarrow 1 : $c' = c$;		

Figure 2.6.: The probabilistic program P_{Ex} .

Note that for a Boolean variable v we write v and $\neg v$ as a shortcut to $v = 1$ and $v = 0$, respectively. The algorithm starts with throwing a coin (h) before the processing step that, based on the outcome of the coin flip, has a different probability to fail (f). In case of a failure, the algorithm restarts with a high probability and erroneously terminates in rare cases. Otherwise, it returns a result r that is either true or false with a certain probability that depends on the coin flip. As false is the (supposedly) incorrect result, the fail flag is set in this case. \square

The semantics of a probabilistic program is an DMRM or a CMRM, depending on the model type of the program.

Definition 2.26 (Semantics of a Probabilistic Program). Let

$$P = (\text{ModelType}, \text{Var}, \text{VarType}, \text{init}, \text{Act}, \text{Comm}, \text{Rew})$$

be a probabilistic program. Furthermore, let $r : \Sigma(\text{Var}) \rightarrow \mathbb{R}_{\geq 0}$ be defined by

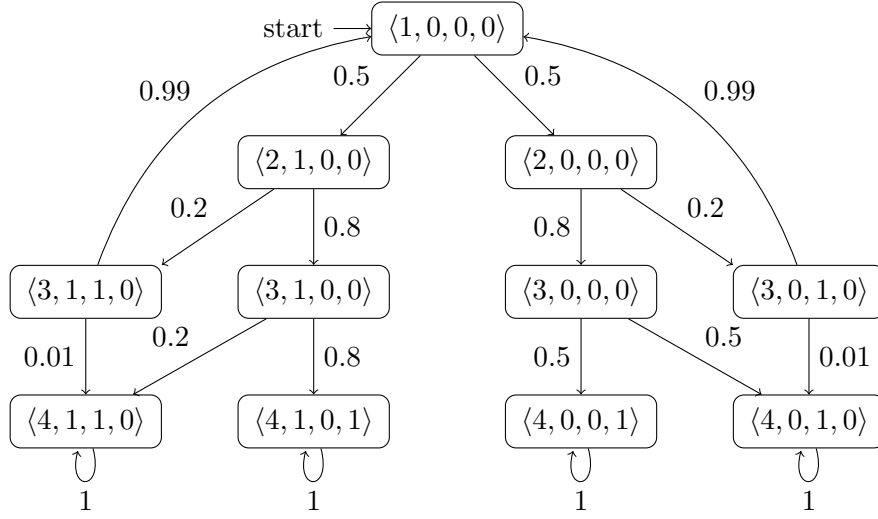
$$r(s) = \begin{cases} r' & \text{if there exists } (b', r') \in \text{Rew} \text{ such that } s \models b' \\ 0 & \text{otherwise,} \end{cases}$$

$L : \Sigma(\text{Var}) \rightarrow AP_{\text{Var}}$ given by $L(s) = \{b \in AP_{\text{Var}} \mid s \models b\}$ and $\text{init} = b(s)$ for $s \in \Sigma(\text{Var})$.

If $\text{ModelType} = dtmc$, the semantics of P is a DMRM

$$\llbracket P \rrbracket = \mathcal{D}_r = (\mathcal{D}, r) \text{ with a DTMC } \mathcal{D} = (\Sigma(\text{Var}), \mathbf{P}_{\mathcal{D}}, s, AP_{\text{Var}}, L),$$

where


 Figure 2.7.: The reachable state space of $\llbracket P_{Ex} \rrbracket$.

- for $s, s' \in S_{\mathcal{D}}$ and $c = (a, g, (p_1, E_1), \dots, (p_n, E_n)) \in Comm$ with $s \models g$, the transition probability is given by $\mathbf{P}_{\mathcal{D}}(s, s') = \sum_{i \in c(s, s')} p_i$.

If $ModelType = ctmc$, the semantics of P is a CMRM

$$\llbracket P \rrbracket = \mathcal{C}_r = (\mathcal{C}, r) \text{ with a CTMC } \mathcal{C} = (\Sigma(Var), \mathbf{R}_{\mathcal{C}}, s, AP_{Var}, L),$$

where

- for $s, s' \in S_{\mathcal{C}}$ and $c = (a, g, (r_1, E_1), \dots, (r_n, E_n)) \in Comm$ with $s \models g$, the transition rate is given by $\mathbf{R}_{\mathcal{C}}(s, s') = \sum_{i \in c(s, s')} r_i$,

If $ModelType = mdp$, the semantics of P is a DMRM

$$\llbracket P \rrbracket = \mathcal{M}_r = (\mathcal{M}, r) \text{ with an MDP } \mathcal{M} = (\Sigma(Var), Act, \mathbf{P}_{\mathcal{M}}, s, AP_{Var}, L),$$

where

- for $s, s' \in S_{\mathcal{M}}$ and $c = (a, g, (p_1, E_1), \dots, (p_n, E_n)) \in Comm$ with $s \models g$, the transition probability is given by $\mathbf{P}_{\mathcal{M}}(s, a, s') = \sum_{i \in c(s, s')} p_i$.

□

Example 2.27. The reachable part of the state space of the probabilistic program P_{Ex} (see Example 2.25) is depicted in Figure 2.7, where the nodes are of the form $\langle c, h, f, r \rangle$.

□

Unlike model checkers that build upon a low-level formalism, such as MRMC², PRISM does not treat states as abstract entities with labels, but rather as the underlying valuations of variables. These are used to derive all atomic propositions that hold in a certain state. Recall that an AP -bisimulation will not lump states that have a different set of labels attached to them. For a probabilistic program P with reward model Rew , $(\mathcal{M}, r) = \llbracket P \rrbracket$ and a formula Φ , an AP_{Var} -bisimulation will obviously produce the original system as the quotient system, as no two states in \mathcal{M} have exactly the same labels attached to them and, hence, no lumping would occur. In other words, in order to obtain a quotient system that is possibly smaller than the original system, it is crucial to find a set of formulae F , such that

1. $\Phi \in PRCTL_F$ or $\Phi \in CSRL_F$, respectively,
2. $Sat_F(s) = Sat_F(s') \Rightarrow r(s) = r(s')$,
3. $Sat_F(s) = Sat_F(s') \not\Rightarrow L(s) = L(s')$ for $s, s' \in S$

and compute an F -bisimulation \sim^F . While the first two conditions ensure that \sim^F is a proper F -bisimulation on \mathcal{M}_r (and hence the preservation of Φ in the quotient system \mathcal{M}_r / \sim^F), the last requirement is necessary, because otherwise only trivial F -bisimulations would be computed, which would in effect not reduce the size of the original model. An obvious candidate for F is the set

$$AF(P, \Phi) = AF(\Phi) \cup AF(\text{Rew}) \subseteq BExpr_{Var}.$$

Including $AF(\Phi)$ ensures $\Phi \in PRCTL_F$ or $CSRL_F$, respectively and $AF(\text{Rew})$ guarantees that two states $s_1 \sim_{\mathcal{M}}^F s_2$ agree on their reward value. In particular, note that from the point of view of bisimulation minimization, preserving reward values is no different than other formulae, i.e. $AF(\text{Rew})$ can be seen as an additional set of atomic propositions. Depending on Φ and Rew , the third condition is not necessarily guaranteed, but it will only be violated in case all states of the original system have to be kept separate in order to preserve the verification result of Φ in the quotient system.

2.5. Preliminaries of the BDD-based approach

2.5.1. General

As stated earlier, the transition functions of Markov models can be interpreted as matrices. In fact, this is the usual approach adopted by probabilistic model checkers, because the numerical operations needed in the model checking process can be elegantly formulated in this way. Model checkers that use an explicit enumeration of the state space often employ sparse matrix representations [13]. While often being very fast, this approach still has rather tight limits regarding the size of the model: state-of-the-art explicit model checkers can deal with state spaces of about 10^8 to 10^9 . Realistic models,

²<http://www.mrmc-tool.org>

however, often exceed this number by far, because of, e.g., combinatorial explosion, and thus give rise to the need for data structures that exploit regularities in the structure of the model. We will now focus on one of these so-called *symbolic data structures*, namely Binary Decision Diagrams (BDDs). More precisely, we will consider Multi-Terminal Binary Decision Diagrams (MTBDDs) as an extension thereof [14], because they are capable of representing arbitrary transition probabilities. PRISM offers a model checking engine that solely uses MTBDDs for both storing the model and numerical computations. Also, it features a symmetry reduction algorithm that exploits the MTBDD representation [15]. Given a component-wise symmetry, this technique can compute a bisimulation quotient very efficiently. What it lacks, however, is the possibility to compute a bisimulation quotient in absence of the aforementioned symmetry and, in case the symmetry is present, to compute a coarser F -bisimulation. In chapter 3 we will close this gap and evaluate the effectiveness of this new feature. Now, we will describe the underlying formalism a little further.

2.5.2. Formalism

Given a set $\mathbb{B} = \{x_1, \dots, x_n\}$ of Boolean variables and a total order $x_1 \prec x_2 \prec \dots \prec x_n$, an (MT)BDD M over \mathbb{B} is a directed acyclic graph with one root node r , denoted $r = \text{root}(M)$. The inner nodes of M are labeled with an $x_i \in \mathbb{B}$ such that v in M with $\text{lab}(v) = x_j$ has exactly two successor nodes, which are either leaf nodes or have $\text{lab}(w) = x_k$ with $k > j$, where $\text{lab}(v) \in \mathbb{B}$ denotes the variable with which the node v is labeled. Additionally, the leaf nodes l of M are labeled with values $\text{lab}(l)$ from a certain domain, which will be the real numbers \mathbb{R} in our setting. M can then be used to represent a function $f_M(x_1, \dots, x_n) : \{0, 1\}^m \rightarrow \mathbb{R}$ in the following way. Each node v of M has two outgoing edges to other nodes, a 0-arc and a 1-arc to v'_0 and v'_1 , respectively. To determine the function value of $f_M(b_1, \dots, b_n)$ with $b_i \in \{0, 1\}$ for $1 \leq i \leq n$, we start at the top node r and read its label $x_i = \text{lab}(r)$. If $b_i = 0$, we follow the 0-arc and otherwise the 1-arc to the next node and repeat the procedure until we arrive at leaf node l . The desired function value then is $\text{lab}(l)$. Justified by this procedure, we will denote v'_0 and v'_1 by $v|_{\text{lab}(v)=0}$ and $v|_{\text{lab}(v)=1}$, respectively. We will often identify f_M with M and write $M(x_1, \dots, x_n)$ for $f_M(x_1, \dots, x_n)$. If $\mathbb{B}' = \{x_1, \dots, x_m\} \subseteq \mathbb{B}$ and $\underline{b} = (b_1, \dots, b_m) \in \{0, 1\}^m$, this induces a path through M to the node w , whose target node we denote by $w = r|_{x_1=b_1, \dots, x_m=b_m}$ or $w = r|_{\mathbb{B}'=\underline{b}}$ for short. The sub-BDD M_w of M rooted by w then represents the function $f_{M_w}(x_{m+1}, \dots, x_n) = f_M(b_1, \dots, b_m, x_{m+1}, \dots, x_n)$.

As we fix \mathbb{R} as the target domain, we can use the usual arithmetic operations, most importantly multiplication, on functions also over BDDs in the canonical way. We will, however, need two additional operations.

Let M be an MTBDD over the variables \mathbb{B} and $\mathbb{B}' = \{x_j, \dots, x_m\} \subseteq \mathbb{B}$. Then we let

$$\begin{aligned} \exists \mathbb{B}' M(x_1, \dots, x_n) &= M'(x_1, \dots, x_{j-1}, x_{m+1}, \dots, x_n) \\ &= \sum_{(b_j, \dots, b_m) \in \{0, 1\}^{m-j}} M(x_1, \dots, x_{j-1}, b_1, \dots, b_m, x_{m+1}, x_n) \end{aligned}$$

be the existential abstraction of M with respect to \mathbb{B}' . Likewise, if M is an MTBDD over \mathbb{B} and $\mathbb{B}'' = \{y_1, \dots, y_n\}$ is a set of Boolean variables such that $\mathbb{B} \cap \mathbb{B}'' = \emptyset$ and

$|\mathbb{B}| = |\mathbb{B}''|$, then $M[\mathbb{B} \rightarrow \mathbb{B}'']$ over the variables \mathbb{B}'' is the MTBDD that results from renaming the \mathbb{B} variables in M to \mathbb{B}'' variables.

2.5.3. State Space Representation Using MTBDDs

Let $\mathcal{D} = (S_{\mathcal{D}}, \mathbf{P}_{\mathcal{D}}, s_{init}^{\mathcal{D}}, AP_{\mathcal{D}}, L_{\mathcal{D}})$ be a DTMC, $n = \lceil \log_2 |S_{\mathcal{D}}| \rceil$ and $enc : S_{\mathcal{D}} \rightarrow \{0, 1\}^n$ be an injective encoding of the state space using n Boolean variables. Furthermore, let $\mathbb{S} = \{s_1, \dots, s_n\}$, $\mathbb{T} = \{t_1, \dots, t_n\}$ be two distinct sets of Boolean variables. The transition matrix $\mathbf{P}_{\mathcal{D}}$ can be represented as an MTBDD M over $\mathbb{S} \cup \mathbb{T}$ by letting

$$M(enc(s), enc(s')) = \mathbf{P}_{\mathcal{D}}(s, s')$$

for any two states $s, s' \in S_{\mathcal{D}}$. Similarly, the rate matrix of a CTMC can be represented in this way by choosing the respective rates as the function value instead of probabilities.

However, for an MDP $\mathcal{M} = (S_{\mathcal{M}}, Act, \mathbf{P}_{\mathcal{M}}, s_{init}, AP, L)$ the situation is slightly more involved, because the matrix $\mathbf{R}_{\mathcal{M}}$ to be represented is not square. Rather, it is a $|Act| \cdot |S_{\mathcal{M}}| \times |S_{\mathcal{M}}|$ matrix. For the MTBDD representation this means that additional variables $\mathbb{A} = \{a_1, \dots, a_m\}$ for $m = \lceil \log_2 |Act| \rceil$ have to be added, such that the MTBDD M representing the transitions is given by

$$M(enc(s), enc(s'), enc(a)) = \mathbf{P}_{\mathcal{M}}(s, a, s')$$

assuming a proper extension of enc for action names.

In the sequel, we will often omit enc and, for example, write $M(s, s')$ for $M(enc(s), enc(s'))$ and $M(s, s', a)$ for $M(enc(s), enc(s'), enc(a))$.

The order of variables plays a crucial role for the number of nodes needed to represent a certain function. The MTBDD engine of PRISM uses an interleaved order $(a_1 \prec \dots \prec a_n \prec) s_1 \prec t_1 \prec s_2 \prec t_2 \dots \prec s_n \prec t_n$ of the source state \mathbb{S} and target state variables \mathbb{T} , which proves to produce reasonably small MTBDDs.

2.5.4. Sigref: A Symbolic Bisimulation Tool

SIGREF³ is a tool that, given an MTBDD representation of a transition system (or in our setting, Markov models), is able to compute a variety of different bisimulations [16]. It implements a signature-based approach, significantly exploiting properties of the MTBDD representation in the process. The computation of the signature of a state, that is the stepwise behavior with respect to the current partition, and the refinement of a block is reduced to simple MTBDD operations and a depth-first traversal of the graph structure of the diagram, respectively. Besides being freely available, SIGREF was particularly appealing, because it uses the same MTBDD library as PRISM, namely CUDD⁴, and, hence, was promising regarding the ease of integration.

³<http://sigref.gforge.avacs.org/>

⁴<http://vlsi.colorado.edu/~fabio/CUDD/>

3. BDD-based bisimulation minimization

Sheldon Cooper: What are the odds that two individuals as unique as ourselves would be connected by someone as comparatively workaday as your son?

Beverly: Is that a rhetorical point, or would you like to do the math?

Sheldon Cooper: I'd like to do the math.

Beverly: I'd like that, too.

The Big Bang Theory - The Maternal Capacitance
(Season 2 - Episode 15)

In this chapter, we describe the integration of SIGREF, a symbolic bisimulation tool, into the PRISM model checker and subsequently evaluate the effectiveness of this approach by means of several case studies.

While being feature-rich in many aspects, PRISM lacks the capability of computing a coarse F -bisimulation quotient of a given system. MRMC, in contrast, provides this feature, but, being intended as a verification backend, lacks a high-level input language and does not employ symbolic data structures. The first shortcoming can be countered by using PRISM to build the state space of a model specified in its modeling language and then exporting it to the MRMC input format. Then, MRMC can be used to perform a bisimulation minimization prior to verification. Apart from relying on file exchange that is often infeasible for big models, this export builds upon explicitly enumerating all states of the (symbolically-stored) PRISM model during the export and the explicit state space representation used in MRMC, and thus is in large parts inherently non-symbolical. Finally, this approach suffers from another severe restriction: except for bounded reachability in CTMDPs, MRMC is limited to models without non-determinism and does not support neither the verification of MDPs nor their bisimulation minimization.

SIGREF complements PRISM by providing a particularly versatile (MT)BDD-based algorithm for computing a bisimulation quotient of a system. Sharing the same underlying BDD library, namely CUDD, they promised to be both, easily combinable and altogether avoiding the aforementioned issues. Figure 3.1 illustrates the process chain. First, the model specification is parsed by PRISM and an MTBDD representation of the system is built. After that, the BDD representation of \mathcal{M} and the initial partition Π_{init}

are passed as arguments to SIGREF, which then computes a bisimulation quotient and returns the minimized system to PRISM. It can then carry out the verification task on the quotient system.

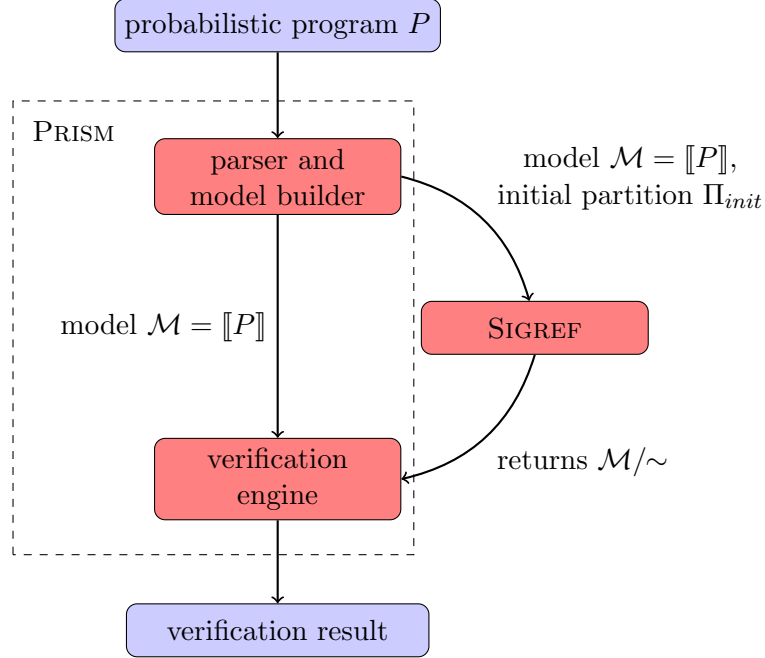


Figure 3.1.: Integration of SIGREF into the PRISM process chain.

We will now present a detailed description of the main steps of this integration process and the necessary modifications to PRISM as well as SIGREF, before we analyze the impact on model checking performance for some case studies. For the remainder of this chapter, let $P = (ModelType, Var, VarType, init, Act, Comm, Rew)$ be a probabilistic program and $\mathcal{M} = \llbracket P \rrbracket$ with state space S and labeling function L . Furthermore, let Φ be a PRCTL or CSRL formula, respectively, that is to be verified on \mathcal{M} .

3.1. Creating the Sigref Input

3.1.1. The System-MTBDD

PRISM parses an input file that complies with the syntax of its modeling language and builds an MTBDD representation of the semantics. In particular, this involves the generation of an MTBDD $trans_{\mathcal{M}}$ that represents the transitions of the system as described in section 2.5.3. We will stick to the notation introduced there, and let \mathbb{S} , \mathbb{T} , \mathbb{A} , n and m be defined as before. Additionally, we will need the set $\mathbb{K} = \{k_1, \dots, k_n\}$ of auxiliary variables for the refinement procedure of SIGREF.

By default, PRISM uses the order

$$(a_1 \prec \dots \prec a_m \prec) s_1 \prec t_1 \prec \dots \prec s_n \prec t_n$$

for the variables in its internal BDD representation. For reasons that we will explain in section 3.2.2, SIGREF's algorithm for computing the bisimulation quotient requires the s_i variables to precede the a_i variables and the k_i variables to follow all other ones. For this, we changed the variable order PRISM uses to

$$s_1 \prec t_1 \prec \dots \prec s_n \prec t_n (\prec a_1 \prec \dots \prec a_m) \prec k_1 \prec \dots \prec k_n.$$

However, this has some impact: first, this clearly influences the size of the MTBDD as the variable order is crucial for this and secondly, because changing the variable order back to the original one after minimization turned out to be too computationally expensive, this rules out the possibility to use any other verification engine of PRISM than the MTBDD engine. For example, the hybrid engine, which usually gives a good compromise between space and time requirements, relies on the a_i preceding all other variables.

3.1.2. The Initial Partition

The Canonical Initial Partition

For the reasons we explained in section 2.4.1, we will compute an $AF(P, \Phi)$ -bisimulation. Using the usual partition refinement approach, we achieve this by taking $\Pi_{AF(P, \Phi)}$ as the initial partition. Given an expression $b \in BExpr_{Var}$, PRISM provides methods to obtain a BDD M_b such that $M_b(s) = 1$ if and only if $s \models b$. The initial partition can thus be symbolically represented by the set of BDDs

$$\mathcal{B}_{\Pi_{AF(P, \Phi)}} = \{M_b \mid b \in B(AF(P, \Phi))\}$$

Distance-sensitive Initial Partition

Reconsidering the definition of strong probabilistic F -bisimulation and the preservation results for PRCTL and CSRL, we make the observation that a bisimulation preserves the minimal number of transitions needed to reach a state $s' \models \Psi$ starting from a state s for all $\Psi \in F$. In other words, if we let $dist(s, \Psi)$ be the minimal distance to a state s' satisfying $\Psi \in F$ from $s \in S_{\mathcal{M}}$, i.e.

$$dist(s, \Psi) = \min \{i \in \mathbb{N} \mid \exists \pi \in Paths_{\mathcal{M}}(s) \text{ with } \pi[i] \models \Psi \text{ and } \forall 0 \leq j < i. \pi[j] \not\models \Psi\},$$

we observe that $dist(s, \Psi) \neq dist(s', \Psi)$ implies $s \not\sim_{\mathcal{M}}^F s'$. This can be seen as follows: without loss of generality, we assume $i = dist(s, \Psi) < dist(s', \Psi)$, which entails $s \models \mathbb{P}_{>0}(true \ \mathbf{U}^{\leq i} \ \Psi)$ and $s' \not\models \mathbb{P}_{>0}(true \ \mathbf{U}^{\leq i} \ \Psi)$. But then it follows directly from the preservation Theorem 2.19 that s and s' cannot be bisimilar. A similar argument

applies to CTMCs and CSRL. We use this knowledge to compute a finer initial partition that is *distance-sensitive* as follows. We can easily compute BDDs $M_{d_{\Psi};i}$ such that $M_{d_{\Psi};i}(s) = 1$ iff $dist(s, \Psi) = i$ by letting $M_{d_{\Psi};0} = M_{\Psi}$ and computing

$$M_{d_{\Psi};i+1} = \exists \mathbb{T} (trans_{\mathcal{M}} * M_{d_{\Psi};i} [\mathbb{S} \rightarrow \mathbb{T}]) [\mathbb{T} \rightarrow \mathbb{S}] \wedge \prod_{j=0}^{j=i} \neg M_{d_{\Psi};j},$$

for DTMCs and CTMCs and additionally existentially abstracting from \mathbb{A} for MDPs, where the star denotes the usual multiplication. Now we can treat the distances as atomic propositions by substituting L by $L'(s) = L(s) \cup \{d_{\Psi};i \mid dist(s, \Psi) = i\}$, letting

$$AF^{dist}(P, \Phi) = AF(P, \Phi) \cup \{d_{\Psi};i \mid \Psi \in AF(\Phi) \text{ and } i \leq \max_{s \in S} dist(s, \Psi)\}$$

and compute the corresponding set of BDDs $\mathcal{B}_{\Pi_{AF^{dist}(P, \Phi)}}$ as before.

3.2. Computing the Quotient

3.2.1. General

SIGREF pursues a *signature*-based approach to compute the sub-blocks into which a block needs to be split. A signature can be thought of as the footprint of a state with respect to the current partition and captures all information about the next-step behavior of that state. Given a function $sig^{\Pi}(s)$ that assigns the signature with respect to the partition Π to a state s , SIGREF computes

$$\Pi^{i+1} = \text{SIGREF}(\Pi^i) = \left\{ \{t \in S \mid sig^{\Pi^i}(s) = sig^{\Pi^i}(t)\} \mid s \in S \right\}$$

starting with $\Pi^0 = \Pi_{init}$ until a fixpoint is reached, i.e. until $\Pi^{n+1} = \Pi^n$ for some $n \in \mathbb{N}$. This approach is particularly versatile, because it only requires an appropriate re-definition of the sig^{Π} function to be able to deal with a variety of systems as well as bisimulations [16].

We will now present the appropriate sig^{Π} functions for Markov models.

Definition 3.1 (Signatures for Strong Bisimulation for DTMCs, CTMCs and MDPs). Let Π be a partition of S and $s \in S$.

If \mathcal{M} is a DTMC,

$$sig^{\Pi}(s) = sig_{\mathcal{D}}^{\Pi}(s) = \{ (p, B) \mid \mathbf{P}_{\mathcal{D}}(s, B) = p \text{ and } B \in \Pi \},$$

if \mathcal{M} is a CTMC,

$$sig^{\Pi}(s) = sig_{\mathcal{C}}^{\Pi}(s) = \{ (r, B) \mid \mathbf{R}_{\mathcal{D}}(s, B) = r \text{ and } B \in \Pi \},$$

and

$$sig^{\Pi}(s) = sig_{\mathcal{M}}^{\Pi}(s) = \{ \{ (p, B) \mid \mathbf{P}_{\mathcal{M}}(s, a, B) = p \text{ and } B \in \Pi \} \mid a \in Act_{\mathcal{M}} \}$$

if \mathcal{M} is an MDP. □

These signatures reflect the requirement that bisimilar states need to agree on their stepwise behavior (see Definition 2.15). Only states s and s' that have the same signature with respect to the current partition Π^i are possibly bisimilar and can be grouped into the same block in Π^{i+1} .

3.2.2. Symbolic Implementation

The Current Partition

In order to implement the whole algorithm symbolically, SIGREF needs to represent all its internal data symbolically, too. More precisely, the current partition $\Pi = \{B_1, \dots, B_j\}$ is represented as a BDD \mathcal{P}^Π over the variables $\mathbb{S} \cup \mathbb{K}$ such that $\mathcal{P}^\Pi(s, k) = 1$ if and only if $s \in B_k$. Usually, SIGREF uses consecutive numbers encoded by the variables in \mathbb{K} as block identifiers. In our setting, however, this is not possible, because these numbers will make up the state space of the quotient system. When the quotient is returned for the verification task, PRISM tries to interpret these encodings of numbers as valuations of the variables in the system, and therefore produces wrong results. Hence, we do not use consecutive numbers as block identifiers, but pick a representative, i.e. for $B \in \Pi$ we pick an $s_{repr}^B \in B$ and let $\mathcal{P}^\Pi(s, s_{repr}^B[\mathbb{S} \rightarrow \mathbb{K}]) = 1$ if and only if $s \in B$. Figure 3.2 sketches the form of the partition BDD \mathcal{P}^Π , where $s_{r,i}$ is the representative of the block B_i . All arcs that deviate from the encoding of $s_{r,i}[\mathbb{S} \rightarrow \mathbb{K}]$ in the i -th line lead to the constant 0-node and have been omitted to improve readability.

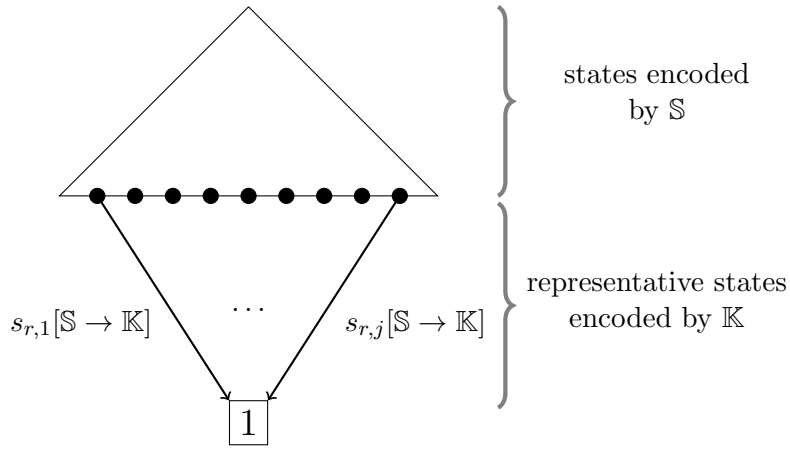


Figure 3.2.: The form of the partition BDD \mathcal{P}^Π for $\Pi = \{B_1, \dots, B_j\}$.

The Signatures

Similarly, the sig^Π functions need to be represented as BDDs. In particular, they have to be easily computable given $trans_{\mathcal{M}}$ and \mathcal{P} . For $sig_{\mathcal{D}}$ and sig_C , this can be done by

simple operations on these MTBDDs:

$$sig_{\mathcal{D}/\mathcal{C}}^{\Pi} = \exists \mathbb{T} (trans_{\mathcal{M}} * \mathcal{P}^{\Pi}[\mathbb{S} \rightarrow \mathbb{T}]).$$

The inner multiplication connects the source-target pairs of $trans_{\mathcal{M}}$ with the blocks of the target states. The existential abstraction then sums up the probabilities over all target states that belong to a single block B for each source state and each $B \in \Pi$, which corresponds to $\mathbf{P}(s, B)$ for $B \in \Pi$.

However, for $sig_{\mathcal{M}}$ the situation is more involved, because a state is mapped to a set of sets. This in turn means that an MTBDD representation of this function would have to provide these sets as leaf nodes. As $trans_{\mathcal{M}}$ and \mathcal{P}^{Π} are of a different form, it is not obvious to us how to easily calculate the signature $sig_{\mathcal{M}}$. Hence, we will follow a slightly different approach: instead of $sig_{\mathcal{M}}$ we will consider the signature

$$sig'_{\mathcal{M}}^{\Pi}(s) = \{(a, p, B) \mid \mathbf{P}_{\mathcal{M}}(s, a, B) = p \text{ and } B \in \Pi\},$$

which can be computed in exactly the same way as $sig_{\mathcal{D}/\mathcal{C}}^{\Pi}$. However, without any other modifications, this will not produce the coarsest bisimulation, because two states agreeing on $sig'_{\mathcal{M}}$ will always have the same probability distributions with respect to the current partition via *exactly the same action labels* whereas $sig_{\mathcal{M}}$ only requires the existence of the same probability distributions in both states. In other words, $sig'_{\mathcal{M}}(s) = sig'_{\mathcal{M}}(s')$ implies $sig_{\mathcal{M}}(s) = sig_{\mathcal{M}}(s')$, but not vice versa, which means that $sig'_{\mathcal{M}}$ imposes a stronger constraint than necessary (and indeed desired). We will modify the refinement algorithm (see section 3.2.2) in a way that accounts for this and compute the coarsest bisimulation despite the simplified signature. The forms of the signature BDDs are sketched in Figure 3.3, where the indicated path illustrates how the MTBDD captures $\mathbf{P}(s, a, B_i) = p$.

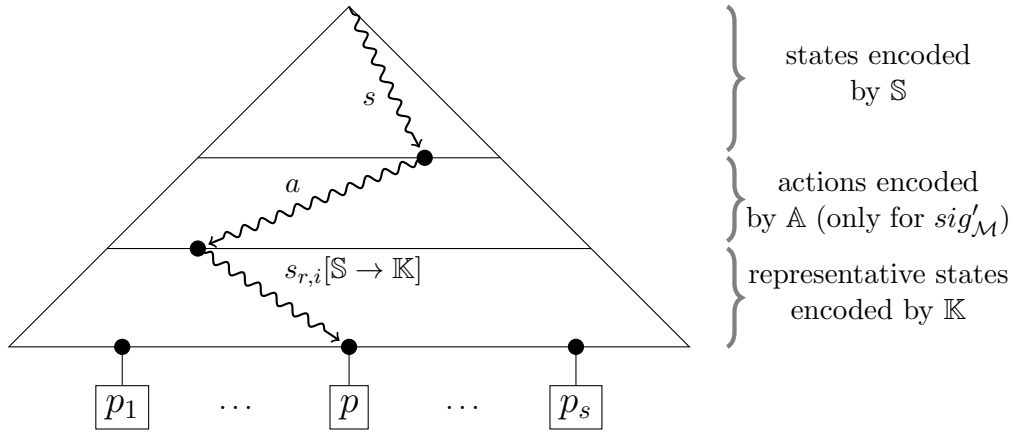


Figure 3.3.: The form of the signature BDDs.

The Refinement Procedure

Ignoring the case of the extended MDP signature for the moment, now that we have computed a BDD sig^Π for the signatures of all states, we need to assign states with different signatures to different blocks. For doing this, SIGREF exploits the fact that CUDD builds *reduced* BDDs that represent identical functions by exactly the same node. Because of the variable order ($\mathbb{S} \prec \mathbb{A} \prec \mathbb{K}$) this means that for two states s, s' we have $sig^\Pi(s) = sig^\Pi(s')$ if and only if the path that is defined by their respective encodings lead to the same node in sig^Π , i.e. $r|_{\mathbb{S}=s} = r|_{\mathbb{S}=s'}$ for the root node r of sig^Π . Algorithm 1 shows SIGREF's algorithmic approach to the refinement procedure.

The algorithm performs a depth-first traversal through the signature MTBDD. If it arrives at a node v that is not labeled with a variable from \mathbb{S} , then either $lab(v) \in \mathbb{K}$ or v is a leaf node of sig^Π . M_v is then substituted by a representative of all the states whose encodings lead to v . For each node that is visited by the traversal, the new sub-BDD is stored in a table. This serves two purposes: first, for non- \mathbb{S} -nodes this ensures that for two states with the same signature, the representative state to which they are mapped will be the same and secondly, for \mathbb{S} -nodes, it avoids unnecessary traversals, because starting from the re-visited node, the traversal would go along the same lines as before. Ultimately, the REFINE procedure transforms sig^{Π^i} into another BDD M that maps states with equal signatures to the same representative, i.e. $M = \mathcal{P}^{\Pi^{i+1}}$. If the number of blocks has increased since the last iteration, i.e. $|\Pi^{i+1}| > |\Pi^i|$, a block has been split in the current iteration and another iteration needs to be triggered. Otherwise, we have reached a fixpoint and conclude the computation.

Algorithm 1 SIGREF's REFINE procedure

```

1: procedure REFINE( $M$ )                                ▷ Refines a block based on its signature BDD  $M$ 
2:    $v \leftarrow root(M)$                                 ▷ Retrieve the top node of the BDD
3:
4:   if  $v \in refineTable$  then
5:     return  $refineTable[v]$                             ▷ Return previously computed sub-BDD
6:   end if
7:
8:   if  $lab(v) \in \mathbb{S}$  then
9:      $l \leftarrow REFINE(v|_{lab(v)=0}), h \leftarrow REFINE(v|_{lab(v)=1})$  ▷ Descend recursively in  $M$ 
10:     $result \leftarrow new\ node(l, h)$                     ▷ Create new node with sub-BDDs  $l$  and  $h$ 
11:   else
12:      $result \leftarrow representativeState()[\mathbb{S} \rightarrow \mathbb{K}]$ 
13:   end if
14:
15:    $refineTable[v] \leftarrow result$ 
16:   return  $result$                                      ▷ Return the resulting node
17: end procedure

```

Modifications for the MDP Signature

The strengthening of the signature $sig_{\mathcal{M}}$ to $sig'_{\mathcal{M}}$ is problematic, because two states might have the same outgoing probability distributions, but via different action labels. However, we make the following observation. Let $r = root(sig_{\mathcal{M}}^{\Pi})$ and $v = r|_{\mathbb{S}=s}$ for some state s , then M_v represents $sig'_{\mathcal{M}}(s)$ and hence, includes \mathbb{A} variables. If we pick $a \in Act_{\mathcal{M}}$ with $a \in Act(s)$, then $w = v|_{\mathbb{A}=a}$ represents the probability distribution that is associated with a . Put differently, it represents the $sig_{\mathcal{D}}(s')$ signature of the state s' resulting from s by omitting all other transitions and only keeping the a -transition. Consequently, $\mathcal{K}_v = \{v|_{\mathbb{A}=a} \mid a \in Act_{\mathcal{M}}\}$, as illustrated by Figure 3.4, is a set of nodes that represents all probability distributions emanating from s .

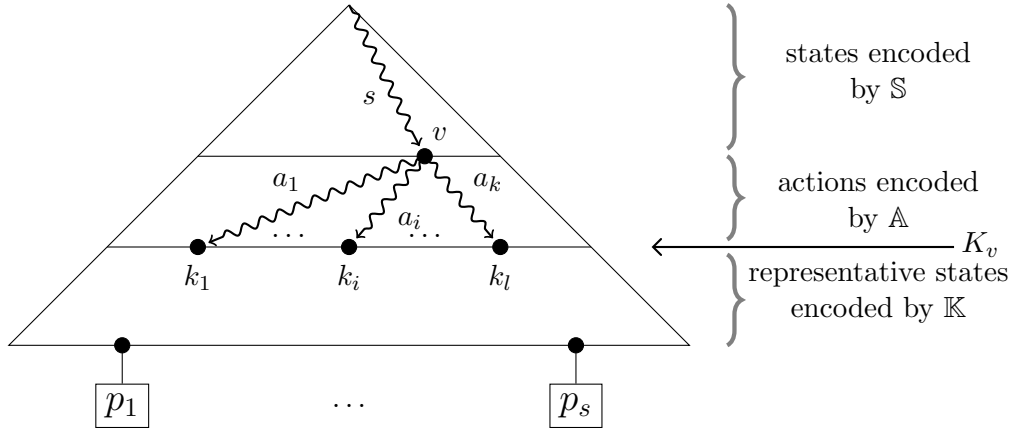


Figure 3.4.: The set \mathcal{K}_v for $v = r|_{\mathbb{S}=s}$.

Rather than assigning two states s, s' to the same block, if $r|_{\mathbb{S}=s} = r|_{\mathbb{S}=s'}$ as SIGREF does by default, we will assign them to the same block if $\mathcal{K}_{r|_{\mathbb{S}=s}} = \mathcal{K}_{r|_{\mathbb{S}=s'}}$. Technically, the set \mathcal{K}_v can be found by performing a depth-first search from v and collecting all nodes w with $lab(w) \in \mathbb{K}$ that are reachable by only passing through \mathbb{A} -labeled nodes before. We formulate this as a modification to the original refinement procedure as illustrated by Algorithm 2. Whenever a non- \mathbb{S} -node v is reached, we compute \mathcal{K}_v and perform a table look-up to see whether a node w with $\mathcal{K}_v = \mathcal{K}_w$ was already encountered. If this is the case, we return the previously chosen representative and substitute v by the corresponding BDD. Otherwise, we pick a presentative for the (new) block and store the mapping of \mathcal{K}_v to the representative for future look-ups.

The correctness of the algorithm can be seen as follows. Let Π^i be the partition of the state space in iteration i and $M = sig'_{\mathcal{M}}^{\Pi^i}$. We observe that states s_1 and s_2 will only be assigned to the same block by REFINE, if either their encodings share a common node (line 4) or for $v = root(M)|_{\mathbb{S}=s_1}$ and $w = root(M)|_{\mathbb{S}=s_2}$ we have $\mathcal{K}_v = \mathcal{K}_w$ (line 13). In the first case, the sub-BDDs representing the quotient distributions are identical and the two states obviously behave identically with respect to Π^i . In the latter case, for each

node v' , $v' \in \mathcal{K}_v$ implies $v' \in \mathcal{K}_w$. As each node in \mathcal{K}_v represents a quotient distribution with respect to the current partition emanating from s_1 , this means that for each $a_1 \in Act(s_1)$ there exists $a_2 \in Act(s_2)$ such that the corresponding quotient distributions with respect to the current partition coincide, i.e. $\mathbf{P}_{\mathcal{M}}(s_1, a_1, \cdot) \equiv_{\sim_{\Pi^i}} \mathbf{P}_{\mathcal{M}}(s_2, a_2, \cdot)$, and consequently, the two states behave equivalently with respect to Π^i .

The algorithm performs a DFS starting from the root node of $M = sig'_{\mathcal{M}}^{\Pi^i}$. For each non- \mathbb{S} -node another nested DFS is started, which collects all reachable \mathbb{K} nodes. As the nested DFSs possibly visit nodes multiple times, an upper bound for the runtime of the REFINE procedure is $\mathcal{O}(|M| \cdot |M(\mathbb{A})|)$ where $|M(\mathbb{A})|$ denotes the number of \mathbb{A} nodes in M . However, at the penalty of using more memory, this can be improved to $\mathcal{O}(|M| + |M(\mathbb{A})|)$ if the reachable \mathbb{K} -nodes for each \mathbb{A} -node are stored in a look-up table as well, preventing them from getting visited multiple times.

Algorithm 2 SIGREF-based refinement procedure for MDPs

```

1: procedure REFINE( $M$ )           ▷ Refines a block based on its signature BDD  $M$ 
2:    $v \leftarrow root(M)$          ▷ Retrieve the top node of the BDD
3:
4:   if  $v \in refineTable$  then
5:     return  $refineTable[v]$      ▷ Return previously computed sub-BDD
6:   end if
7:
8:   if  $lab(v) \in \mathbb{S}$  then
9:      $l \leftarrow REFINE(v|_{v=0})$ ,  $h \leftarrow REFINE(v|_{v=1})$    ▷ Descend recursively in  $M$ 
10:     $result \leftarrow \mathbf{new\ node}(l, h)$    ▷ Create new node with sub-BDDs  $l$  and  $h$ 
11:  else
12:     $dists \leftarrow getReachableKNodes(v)$    ▷ Compute  $\mathcal{K}_v$ 
13:    if  $dists \in distTable$  then
14:      return  $distTable[dists]$            ▷ Look up block representative
15:    else
16:       $result \leftarrow representativeState()[\mathbb{S} \rightarrow \mathbb{K}]$ 
17:       $distTable[dists] \leftarrow result$ 
18:    end if
19:  end if
20:
21:   $refineTable[v] \leftarrow result$ 
22:  return  $result$                  ▷ Return the resulting node
23: end procedure

```

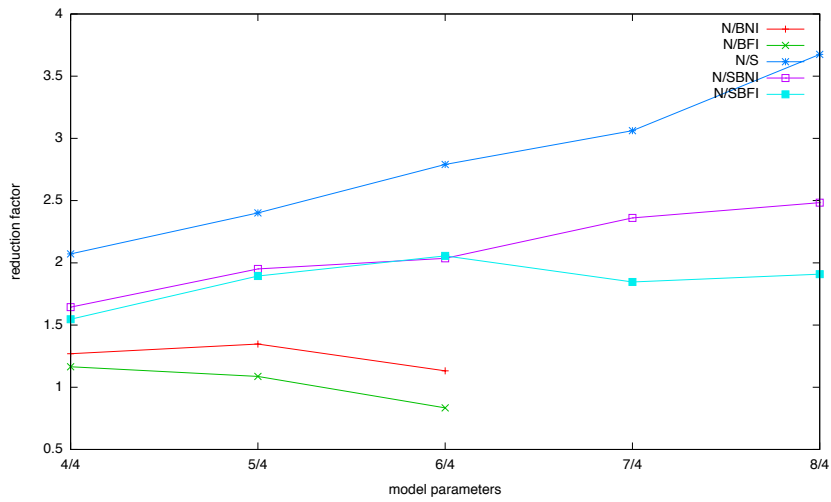
3.3. Case Studies

To study the impact of the previously described method for symbolic bisimulation minimization, we applied it to a variety of case studies, which all can be found on the PRISM website (except for the Crowds protocol, which we remodeled; see section 3.3.7). The experiments were conducted on a blade server with 4 AMD Opteron processors with 12 cores each and 192GB of memory. However, as PRISM and SIGREF both do not use multiple threads, only one core was used. Additionally, we restricted the maximally available memory to 10GB for each experiment. We will now discuss the models that we used and provide diagrams for the most significant data. For the sake of readability, we omitted the properties, probabilistic and reward formulae, we checked as well as all times and sizes here; they can be found in Appendix A. For each case study we include three diagrams: one for the time reduction factor, one for the state reduction factor and one for the BDD size reduction factor. The time reduction diagram illustrates the ratio of the total time, i.e. model construction and verification, needed for the original model (N) to that using bisimulation with $AF(P, \Phi)$ (BNI) and $AF^{dist}(P, \Phi)$ (BFI) as the initial partition, respectively. Consequently, the state reduction diagrams depict the ratios of the sizes of the original system to the sizes of the bisimulation quotient (B) in terms of states, and the BDD size reduction diagrams in terms of nodes of the MTBDD representation. Where applicable, we compared our results with the symmetry reduction (S) feature of PRISM and also evaluated the impact of performing symmetry reduction prior to bisimulation minimization (SBNI and SBFI). A reduction factor of less than one means that the particular time or size actually increased. Note that first, we used a logarithmic scale for some models due to the rapidly increasing reduction factors and secondly, the model size does not necessarily increase from left to right as we ordered the models on the horizontal axes according to the first parameter where applicable. For details we refer to Appendix A.

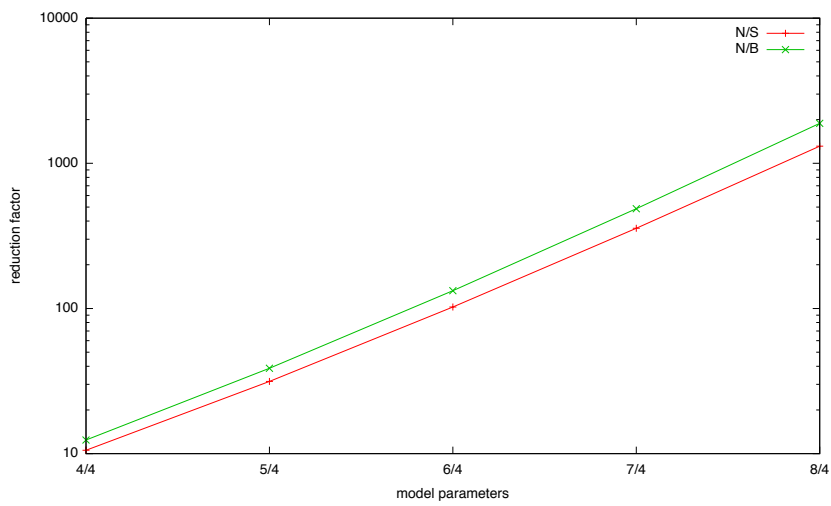
3.3.1. Shared Coin Protocol (MDP)

As a first case study, we used the shared coin protocol of the randomised consensus algorithm of Aspnes and Herlihy [21]. N processes are supposed to express a preference for either 1 or 2. All processes together perform a random walk on a shared global counter starting with the value 0. When the protocol is started, each process throws a coin and increases or decreases the global counter based on the outcome. It may be the case that different processes try to throw the coin simultaneously, which is modeled as a non-deterministic choice in the respective states, so the resulting model is an MDP. After having modified the global counter, the process checks whether the current value of the counter is at most $-N \cdot K$ or at least $N \cdot K$, where $K > 1$ is another parameter of the model, and chooses its preference for either 1 or 2 accordingly. If the observed value is in between the bounds, the process starts over again until a decision has been made.

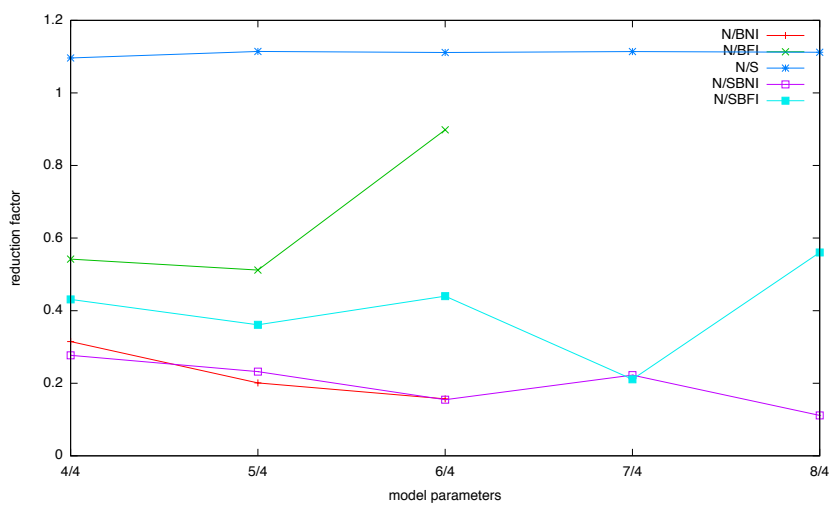
As Figure 3.5 (b) shows, we gain exponentially growing state space reduction factors for both bisimulation minimization and symmetry reduction. Regarding times, BNI



(a) time



(b) states



(c) MTBDD

Figure 3.5.: Shared Coin Protocol results for BDD-based bisimulation.

tends to be slightly faster than N, but for model sizes $N = 7/K = 4$ and $N = 8/K = 4$ BNI and BFI ran out of memory during minimization. SBNI and SBFI perform significantly better than BNI and BFI, because symmetry reduction reduces the sizes of the $trans_{\mathcal{M}}$ MTBDD for this model. Overall, although the state space reduction factor is 20% to 40% greater for bisimulation, it is clearly outperformed by symmetry reduction with respect to BDD sizes and run times. In particular, bisimulation increases MTBDD sizes after symmetry reduction 4 to 10 times, which manifests itself in increased verification times rendering the additional minimization step unreasonable in both time and space requirements.

3.3.2. Firewire (MDP)

Our second case study concerns the IEEE 1394 Tree Identify Protocol, that is designed to elect a leader in a Firewire network [22]. The model is parameterized in the delay DE along the wires that is assumed to be constant. We will consider the shortest and longest possible delay that is still in accordance with the IEEE specification, i.e. $30ns$ and $360ns$. The second parameter is the deadline DL by which we require the protocol to be finished.

In Figure 3.6 (b) we see that we get decreasing BDD reduction factors with increasing model size. This leads to decreasing time reduction factors, which are throughout significantly lower than one, meaning that bisimulation minimization is not worthwhile. At the same time, the state reduction drops from 30 to about 2, suggesting that for bigger models even less redundancy can be eliminated by bisimulation minimization.

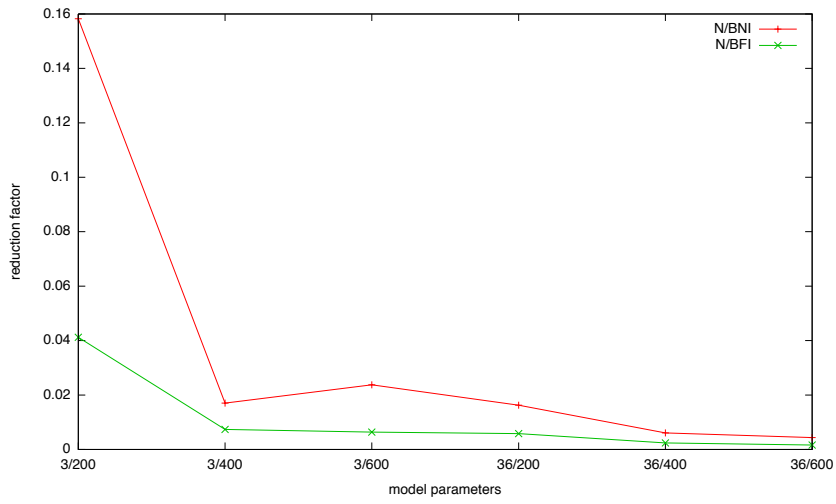
3.3.3. Peer-To-Peer Protocol (CTMC)

The third case study models a simple peer-to-peer protocol of BitTorrent clients [23]. $N + 1$ clients try to download a file that has been split into K parts. In the beginning, one client has all parts of the file in question, whereas all other clients have none. A client may download a part of the file from any of the other clients, who are already in possession of that particular part, but only from a total of 4 different sources.

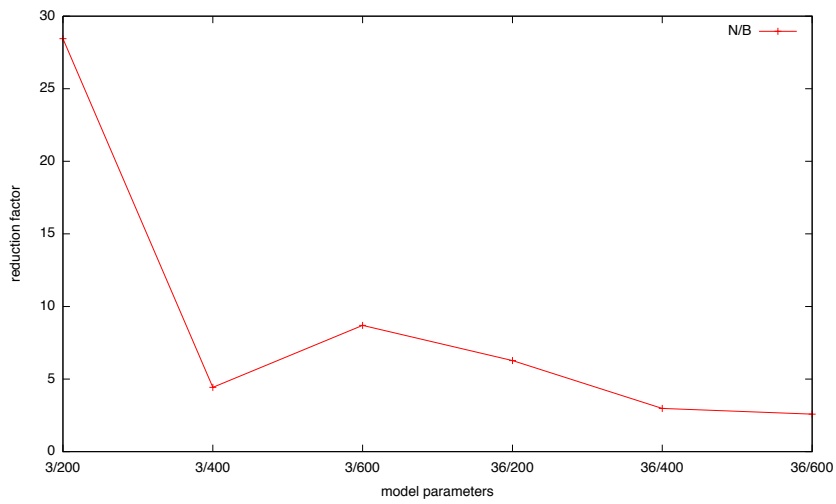
Figure 3.7 shows that for the peer-to-peer protocol huge state space reductions are obtained using bisimulation minimization, 10^3 to 10^5 times more than for symmetry reduction. Interestingly, for this model symmetry reduction does not pay off in terms of time, because it increases the MTBDD size, whereas bisimulation minimization drastically reduces it. Hence, bisimulation outperforms verification of the original model by a factor of 20 to 100. Not surprisingly, applying symmetry reduction prior to bisimulation minimization increases minimization times, because of the aforementioned increase of the BDD size.

3.3.4. Wireless Network (MDP)

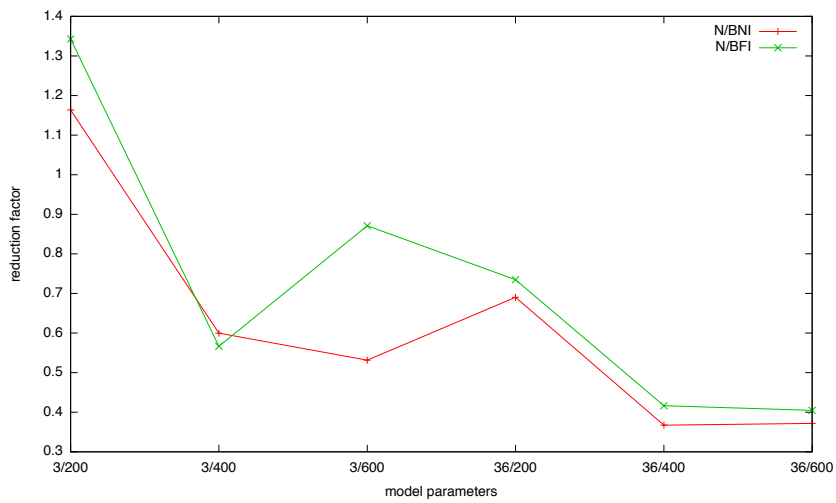
This case study models two wireless stations whose messages collide, i.e. they want to send a message at the same time. In this case, the IEEE 802.11 standard defines a Car-



(a) time



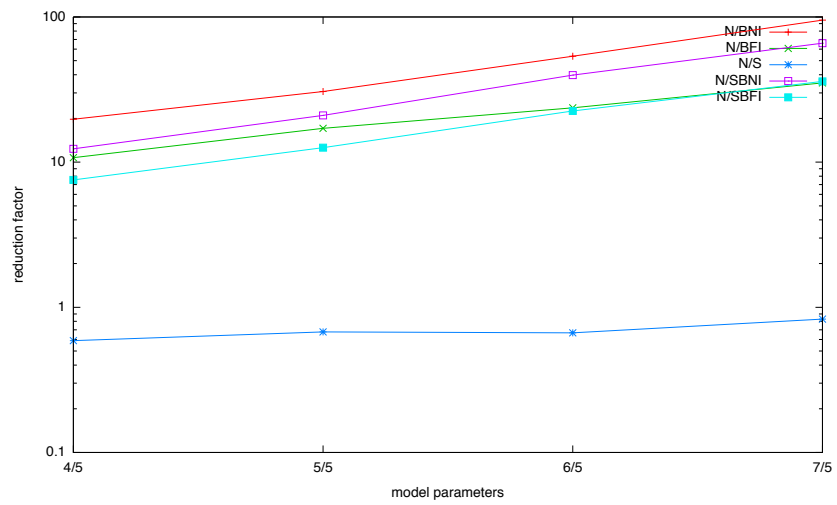
(b) states



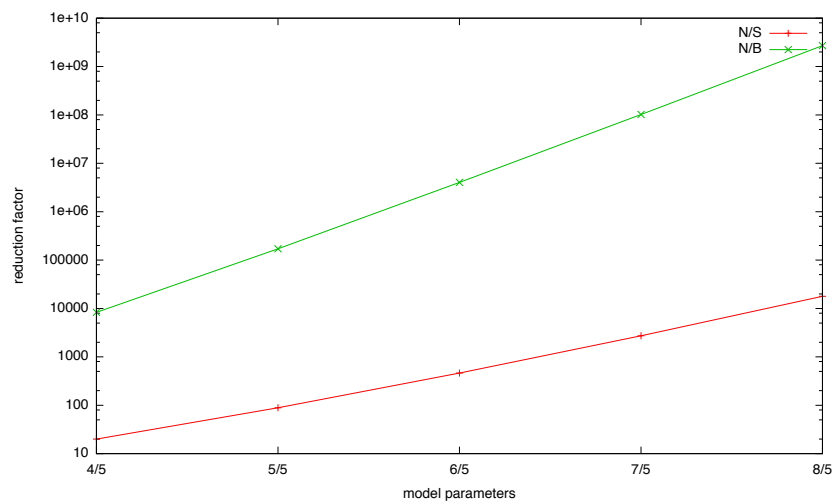
(c) MTBDD

Figure 3.6.: Firewire results for BDD-based bisimulation.

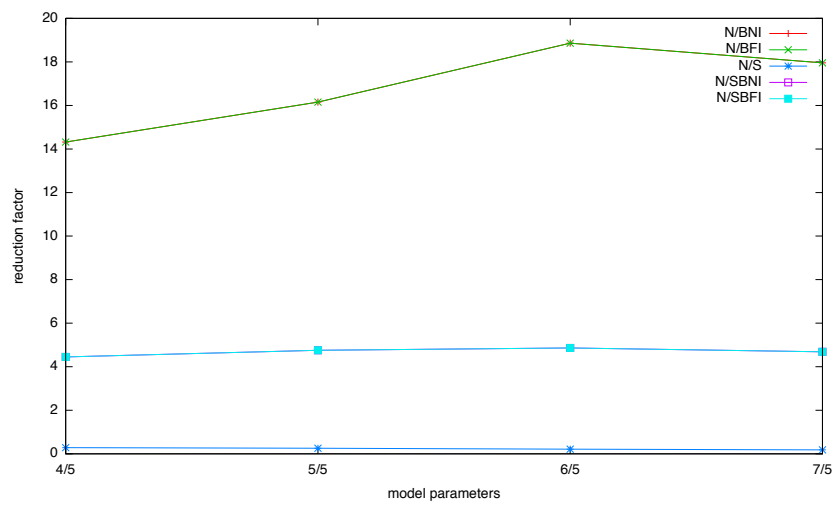
3. BDD-based bisimulation minimization



(a) time



(b) states



(c) MTBDD

Figure 3.7.: Peer-To-Peer Protocol results for BDD-based bisimulation.

rier Sense Multiple Access Collision Avoidance (CSMA/CA) mechanism that seeks to reduce the likelihood of another collision [24]. This mechanism is driven by a randomized exponential backup rule, whose timing depends on the parameter N . The second parameter, C , specifies the maximal number of collisions that can occur.

As Figure 3.8 indicates, for this case study bisimulation minimization yields a relatively constant space reduction factor for models with fixed N and varying C . An increase of N apparently introduces more redundancy, which is lumped away by bisimulation. Nevertheless, BNI and BFI result in an increase of the total time needed, the best case being that minimization overhead and verification gain roughly cancel each other out ($N = 4/C = 8$).

3.3.5. Synchronous Leader Election Protocol (DTMC)

Our fifth case study concerns the synchronous leader election protocol of Itai and Rodeh [25]. A ring of N processors is to elect a leader among themselves by sending messages around the ring. Each of these processors starts with choosing an initial *id* randomly from the set $\{1, \dots, K\}$ which is then passed along the ring by all processors. If there is a unique highest *id* after one complete round, the corresponding processor is determined as the leader, and otherwise another round is started. The timing is controlled by a global clock: on each tick a processor reads the value passed by its predecessor in the ring, decides which value to pass on and writes that value to its output.

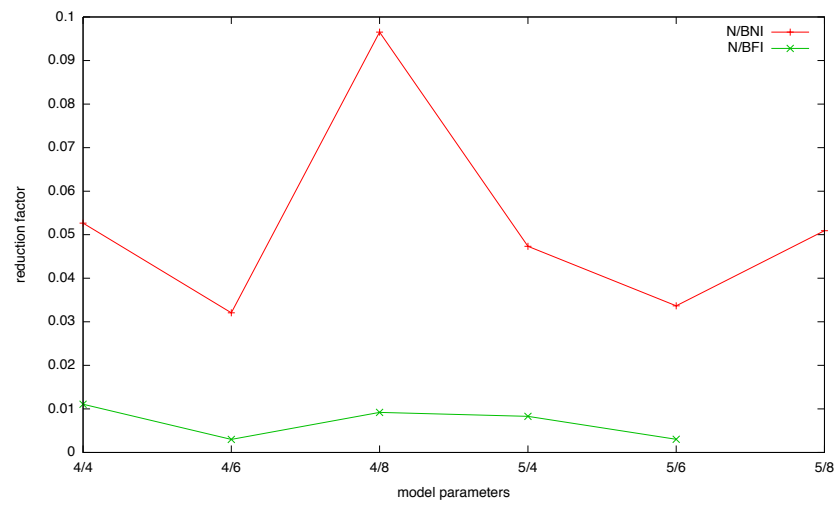
Figure 3.9 illustrates that for the synchronous leader election protocol bisimulation minimization results in huge space reduction up to a factor of 10^5 . For fixed N and varying K we observe that the savings in state space size become bigger with increasing K . Because of this, bisimulation performs better for $K = 8$. But as for the WLAN case study, the time spent on minimization is not outweighed by the reduced verification time, except for $N = 6/K = 8$. All curves indicate that for greater model sizes, the space and BDD reduction grows exponentially and, in particular, bisimulation minimization could become worthwhile with regard to the total time needed. Unfortunately, we were not able to build the system for $N = 7/K = 8$ as PRISM ran out of memory in the process.

3.3.6. Zeroconf Protocol (MDP)

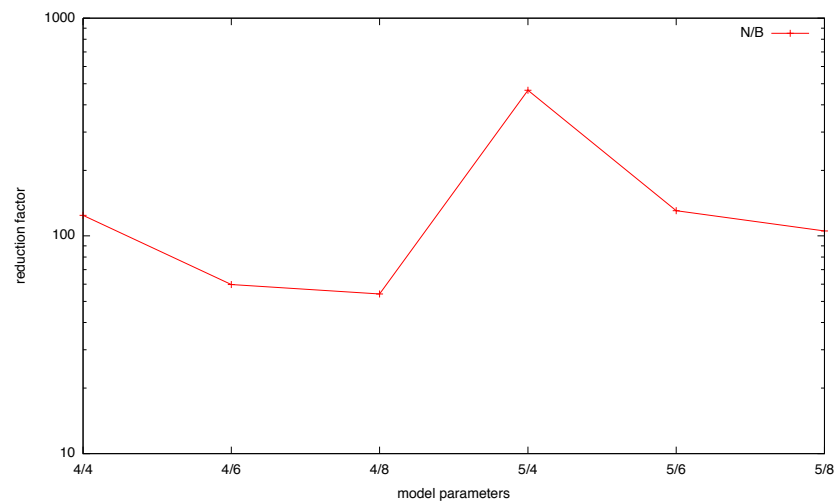
The second to last case study we applied our minimization technique to, is the Zeroconf protocol [26]. It solves the problem of automatically configuring IPv4 addresses in a network. Whenever a new host is added to the network, it randomly chooses one from a fixed pool of K different IP addresses. It then sends N messages, asking the network whether there is a device that already claimed this address. If this is not the case, it will claim this address and otherwise it is required to reconfigure.

Once again we obtain exponentially growing state space reductions using bisimulation minimization (see Figure 3.10), which also results in a reduction of the BDD size for this model. Nevertheless, the time spent on minimization outweighs any reduction of

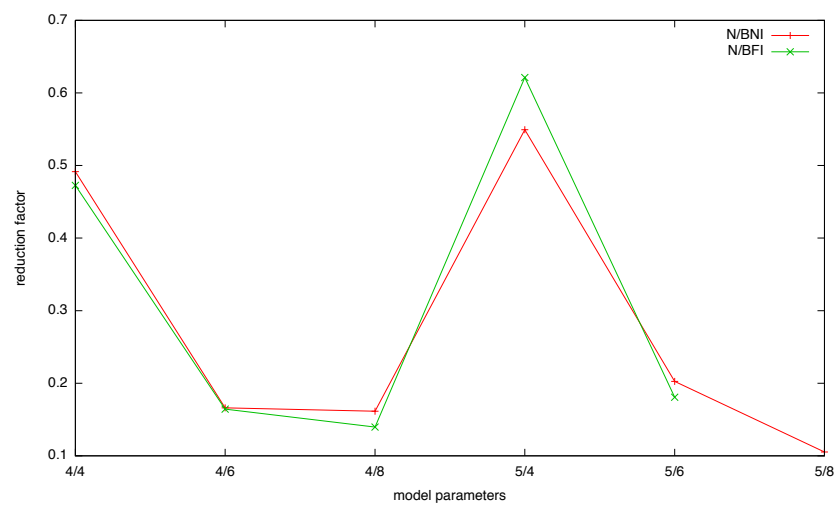
3. BDD-based bisimulation minimization



(a) time

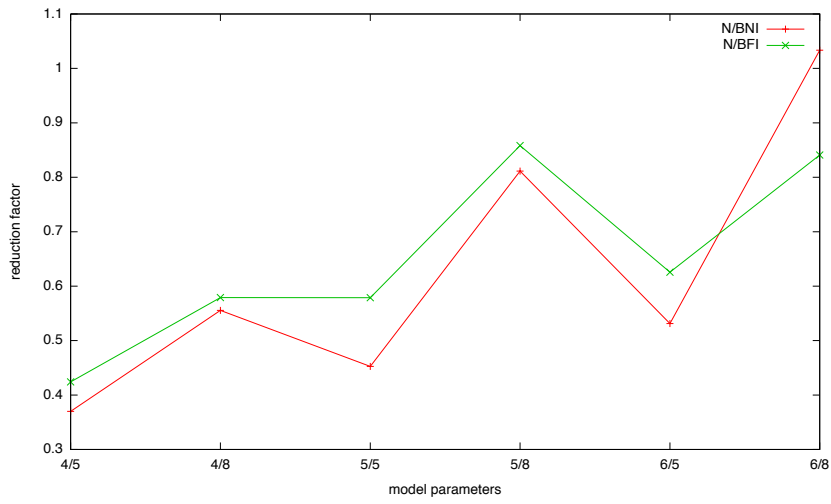


(b) states

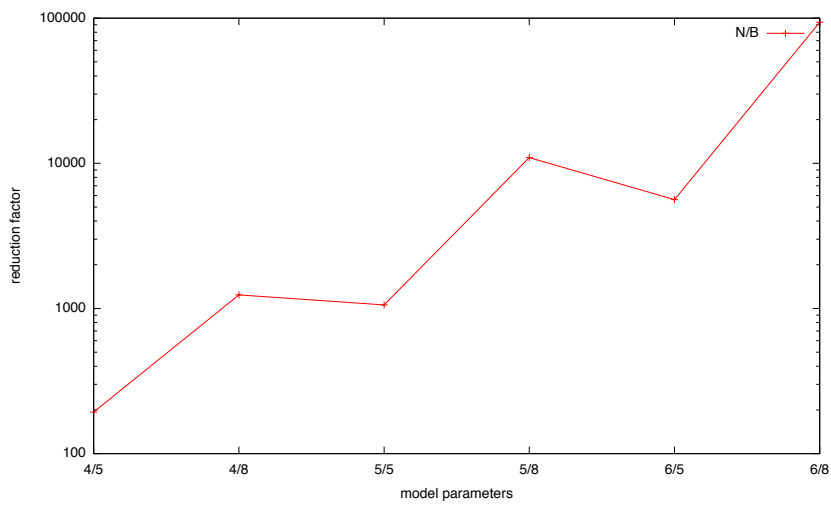


(c) MTBDD

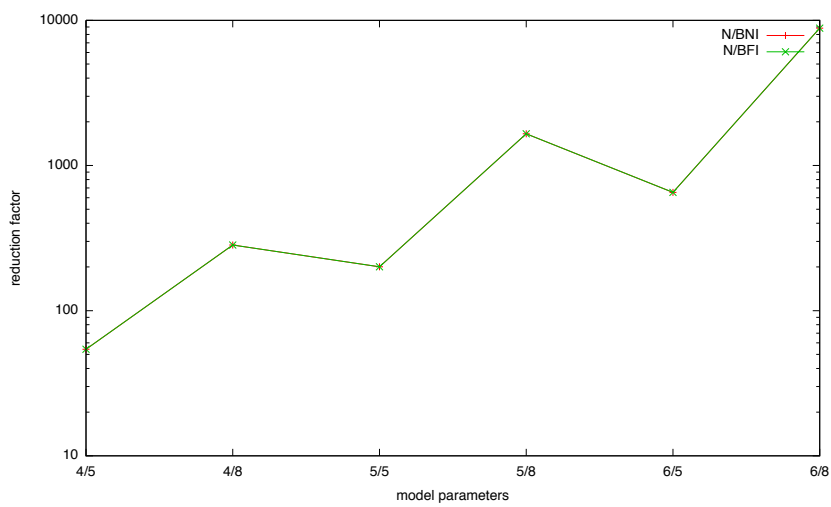
Figure 3.8.: WLAN results for BDD-based bisimulation.



(a) time



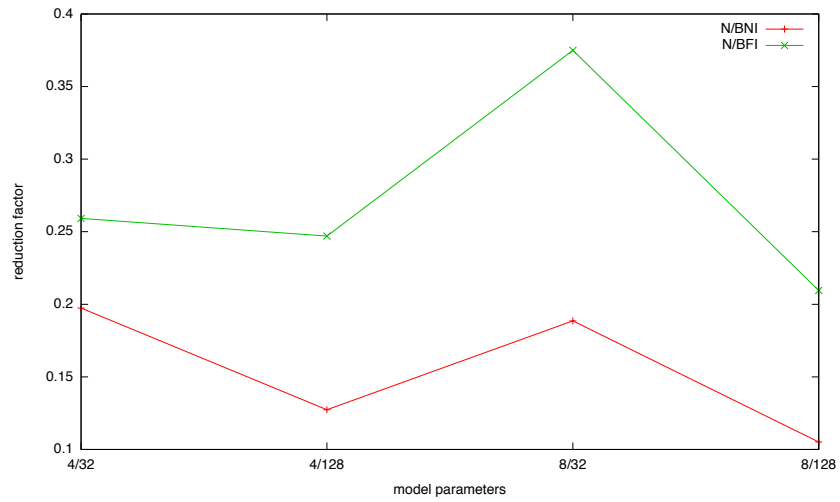
(b) states



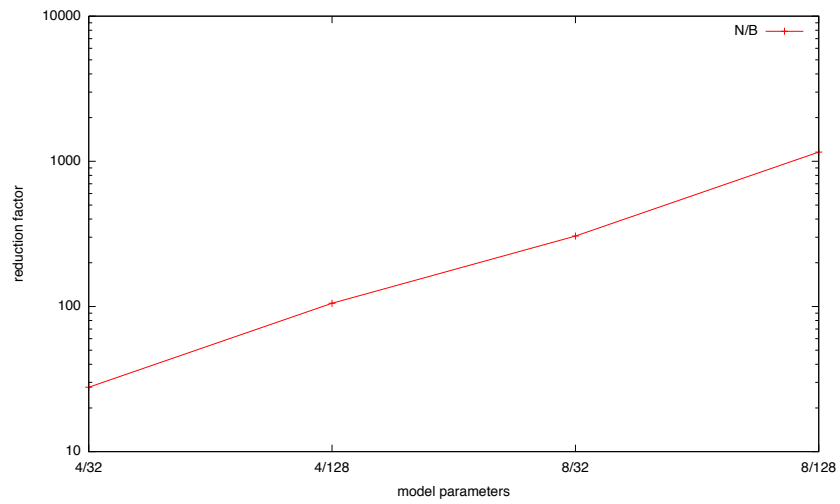
(c) MTBDD

Figure 3.9.: Synchronous Leader Election Protocol results for BDD-based bisimulation.

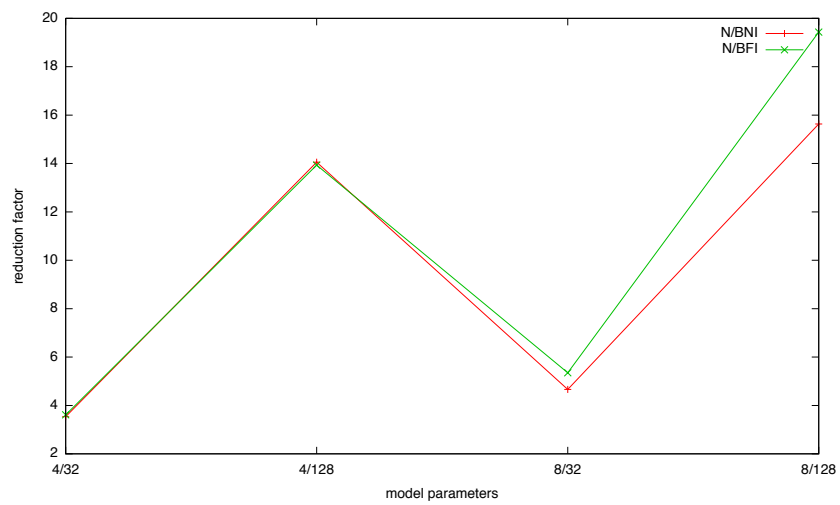
3. BDD-based bisimulation minimization



(a) time



(b) states



(c) MTBDD

Figure 3.10.: Zeroconf Protocol results for BDD-based bisimulation.

verification time that is gained, as shown by the time reduction factors being significantly lower than one.

3.3.7. Crowds Protocol (DTMC)

The Crowds protocol by Reiter and Rubin [27] is a technique for sending a message anonymously to its destination by randomly routing it through a crowd of size N whose members are attackers with a probability bad . A protocol run starts with the sender forwarding the message into the crowd. All good crowd members either send the message to another crowd member with a given probability or deliver the message to its destination, whereas bad crowd members record the most recently seen crowd member, suspecting that this might have been the original sender, and deliver the message directly. The protocol is repeated a total of R times.

Note that although the model is available from the PRISM website, we used a reformulation (see Appendix C) of the model for the sake of comparability with the method devised in chapter 4.

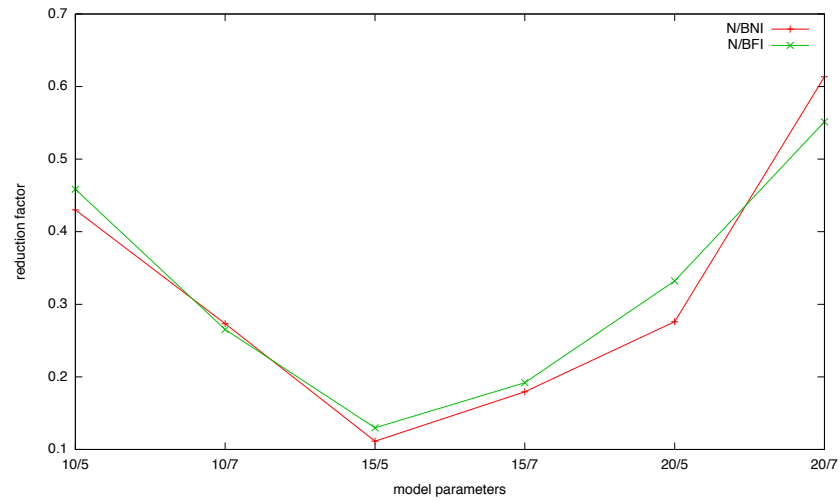
Although the huge state space reduction is reflected in the BDD sizes (see Figure 3.11) and thus in verification times, the time spent on minimizing the system outweighs the verification gain. However, with increasing model size the time reduction factor increases, suggesting that for greater model sizes, bisimulation minimization could pay off in terms of time. But, as for the synchronous leader election protocol (see section 3.3.5), we were not able to build bigger systems without relaxing the memory restriction.

3.4. Analysis

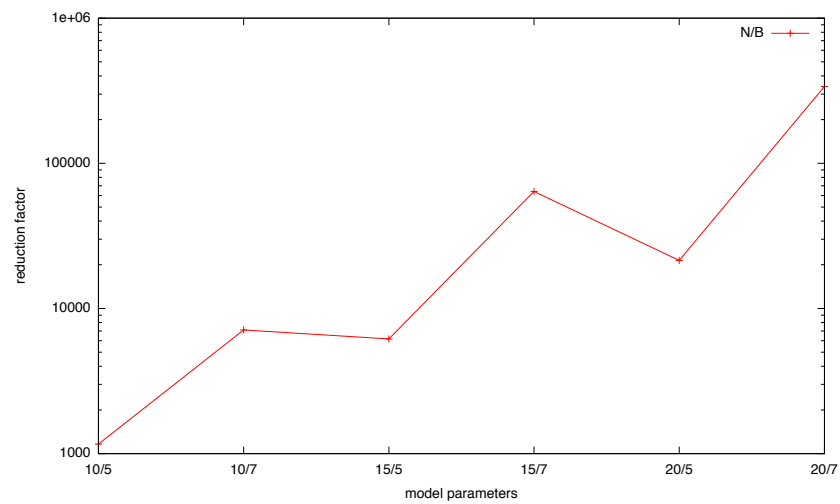
3.4.1. Bisimulation in General

The experiments show that significant state space reductions can be obtained using bisimulation minimization. In 5 out of 7 of the presented case studies even logarithmic state space savings can be observed. This reduction is, however, only partially reflected in the BDD size needed to represent the system. In some cases, the BDD size is indeed reduced, even exponentially, but other examples show that also the reverse is possible. For example, for the shared coin protocol, we get a significant blow-up of the BDD size despite the enormous state space reduction. This is because a reduction of the state space may add irregularity to the system, which negatively influences the numbers of nodes needed in the BDD. This confirms observations. Not surprisingly, in almost all case studies for which this BDD blow-up can be observed, applying bisimulation minimization results not only in an increase of the memory requirements for storing the model but also of verification time compared to the original model. The same holds true for symmetry reduction, which can be seen, e.g., in the peer-to-peer protocol case study. For the cases where the BDD size is actually reduced, we notice reduced verification times. But this reduction comes at a high price: in most cases, the time needed to compute the bisimulation quotient greatly exceeds the saving in verification time, leading to a significantly higher amount of time needed in total. Only for the

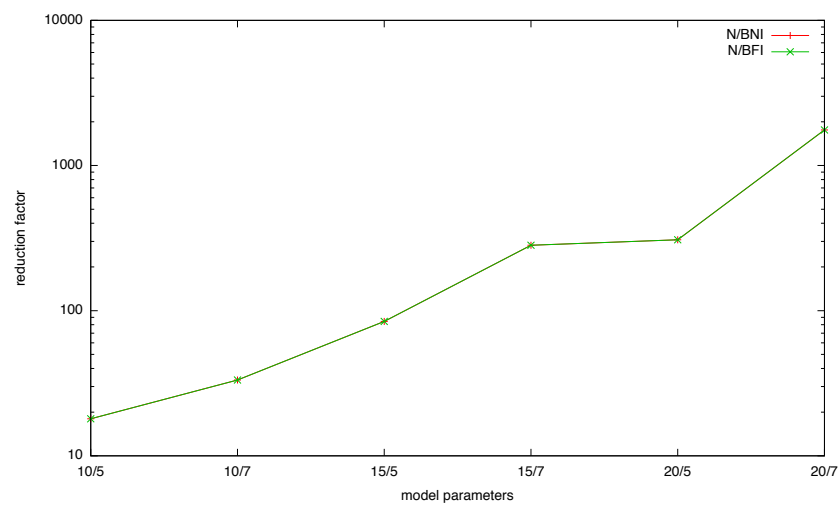
3. BDD-based bisimulation minimization



(a) time



(b) states



(c) MTBDD

Figure 3.11.: Crowds Protocol results for BDD-based bisimulation.

peer-to-peer protocol case study we could gain any notable total time savings. Also, even if the BDD representing the quotient system is smaller than the original BDD, peak memory usage is usually not reduced, as the full system needs to be build first and additionally, the intermediate partition and signature BDDs used during bisimulation computation tend to be bigger than the original system. So all in all, bisimulation minimization seems to be not particularly worthwhile in this setting, neither with respect to time nor with respect to memory requirements.

3.4.2. BNI vs. BFI

Using $AF^{dist}(P, \Phi)$ instead of $AF(P, \Phi)$ significantly reduces the number of iterations needed for most of the models. Unexpectedly, the time spent on bisimulation computation did not decrease, but actually worsened. This is because of two reasons: first, the sets of states with identical minimal distance to "target states" tends to have a bigger BDD representation than the naive initial partition because of irregularity and secondly, in each iteration, SIGREF loops through the initial partition BDDs and refines them instead of the full partition BDD to reduce peak memory usage. In both scenarios, the number of nodes the partition algorithm has to traverse is increased. This is only made up in case the number of iterations is sufficiently reduced by the finer initial partition, for example in the Zeroconf protocol case study.

3.4.3. Bisimulation and Symmetry Reduction

While both reduction techniques produce a bisimulation quotient - symmetry reduction a finer one and bisimulation the coarsest one preserving the respective properties - they differ in some other key aspects. Symmetry reduction is only applicable if certain components of the system are perfectly symmetric which needs to be detected by hand. This is contrasted by bisimulation minimization, which is a fully automated approach and applicable to any of the aforementioned Markov models. Another key difference is the computational approach: while the bisimulation quotient is computed using the partition refinement approach we described earlier, symmetry reduction merges equivalent states via a sorting algorithm applied to the MTBDD parts of the symmetric components of the system. Experiments show that the symmetry reduction quotient is extremely efficient to compute as opposed to the F -bisimulation quotient. Both suffer from the potential BDD blow-up due to reduced regularity in the system. This is nicely illustrated by the shared coin protocol and the peer-to-peer protocol case study. In the first one symmetry reduction outperforms bisimulation, whereas in the latter one this relationship is exactly reversed. Both times it is due to the BDD size of the reduced system. Another interesting observation is that for both case studies, although BDD sizes differ significantly, symmetry reduction computes a bisimulation quotient that is relatively close to exact bisimulation for the shared coin protocol. Regarding the interplay of the two minimization techniques, we note that symmetry reduction is only beneficial for the bisimulation computation if it reduces the BDD size. We did not encounter any case studies for which the combination of the two was more efficient than the application

of only a single minimization step in terms of time. However, in case of the shared coin protocol, symmetry reduction sufficiently reduces the BDD sizes and thus the memory requirements to enable a previously infeasible bisimulation minimization for the larger parameters.

3.4.4. Symbolic vs. Explicit

Katoen et al. [4] investigate the impact of bisimulation minimization on explicit-state model checking. They conclude that for several case studies, including the peer-to-peer (see section 3.3.3) and the crowds protocol (see section 3.3.7), bisimulation minimization positively influences the time needed for the verification of the system. Although the time reduction depends on multiple factors, such as the convergence rate of numerical computations, the huge state space reductions result in smaller data structures, which accelerates all subsequent operations. This contrasts our setting in which decreasing the state space size is not always accompanied by a smaller representation of the system in memory.

Altogether, we conducted three case studies that also appear in [4]. Interestingly, for the synchronous leader election protocol and the peer-to-peer protocol, we get very similar time reduction factors compared to the explicit setting, whereas the relatively high time gain for the crowds protocol in the explicit setting did not occur in our setting.

However, probably the most important observation is the following. Despite the fact that bisimulation minimization in the explicit case often decreases time requirements while in the BDD case it increases computation time, bisimulation computation on BDDs tends to be much faster than on an explicit-state representation. This can be seen by comparing the times needed for bisimulation minimization on all three common case studies. So the reason for the overall observations is not the time requirements for the BDD-based bisimulation minimization per se, but rather the extremely efficient model checking procedures on MTBDDs.

3.5. Conclusion

Confirmed by a number of case studies, we showed that bisimulation minimization yields enormous state space reductions for a variety of models. However, this reduction sometimes leads to a blow-up of the BDD, which in turn leads to both higher memory consumption and an increase in verification time. This blow-up is the result of the unpredictability of BDD sizes in general and a loss of regularity in the system in particular. For the examples for which we get a reduction of the data structure size, bisimulation minimization is not worthwhile for two reasons: first, the peak memory consumption is typically not reduced, because the full system BDD needs to be build upfront and intermediate BDDs during minimization tend to be larger than the original system itself and secondly, the time savings gained are almost always drastically outweighed by the time spent on minimization. Additionally, a finer initial partition reduces the number of iterations needed, but in essence led to even longer minimization times because of

larger intermediate BDDs.

4. SMT-based bisimulation minimization

Ignoring the trees to see the forest doesn't mean that one is more important than the other; it just gives a different perspective.

M. Sipser - *Introduction to the Theory of Computation*

In this chapter, we present a novel technique to extract the bisimulation quotient from a probabilistic program without building the full system model first by leveraging SMT solvers. We describe our prototypical implementation and present first case study results.

The approach described in the previous chapter computes the bisimulation quotient symbolically by BDD manipulations. This, however, has one serious drawback: it requires the data structure representation of the full system to be built prior to minimization although all necessary information about the behavior of the system is already included in the probabilistic program. But this means that if there is a means to reason at the language level, it would be possible to avoid constructing the full model in terms of a symbolic or explicit data structure before minimizing it, but instead directly compute a bisimulation quotient. As a probabilistic program mainly consists of variables and expressions over them, satisfiability checking is an obvious candidate for the aforementioned reasoning gap.

The Boolean satisfiability problem (SAT) is one of the most famous problems in computer science. Given a Boolean formula in conjunctive normal form, it is to be determined whether there exists a valuation (or interpretation) of the Boolean variables, such that the formula evaluates to true, i.e. is *satisfiable*. Satisfiability Modulo Theories (SMT) is an extension of this problem, that allows for formulating satisfiability queries of formulae using more expressive theories, such as, among others, real and linear integer arithmetic (RA and LIA, respectively). Interest in leveraging SAT and SMT formulations in formal verification has grown rapidly, although there remain challenges [17]. It is widely applied, for example in bounded model checking [18, 19], abstraction techniques [5] as well as other verification methods [20].

We will now first define the problem instance we are going to utilize in the SMT-based approach, namely an SMT instance using quantifier-free linear integer arithmetic (QF-LIA), before we present the actual method.

Definition 4.1 (SMT problem instance for QF LIA). Let φ be a quantifier-free formula in the first-order logic of linear arithmetic over integers ($\mathcal{LA}(\mathbb{Z})$). Then

$$SMT_{LIA}(\varphi) = \begin{cases} \text{yes} & \text{if } \exists x_1 \dots \exists x_n \varphi \text{ satisfiable} \\ \text{no} & \text{otherwise.} \end{cases}$$

□

In the sequel, let $P = (ModelType, Var, VarType, b_{s_{init}}, Act, Comm, Rew)$ be a probabilistic program with $ModelType = dtmc$,

$$c = [a] g \longrightarrow p_1 : Var' = E_1 + \dots + p_n : Var' = E_n$$

be a command $c \in Comm$ and $\mathcal{D}_r = (\mathcal{D}, r)$ be the DMRM induced by P with DTMC $\mathcal{D} = (\Sigma(Var), \mathbf{P}_{\mathcal{D}}, s_{init}, AP_{Var}, L)$ such that there are no deadlock states in \mathcal{D} , i.e. for each $s \in \Sigma(Var)$ there exists a command $c' \in Comm$ such that $s \models g_{c'}$. Although we will present the algorithm for DTMCs, it is also applicable to CTMCs. However, its extension to MDPs appears to be non-trivial and is left unconsidered here.

To illustrate the main observations and ideas, we will return to the examples 2.25 and 2.27 throughout this chapter. As shortcuts, we identify a command c with its (unique) label a_c and refer to the i -th update of a command c as $E_{c,i}$. Also recall that g_c denotes the guard of c . Additionally, we use previously introduced denotations such as Var_{E_x} .

4.1. Meaning of Weakest Preconditions

As our algorithm is based on the use of weakest preconditions, it is important to have an intuitive understanding of what they express. Recall that $b(s)$ is a Boolean expression that characterizes the state $s \in \Sigma(Var)$. Given a state $s' \in \Sigma(Var)$ and an assignment E over Var , $WP(b(s'), E) \equiv b(s)$ if and only if $s \xrightarrow{E} s'$. In other words, the weakest precondition $WP(b(s'), E)$ characterizes exactly the state s that is transformed into s' by E . Given an arbitrary expression $b \in BExpr_{Var}$ that characterizes the set of states $\llbracket b \rrbracket$, $\llbracket WP(b, E) \rrbracket = \{s \mid s \xrightarrow{E} s' \text{ for some } s' \in \llbracket b \rrbracket\}$. Put differently, the weakest precondition of a Boolean expression characterizes those states that are transformed into a state in $\llbracket b \rrbracket$ by E .

Example 4.2. Consider the coin flip command

$$[coin] c = 1 \longrightarrow 0.5 : c' = c + 1 \wedge h' = \neg h + 0.5 : c' = c + 1 \wedge h' = h;$$

of the probabilistic program P_{E_x} in example 2.25 and in particular its first assignment

$$E_{coin,1}(v) = \begin{cases} c + 1 & \text{if } v = c \\ \neg h & \text{if } v = h \\ v & \text{otherwise.} \end{cases}$$

Let $b = g_{process} = (c = 2)$ be the guard of $c_{process}$, then

$$WP(b, E_{coin,1}) = b[Var'_{Ex}/E_{coin,1}] = c + 1 = 2,$$

which reflects the fact that every state s with $s \xrightarrow{E_{coin,1}} s'$ for some state $s' \in \llbracket b \rrbracket$ must satisfy $c = 1$. \square

4.2. Idea

Recall that for a probabilistic program over Var , $b(Var)$ is a Boolean expression that ensures the bounds for all variables. Assume that we are given a set of Boolean expressions $\pi = \{b_1, \dots, b_k\}$ that induce a partition

$$\Pi = \{B_1 = \llbracket b_1 \rrbracket, \dots, B_k = \llbracket b_k \rrbracket\}$$

of $\Sigma(Var)$ for which we write $\Pi = \llbracket \pi \rrbracket$. For

$$c = [a] g \longrightarrow p_1 : Var' = E_1 + \dots + p_n : Var' = E_n,$$

and $1 \leq i_1, \dots, i_n \leq k$ we let

$$WP_c(B_{i_1}, \dots, B_{i_n}) = \bigwedge_{j=1}^n WP(b_{i_j}, E_j)$$

be the weakest precondition of B_{i_1}, \dots, B_{i_n} with respect to $c \in Comm$. We can now conclude that

$$\llbracket WP_c(B_{i_1}, \dots, B_{i_n}) \rrbracket = \{s \in \Sigma(Var) \mid s \xrightarrow{E_j} s_j \text{ for some } s_j \in B_{i_j} \text{ for all } 1 \leq j \leq n\}.$$

In other words, the expression characterizes exactly those states that are transformed into an $s_j \in B_{i_j}$ by the j -th assignment of c for all $1 \leq j \leq n$. If additionally c is enabled in $s \in \llbracket WP_c(B_{i_1}, \dots, B_{i_n}) \rrbracket$, i.e. $s \models g_c$, we say that c leads these states into the block combination $(B_{i_1}, \dots, B_{i_n})$ and write $s \xrightarrow{c} (B_{i_1}, \dots, B_{i_n})$. If $s \xrightarrow{c} (B_{i_1}, \dots, B_{i_n})$, we let

$$p_s(j, B) = \begin{cases} p_j & \text{if } B_{i_j} = B \\ 0 & \text{otherwise} \end{cases}$$

and denote the total probability of entering the block B from s with command c by

$$Pr_c(s, B) = \sum_{j=1}^n p_s(j, B).$$

If $s \xrightarrow{c} (B_{i_1}, \dots, B_{i_n})$, the quotient distribution with respect to the current partition in s coincides with $Pr_c(s, \cdot)$, i.e. $[\mathbf{P}_{\mathcal{D}}(s, \cdot)]_{\sim \Pi} = Pr_c(s, \cdot)$, where $\sim \Pi$ is the equivalence relation induced by Π . If additionally $s' \xrightarrow{c} (B_{i_1}, \dots, B_{i_n})$ for another state s' , we

can conclude $Pr_c(s, \cdot) = Pr_c(s', \cdot)$, which in turn implies $\mathbf{P}_{\mathcal{D}}(s, \cdot) \equiv_{\sim_{\Pi}} \mathbf{P}_{\mathcal{D}}(s', \cdot)$. Note, however, that the reverse directions of these implications do not hold, because first, a command might lead to different block combinations that induce the same quotient distribution and secondly, there might be states that have different commands enabled, but still agree on the quotient distribution.

Example 4.3. Reconsider the examples 2.25 and 2.27. Let the partition $\Pi = \llbracket \pi_{ex} \rrbracket$ with $\pi_{ex} = \{f, \neg f\}$, $B_1 = \llbracket f \rrbracket$ and $B_2 = \llbracket \neg f \rrbracket$. Furthermore, let $s_1 = \langle 2, 0, 0, 0 \rangle$. Then, the weakest precondition of (B_1, B_2) with respect to the process command $c_{process}$ is given by

$$WP_{c_{process}}(B_1, B_2) = WP(f, E_{process,1}) \wedge WP(\neg f, E_{process,2}) = \neg f \wedge \neg f \equiv \neg f$$

and $s_1 \in \llbracket WP_{c_{process}}(B_1, B_2) \rrbracket$, because $s_1 \models \neg f$. As s_1 also satisfies $g_{process} = (c = 2)$, we conclude $s_1 \xrightarrow{c_{process}} (B_1, B_2)$,

$$Pr_{c_{process}}(s_1, B) = \begin{cases} 0.2 & \text{if } B = B_1 \\ 0.8 & \text{if } B = B_2 \end{cases}$$

and $\llbracket \mathbf{P}_{\mathcal{D}}(s_1, \cdot) \rrbracket_{\sim_{\Pi}} = Pr_{c_{process}}(s_1, \cdot)$.

Also observe that for $s_2 = \langle 3, 1, 0, 0 \rangle$ we have $s_2 \xrightarrow{c_{return^1}} (B_1, B_2)$ and $\mathbf{P}_{\mathcal{D}}(s_2, \cdot) \equiv_{\sim_{\Pi}} \mathbf{P}_{\mathcal{D}}(s_1, \cdot)$, although there exists no $c \in Comm$ such that $Pr_c(s_1, \cdot) = Pr_c(s_2, \cdot)$. \square

Letting $B = \llbracket b \rrbracket \in \Pi$ and

$$\varphi = \underbrace{b}_{s \in B} \wedge \underbrace{g_c \wedge WP_c(B_{i_1}, \dots, B_{i_n})}_{s \xrightarrow{c} (B_{i_1}, \dots, B_{i_n})}$$

we observe that $s \models \varphi$ if and only if $s \in B$ and $s \xrightarrow{c} (B_{i_1}, \dots, B_{i_n})$. In other words, φ characterizes exactly the subset of states of B that is led into $(B_{i_1}, \dots, B_{i_n})$ by c and therefore agrees with the quotient distribution with respect to Π . Recall that $b(Var)$ is a Boolean expression that ensures the bounds for all variables in Var . Then, $\llbracket \varphi \rrbracket$ is non-empty if and only if the QF_LIA formula $\varphi \wedge b(Var)$ is satisfiable, i.e. if $SMT_{LIA}(\varphi \wedge b(Var)) = yes$, which can be determined using an SMT solver. The additional conjunct $b(Var)$ is necessary, because otherwise the solver might find integer solutions that are not valid valuations of variables and are thus not states in $\Sigma(Var)$.

Example 4.4. Let $\Pi = \llbracket \pi'_{Ex} \rrbracket$ with $\pi'_{Ex} = \{c \neq 4, c = 4\}$ be a partition of $\Sigma(Var)$, $B_1 = \llbracket c \neq 4 \rrbracket$ and $B_2 = \llbracket c = 4 \rrbracket$. By constructing

$$\begin{aligned} \varphi_{Ex} &= \underbrace{c \neq 4}_{s \in B_1} \wedge \underbrace{c = 3 \wedge f \wedge WP_{c_{restart}}(B_1, B_2)}_{s \xrightarrow{c_{restart}} (B_1, B_2)} \\ &= c \neq 4 \wedge c = 3 \wedge f \wedge 1 \neq 4 \wedge c + 1 = 4 \\ &\equiv c = 3 \wedge f \end{aligned}$$

and checking $\varphi_{Ex} \wedge b(\text{Var}_{Ex})$ for satisfiability, we determine that there are states $s \in B_1$ with $s \xrightarrow{c_{restart}} (B_1, B_2)$ and, as expected, we observe $\langle 3, 1, 1, 0 \rangle$ and $\langle 3, 0, 1, 0 \rangle \in \llbracket \varphi_{Ex} \rrbracket$. Accordingly, we have

$$Pr_{c_{reset}}(s, B) = \begin{cases} 0.99 & \text{if } B = B_1 \\ 0.01 & \text{if } B = B_2 \end{cases}$$

for all $s \models \varphi_{Ex}$. □

4.3. Algorithm

We will now present an algorithm that takes a probabilistic program as its input, computes a bisimulation based on the aforementioned idea and produces another probabilistic program whose semantics is the quotient of the original system with respect to the bisimulation.

4.3.1. Computing the Bisimulation

As for the BDD-based approach, the algorithm utilizes the usual partition refinement approach. At all times, the current partition $\Pi = \{B_1, \dots, B_k\}$ is represented by a set of expressions $\pi = \{b_1, \dots, b_k\} \subseteq BExpr_{Var}$ such that $\Pi = \llbracket \pi \rrbracket$. Initially, $\pi_{init} = B(AF(P, \Phi))$ canonically represents the initial partition $\Pi_{init} = \Pi_{AF(P, \Phi)}$.

Algorithm 3 illustrates the refinement loop, which is based on a procedure `SMTREFINE`. It refines the block $B = \llbracket b \rrbracket$ with respect to the current partition $\Pi = \llbracket \pi \rrbracket$ and returns expressions characterizing the sub-blocks into which B was split.

Algorithm 3 SMT-based partition refinement

Require: probabilistic program P , formula Φ

Ensure: $\Pi = \llbracket \pi \rrbracket$ induces $AF(P, \Phi)$ -Bisimulation

```

1: procedure SMTBISIM( $P, \Phi$ )                                ▷ Computes an  $AF(P, \Phi)$ -bisimulation
2:    $\pi \leftarrow B(AF(P, \Phi))$                                ▷ Set up initial partition
3:    $\pi_{old} \leftarrow \emptyset$ 
4:
5:   while  $|\pi_{old}| \neq |\pi|$  do                               ▷ Refine until stable partition reached
6:      $\pi_{old} \leftarrow \pi$ 
7:     for  $b \in \pi$  do
8:        $\pi = (\pi \setminus b) \cup \text{SMTREFINE}(b, \pi)$            ▷ Refine  $B = \llbracket b \rrbracket$  wrt.  $\llbracket \pi \rrbracket$ 
9:     end for
10:  end while
11:
12:  return  $\pi$ 
13: end procedure

```

SMTREFINE is the core procedure of the algorithm and is sketched in Algorithm 4 on page 63. It makes use of the following capabilities of the SMT solver:

- $assert(b)$ for a $b \in BExpr_{Var}$ adds the formula b to the assertion stack of the solver,
- $removeAssertions()$ removes all currently asserted formulae,
- $hasNextSolution()$ returns true if and only if the conjunction of all formulae on the assertion stack is satisfiable and stores a model, i.e. an assignment of values to variables such that all formulae evaluate to true, if it was found to be satisfiable;
- if the last call to $hasNextSolution()$ returned true and given ordered sets $Z_1 = \{z_{1,1} \dots, z_{1,k_1}\}, \dots, Z_n = \{z_{n,1} \dots, z_{n,k_n}\}$ of Boolean auxiliary variables, $getTrueVarInd(Z_1, \dots, Z_n)$ returns an n -tuple (i_1, \dots, i_n) of indices, such that $z_{1,i_1}, \dots, z_{n,i_n}$ are assigned 1 in the model, i.e. the valuation of variables that makes all asserted formulae evaluate to true, the solver found to prove satisfiability.

The procedure SMTREFINE computes a mapping sig from probability distributions $\mu \in Dist_{\Pi}$ to sets $B_{\mu} = sig[\mu]$ of Boolean expressions such that $s \in B$ with $[\mathbf{P}_{\mathcal{D}}(s, \cdot)]_{\sim_{\Pi}} = \mu$ if and only if there exists $b_{\mu} \in B_{\mu}$ with $s \models b_{\mu}$. If the number of keys of the mapping after the refinement procedure is greater than one, this means the block needs to be split, because there exist states with different quotient distributions. Additionally, we can derive Boolean expressions that characterize the necessary sub-blocks from sig as follows. As there are no deadlock states and Π is a partition of $\Sigma(Var)$, $[\mathbf{P}_{\mathcal{D}}(s, \cdot)]_{\sim_{\Pi}} \in keys(sig)$ for all $s \in B$. But then

$$s \in B \text{ with } [\mathbf{P}_{\mathcal{D}}(s, \cdot)]_{\sim_{\Pi}} = \mu \Leftrightarrow \exists b_{\mu} \in B_{\mu} \text{ such that } s \models b_{\mu} \Leftrightarrow s \models \bigvee_{b_{\mu} \in B_{\mu}} b_{\mu}$$

and the expressions for the sub-blocks into which B needs to be split are given by

$$\left\{ \bigvee_{b_{\mu} \in B_{\mu}} b_{\mu} \mid \mu \in keys(sig) \right\},$$

which is returned by SMTREFINE upon termination.

In order to compute sig appropriately, the key idea is to perform an ALLSAT-like enumeration of the different block-combinations a command c might lead to for each command $c \in Comm$. To that end, we construct the following system of formulae for each command c and auxiliary variables $z_{i,j} \notin Var$

$$b(Var) \wedge b \wedge g_c \tag{4.1}$$

$$z_{i,j} \rightarrow WP(b_j, E_i) \text{ for all } 1 \leq j \leq k \text{ and all } 1 \leq i \leq n \tag{4.2}$$

$$\bigvee_{j=1}^k z_{i,j} \text{ for all } 1 \leq i \leq n \tag{4.3}$$

While the formula (4.1) ensures that the states we are dealing with are in B and have c enabled, (4.2) is a system of formulae that asserts the weakest preconditions of all blocks with respect to all assignments of c , each implied by a unique auxiliary variable $z_{i,j}$. Finally, the formula (4.3) enforces that in each solution at least one auxiliary variable belonging to a weakest precondition with respect to the assignment E_i equals 1 for all $1 \leq i \leq n$. Because of the last requirement, in each model found by the solver there exist indices $1 \leq i_1, \dots, i_n \leq k$ such that $z_{1,i_1}, \dots, z_{n,i_n}$ all equal 1, which inevitably implies $\bigwedge_{j=1}^n WP(b_{i_j}, E_j)$, because of (4.2). That is, in combination with (4.1), we can conclude that there exist $s \in B$ with $s \xrightarrow{c} (B_{i_1}, \dots, B_{i_n})$. Hence, enumerating all solutions of the $z_{i,j}$ for the formulae system and identifying the appropriate indices i_1, \dots, i_n produces all n -tuple of blocks in Π into which some state in B is led by c . Every time a model is found, we first compute the probability distribution that is induced by the block-tuple and update the signature table accordingly, before the current solution is ruled out by asserting $\bigvee_{j=1}^n \neg z_{j,i_j}$ and the solver is used to determine the next solution.

Example 4.5. Let Π , π'_{Ex} , B_1 , B_2 be as defined in Example 4.4 and $B = B_1$ the block to be refined.

We choose c_{coin} as the first command and construct the formulae system:

$$\begin{aligned} b(Var_{Ex}) \wedge c \neq 4 \wedge c = 1 \\ z_{1,1} \rightarrow c + 1 \neq 4 \\ z_{1,2} \rightarrow c + 1 = 4 \\ z_{2,1} \rightarrow c + 1 \neq 4 \\ z_{2,2} \rightarrow c + 1 = 4 \\ z_{1,1} \vee z_{1,2} \\ z_{2,1} \vee z_{2,2} \end{aligned}$$

The solver returns $z_{1,1} = 1$ and $z_{2,1} = 1$ as the only solution of the auxiliary variables, which corresponds to the fact that for all states in B_1 that have c_{coin} enabled the quotient distribution is $\mu_1(B') = 1$ if and only if $B' = B_1$. We update the previously empty mapping sig accordingly

$$sig = \{\mu_1 \mapsto \{c \neq 4 \wedge c = 1 \wedge c + 1 \neq 4 \wedge c + 1 = 4\}\}$$

and continue with constructing the formulae system for $c_{restart}$

$$\begin{aligned} b(Var_{Ex}) \wedge c \neq 4 \wedge c = 3 \wedge f \\ z_{1,1} \rightarrow 1 \neq 4 \\ z_{1,2} \rightarrow 1 = 4 \\ z_{2,1} \rightarrow c + 1 \neq 4 \\ z_{2,2} \rightarrow c + 1 = 4 \\ z_{1,1} \vee z_{1,2} \\ z_{2,1} \vee z_{2,2} \end{aligned}$$

for which the solver identifies $z_{1,1} = 1$ and $z_{2,2} = 1$ as the only solution. This means that all states s with $s \models g_{restart}$ possess the quotient distribution

$$\mu_2(B') = \begin{cases} 0.99 & \text{if } B' = B_1 \\ 0.01 & \text{if } B' = B_2 \end{cases}$$

which updates sig to

$$sig = \{ \mu_1 \mapsto \{c \neq 4 \wedge c = 1 \wedge c + 1 \neq 4 \wedge c + 1 \neq 4\}, \\ \mu_2 \mapsto \{c \neq 4 \wedge c = 3 \wedge f \wedge 1 \neq 4 \wedge c + 1 = 4\} \}.$$

Following this process and omitting the intermediate results, SMTREFINE computes the final mapping sig as

$$sig = \{ \begin{array}{l} \mu_1 \mapsto \{c \neq 4 \wedge c = 1 \wedge c + 1 \neq 4 \wedge c + 1 \neq 4, \\ \quad c \neq 4 \wedge c = 2 \wedge c + 1 \neq 4 \wedge c + 1 \neq 4\}, \\ \mu_2 \mapsto \{c \neq 4 \wedge c = 3 \wedge f \wedge 1 \neq 4 \wedge c + 1 = 4\}, \\ \mu_3 \mapsto \{c \neq 4 \wedge c = 3 \wedge h \wedge \neg f, \\ \quad c \neq 4 \wedge c = 3 \wedge \neg h \wedge \neg f\} \end{array} \},$$

where μ_1 and μ_2 are as before and $\mu_3(B') = 1$ if and only if $B' = B_2$. Consequently, B needs to be split in three different sub-blocks according to the different quotient distributions. \square

4.3.2. Extracting the Quotient

Once a stable partition $\Pi = \llbracket \pi \rrbracket = \llbracket \{b_1, \dots, b_k\} \rrbracket$ has been reached, the quotient system has to be computed. We do this by constructing a probabilistic program whose semantics is the bisimulation quotient of the original system. During each refinement step, we keep track of the quotient probability distribution μ_B associated with each block B . Then, the resulting probabilistic program is given by

$$P_{\sim} = (ModelType, Var_{\sim}, VarType_{\sim}, init_{\sim}, \{a_1, \dots, a_k\}, Comm_{\sim}, Rew_{\sim})$$

where

- $Var_{\sim} = \{ \text{block} \}$,
- $VarType_{\sim}(\text{block}) = int[1, k]$,
- $init_{\sim} = (\text{block} = i)$ if and only if $b_i \wedge b(\text{init})$ is satisfiable,
- $Comm_{\sim} = \{[a_i] \text{block} = i \longrightarrow \sum_{j=1}^k \mu_{B_i}(B_j) : \text{block}' = j \mid 1 \leq i \leq k\}$,
- $Rew_{\sim} = \{(\text{block} = i, r) \mid (b, r) \in Rew \text{ such that } b \wedge b_i \text{ satisfiable}\}$.

Algorithm 4 SMT-based block refinement

Require: block $B = \llbracket b \rrbracket$ given by $b \in BExpr_{Var}$, partition $\Pi = \llbracket \pi \rrbracket$ of $\Sigma(Var)$ given by $\pi = \{b_1, \dots, b_k\} \subseteq BExpr_{Var}$
Ensure: returns partition $\pi_b \subseteq BExpr_{Var}$ of $\llbracket b \rrbracket$ such that $\mathbf{P}_{\mathcal{D}}(s, \cdot) \equiv_{\sim_{\Pi}} \mathbf{P}_{\mathcal{D}}(s', \cdot)$ if and only if $s \models b' \Leftrightarrow s' \models b'$ for all $b' \in \pi_b$

```

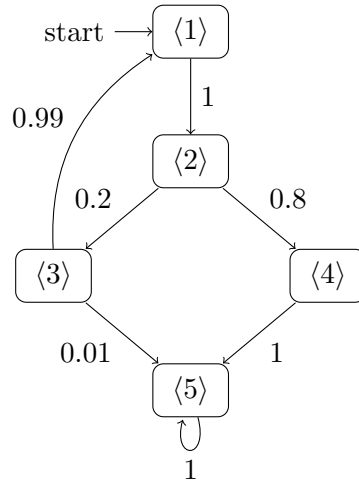
1: procedure SMTREFINE( $b, \pi = \{b_1, \dots, b_k\}$ )  $\triangleright$  Refines  $B = \llbracket b_{var} \wedge b \rrbracket$  wrt.  $\Pi = \llbracket \pi \rrbracket$ 
2:    $sig = \emptyset$   $\triangleright$  Initialize mapping of  $Dist_{\Pi}$  to set of expressions in  $BExpr_{Var}$ 
3:
4:   for each  $c = [a] g \longrightarrow r_1 : Var' = E_1 + \dots + r_n : Var' = E_n \in Comm$  do
5:      $assert(b_{var})$   $\triangleright$  Assert common part to all SMT calls
6:      $assert(b)$ 
7:      $assert(g)$ 
8:
9:     for  $i = 1 \dots n$  do
10:      for  $j = 1 \dots k$  do
11:         $assert(z_{i,j} \rightarrow WP(b_j, E_i))$   $\triangleright$  Assert all possible WPs wrt.  $E_i$ 
12:      end for
13:       $Z_i \leftarrow \{z_{i,m} \mid 1 \leq m \leq k\}$   $\triangleright$  Store auxiliary variable group for  $E_i$ 
14:       $assert(\bigvee_{j=1}^k z_{i,j})$   $\triangleright$  Require at least one WP to hold in each group
15:    end for
16:
17:    while hasNextSolution() do
18:       $(i_1, \dots, i_n) \leftarrow getTrueVarInd(Z_1, \dots, Z_n)$   $\triangleright$  Get active WPs in solution
19:
20:       $b_{sig} \leftarrow b \wedge g \wedge \bigwedge_{j=1}^n WP(b_{i_j}, E_{i_j})$   $\triangleright s \longrightarrow (B_{i_1}, \dots, B_{i_n})$  for all  $s \models b_{sig}$ 
21:       $sig[Pr_{s,c}(\cdot)] \leftarrow sig[Pr_{s,c}(\cdot)] \cup \{b_{sig}\}$   $\triangleright$  Update signature accordingly
22:
23:       $assert(\bigvee_{j=1}^n \neg z_{i,j})$   $\triangleright$  Rule out currently active WP combination
24:    end while
25:
26:     $removeAssertions()$   $\triangleright$  Clear assertion stack of solver
27:  end for
28:
29:  if  $|keys(sig)| > 1$  then
30:    return  $\{\bigvee_{e \in V} e \mid V \in val(sig)\}$   $\triangleright$  Split block if necessary
31:  else
32:    return  $b$   $\triangleright$  Otherwise return input block
33:  end if
34: end procedure

```

```

    block      :   int[1, 5] init 1;

    [a1] block = 1  →  1 : block' = 2;
    [a2] block = 2  →  0.2 : block' = 3 + 0.8 : block' = 4;
    [a3] block = 3  →  0.99 : block' = 1 + 0.01 : block' = 5;
    [a4] block = 4  →  1 : block' = 5;
    [a5] block = 5  →  1 : block' = 5;
    
```

 Figure 4.1.: The probabilistic program $P_{Ex, \sim}$.

 Figure 4.2.: The quotient DTMC $\mathcal{D}_{Ex, \sim} = \llbracket P_{Ex, \sim} \rrbracket$.

Example 4.6. Continuing the previous example, we compute the stable partition (expressions have been simplified to improve readability)

$$\pi_{Ex}^{sta} = \{ \underbrace{c = 1 \wedge \neg f}_{B_1^{sta}}, \underbrace{c = 2 \wedge \neg f}_{B_2^{sta}}, \underbrace{c = 3 \wedge f}_{B_3^{sta}}, \underbrace{c = 3 \wedge \neg f}_{B_4^{sta}}, \underbrace{c = 4}_{B_5^{sta}}, \underbrace{c = 1 \wedge f}_{B_6^{sta}}, \underbrace{c = 2 \wedge f}_{B_7^{sta}} \}$$

and extract the quotient program depicted in Figure 4.1, in which we already omitted the unreachable blocks B_6^{sta} and B_7^{sta} . The corresponding quotient DTMC is presented in Figure 4.2. \square

4.4. Prototypical Implementation

4.4.1. General

There is a number of SMT solvers that regularly participate in competitions and are particularly successful in the QF_LIA class, e.g. Yices¹, MathSat² and SatEEn³. In our setting, it turned out to be of extreme importance that the solver provides a C/C++ API instead of just an executable which is to be used with standard file input, as this significantly shortens computation times. Against this background, Yices and MathSat 4 are particularly interesting. With MathSat being faster in the competitions, we implemented a prototype of our algorithm in about 10,000 lines of Java code and connected it to the C API of MathSat via the Java Native Interface (JNI).

Choosing Java allowed reusing the expression and parsing engine of PRISM, but was not the most efficient choice with respect to memory and time. Using a C solver requires the prototype to sometimes keep two copies of the same expression, one Java-version for further modifications in the main algorithm and one native version to serve the solver. While being memory-inefficient, it also comes at a time penalty: whenever an expression needs to be passed to the solver, the formula needs to be specifically built for the solver via JNI-calls.

4.4.2. Optimizations

During the implementation, we soon discovered the necessity for optimizations that counter some of the challenges of the approach.

Reachability

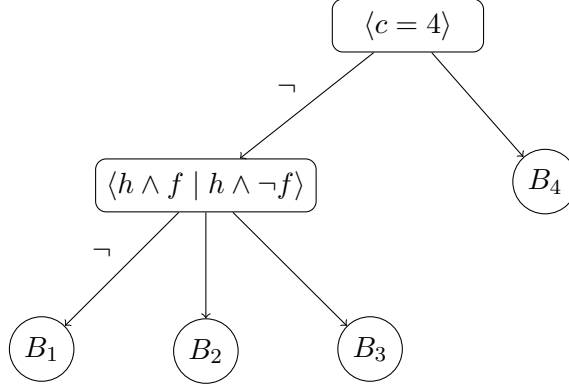
As illustrated by Example 4.6, by default the algorithm may produce unreachable blocks. If this is not detected, they are refined in each iteration, which might even produce unnecessary iterations. This can, however, be partially avoided by keeping track of the current initial block B_{init} and the quotient probability distributions in each block. While the latter is automatically determined in each block refinement step, the former can be determined by letting $B_{init} = \llbracket b_{start} \rrbracket$ such that $b_{start} \in \pi$ and $b(Var) \wedge b_{start} \wedge b(s_{init})$ is satisfiable. We can use this information to perform a breadth-first search through the current quotient model and remove all blocks from the partition that were not visited. Additionally, we use the order of the traversal as a guide to which blocks to refine first. We implemented several splitting orders, which sometimes differ drastically in terms of runtime.

To improve on this countermeasure, we provide the possibility to specify an expression b_{reach} over the variables of the model, which is additionally asserted in each call to the solver. Given that b_{reach} over-approximates the reachable states, this leads to the same

¹<http://yices.csl.sri.com/>

²<http://mathsat4.disi.unitn.it/>

³<http://vlsci.colorado.edu/~hhkim/sateen/>

Figure 4.3.: The partition tree T_π .

quotient system whilst potentially ruling out a lot of unreachable blocks during the computation.

Partition Representation

Following directly from the partition refinement principle, a block B is split into sub-blocks B_1, \dots, B_k such that the B_i are a partition of B . In our setting, this is reflected by the fact that all expressions b_1, \dots, b_k for the sub-blocks of $B = \llbracket b \rrbracket$ share the conjunct b . This suggests to represent the current partition using a tree as follows. An inner node n of the tree carries a tuple of expressions (e_1, \dots, e_m) and has $m + 1$ successors, which represent the sub-blocks characterized by the respective e_i and $\bigwedge_{i=1}^m \neg e_i$. The leaf nodes v represent the current blocks of the partition and are characterized by the conjunction of expressions along the path from the root node to v . If a block $B = \llbracket b \rrbracket$ represented by the leaf node v_B needs to be split, we only need to split v_B using the expressions for the sub-blocks without their common conjunct b . Initially, the root node is assigned the expressions $B(AF(P, \Phi))$.

Example 4.7. The partition tree of Figure 4.3 represents the partition

$$\pi = \left\{ \underbrace{(c \neq 4) \wedge \neg(h \wedge f) \wedge \neg(h \wedge \neg f)}_{B_1}, \underbrace{(c \neq 4) \wedge (h \wedge f)}_{B_2}, \underbrace{(c \neq 4) \wedge (h \wedge \neg f)}_{B_3}, \underbrace{(c = 4)}_{B_4} \right\}.$$

□

Eligible Target Combinations

Besides, we can easily restrict the number of weakest preconditions we have to assert during the refinement of a block B as follows. If in one iteration, we determined that there only exist states $s \in B$ such that $s \xrightarrow{c} (B_1, \dots, B_n)$, we can conclude that for

all sub-blocks of B , we must only consider sub-blocks of B_i as possible target blocks of the i -th assignment of c . In particular, the sub-blocks of B do not need to be refined in case no B_i was split. Also, if only one command is enabled in a block and only some of the target blocks were split, we only need to reassert the weakest preconditions of the blocks that were split. Additionally, it suffices to only assert the parts of the weakest preconditions the new target blocks do not have in common with their ancestors.

Program Flow Variables

Often it is easy to identify variables in a probabilistic program that govern the program flow, i.e. the possible behavior of the program in the next step. Because of this, in most cases the exact value of these variables is preserved by bisimulation minimization, i.e. all states in a bisimulation block agree on the values of the program flow variables, and can thus not be abstracted. We implemented a feature that, given a set of program flow variables, performs an exploration of the model with respect to these variables and uses the results to determine the reachable valuations of these variables to refine the initial partition on the one hand, and to restrict the eligible target combinations for the command (see section 4.4.2) on the other hand.

Example 4.8. In Example 4.6 we observe that c is a program flow variable in P_{Ex} . \square

Simplification of Expressions

As illustrated by the final mapping sig of Example 4.5 for our toy example, the expressions in the computation steps tend to become unnecessarily large and redundant. During first experiments, we noticed that even for small models and small bisimulation quotients, the process ran out of memory because of the excessive size of the expressions. To at least partially stem this problem, we implemented a simplifier capable of performing elementary simplifications, enabling us to treat larger models.

Further Solver Capabilities

The rich MathSat API offers further useful methods for improving performance, such as saving and restoring the assertion stack of the solver instance, which utilizes previous results of satisfiability calls as far as possible.

4.5. Case Studies

The setting we used for conducting our case studies was the same as for the BDD-based bisimulation (see section 3.3). Also, we stick to the diagram forms used there and provide details in terms of the properties checked and exact times in Appendix B. As $AF^{dist}(P, \Phi)$ is not available for the SMT-based approach because we explicitly avoid the exploration of the full model, there is only one bisimulation minimization variant (B)

used in the diagrams. Also note that although the bisimulation minimization is SMT-based, we can give the BDD sizes for the quotient systems, because PRISM transforms the quotient program into its BDD representation for the actual verification.

4.5.1. Synchronous Leader Election Protocol (DTMC)

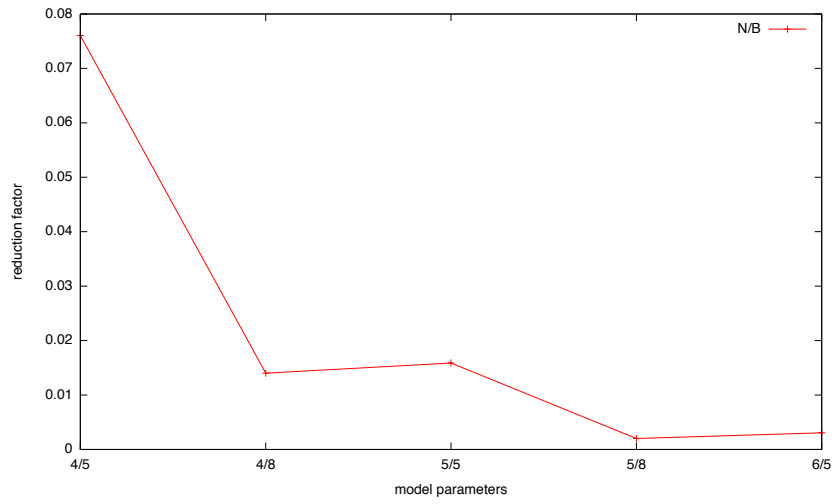
As a first example, we reconsidered the synchronous leader election protocol (see section 3.3.5). The results are shown in Figure 4.4 where diagram (b) coincides with Figure 3.9(b) on page 47 for the common parameters, because we effectively compute an isomorphic quotient system despite the differences of the approaches. However, the resulting BDD sizes of the quotient systems of the SMT-based and the BDD-based approach differ by one order of magnitude (cf. Figure 4.4(c) and Figure 3.9(c)).

As for the BDD-based method, bisimulation minimization is not worthwhile in terms of the total time needed. However, the experiments showed that the crucial requirement for the bisimulation computation is time and not memory. Together with the fact that BDD sizes are reduced by 3 to 4 orders of magnitude, this suggests that with additional runtime improvements it might be possible to build larger models than before with the same memory bounds (recall that with 10GB RAM, PRISM failed to build the system for $N = 7/K = 8$) as, unlike for the BDD-based bisimulation, the full model does not need to be computed first.

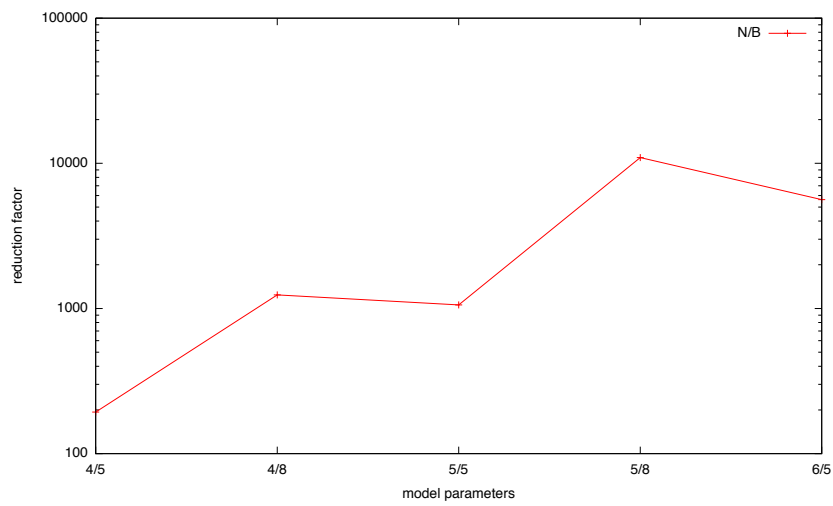
4.5.2. Crowds Protocol (DTMC)

The second case study we reused is a reformulation of the Crowds protocol (see section 3.3.7). We used a reformulation of the model available from the PRISM website, as this model contains a lot of superfluous variables, which unnecessarily increases the number of unreachable blocks. Additionally, we restricted the reachable blocks by overapproximating the reachable state space by $b_{reach} = (\sum_{i=0}^j observe_i \leq runCount)$, which captures the obvious fact that the total number of observations cannot be higher than the number of instances the protocol has been run. Again, Figure 4.5 shows that compared to the BDD-based approach (cf. Figure 3.11), we obtain quotient system BDDs that are one order of magnitude smaller although the number of states is the same.

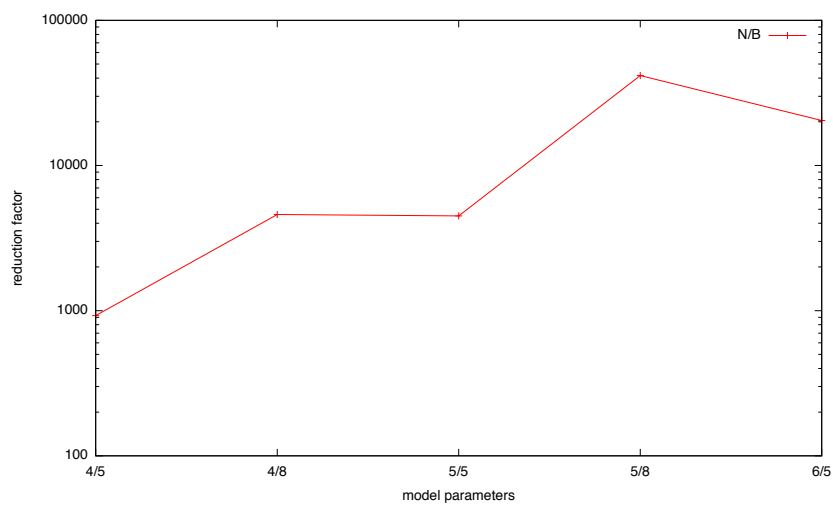
For this example SMT-based bisimulation minimization outperforms both PRISM without abstraction techniques as well as the BDD-based bisimulation minimization in terms of time (see Figure 4.5). In fact, the reduction factor increases with increasing model size. Additionally, not only is the total time reduced by up to 3 orders of magnitude, but the memory requirements are reduced drastically, too. In effect, where PRISM fails to build the model's MTBDD for $N = 25/K = 5$ due to memory limitations, with our approach model checking $N = 100/K = 5$ could be done in less than 1500 seconds with the same memory bound.



(a) time



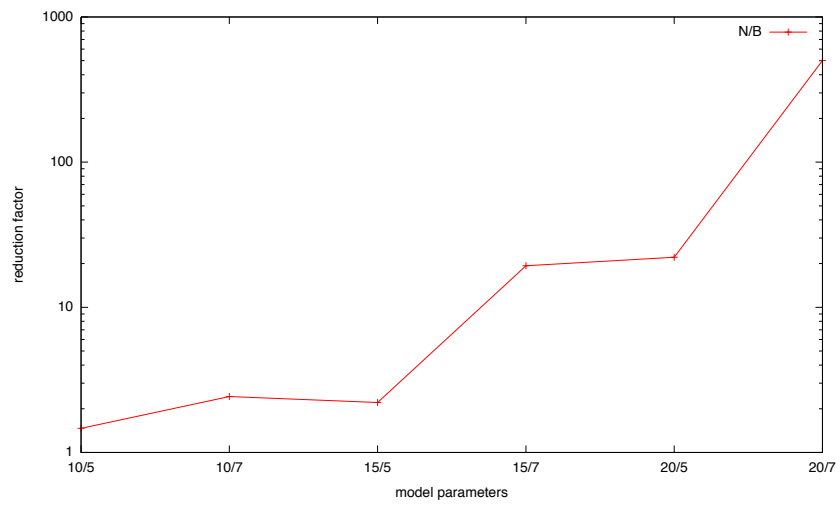
(b) states



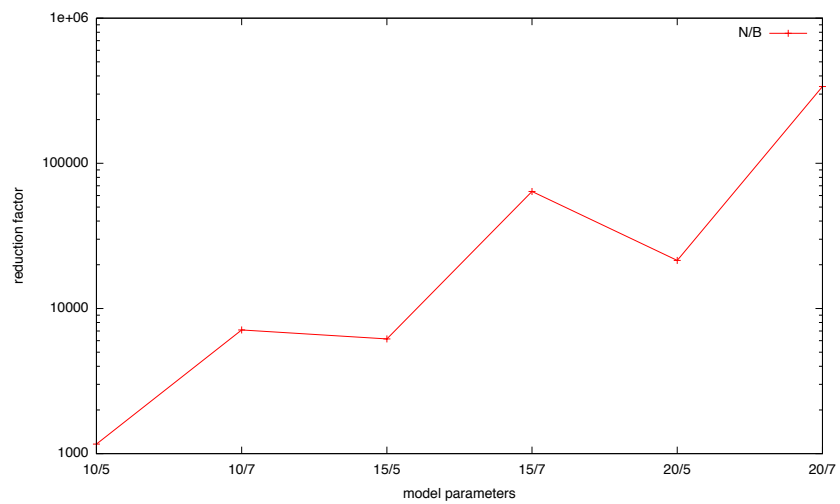
(c) MTBDD

Figure 4.4.: Synchronous Leader Election Protocol results for SMT-based bisimulation.

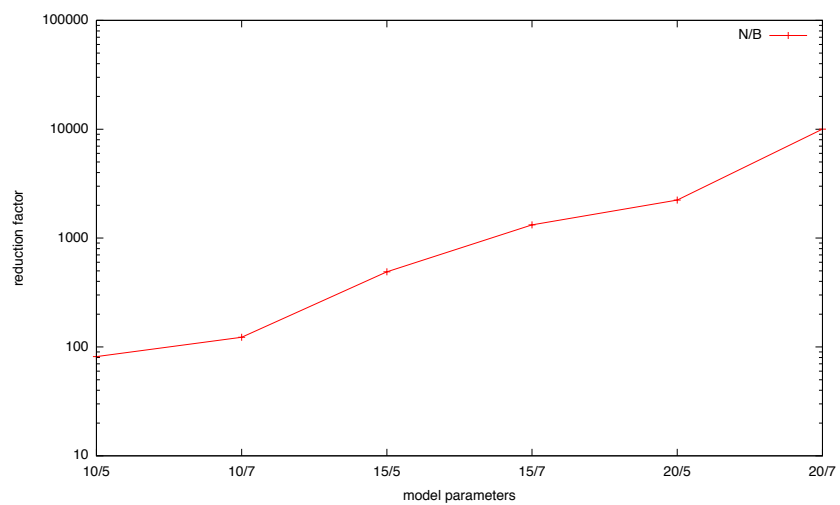
4. SMT-based bisimulation minimization



(a) time



(b) states



(c) MTBDD

Figure 4.5.: Crowds Protocol results for SMT-based bisimulation.

4.6. Analysis

4.6.1. Performance

In terms of time, our prototypical implementation turns out to be worse than BDD-based bisimulation for the synchronous leader election protocol, but better for the crowds protocol. Interestingly, for both examples most of the time is spent on *assertion* calls to the solver (synchronous leader election: 70 – 99%, Crowds protocol: 30 – 70%), whereas the time spent on actual *solve* calls is negligible. The time seems to be used on transforming the asserted formulae into conjunctive normal form (CNF), which suggests that keeping all formulae in CNF might greatly benefit runtime performance. Yet, it remains unclear whether this would worsen memory requirements greatly.

Furthermore, the Crowds protocol example nicely illustrates how time and memory is saved by avoiding to construct the system BDD before minimizing it. In this case, our method enables the analysis of the model for parameters far beyond what PRISM can analyze without abstraction techniques.

Worth mentioning is also the quotient size: for both models we see that the size of the BDD representation of the quotient model is smaller for the SMT-based method than for the BDD-based method. This is due to the fact that we do not need to pick representative states, which include possibly unused variables as in the BDD-based method, but construct another probabilistic program. Experiments suggest that the size of the BDD is closely related to the number of states of the quotient. If this result carried over to MDPs, this would possibly avoid the BDD blow-up we experienced for BDD-based method.

4.6.2. Problems and Improvements

As stated earlier, we faced several problems during the implementation phase, the most significant of which we will once again point out.

Excessive Length of Formulae

The excessive growth of the length of formulae prevented even very small models from being treated successfully. The simplifying steps we implemented improved this situation, but remain naive. As the sizes of the formulae are crucial for both time and memory requirements, it needs to be determined to what extent a better simplification engine improves overall performance.

Unreachable blocks

Our experiments showed that even with a reachability analysis based on the quotient distributions, intermediate partitions were much bigger than the resulting quotient. While providing an expression b_{reach} that over-approximates the reachable state space somewhat mends this problem, it introduces a user interaction that is normally not required for bisimulation quotient computations. Besides, it appears to be non-trivial

to find good expressions for this purpose. A possibility to improve this could be a static analysis of the probabilistic program, deriving such an expression automatically.

4.7. Conclusion

Despite the early development stage of the prototype, experiments are encouraging and show that verifying the bisimulation quotient while avoiding to build the full system representation enables the verification of large systems. Even though it is to be expected that the method will often not outperform others in terms of time, it is still appealing because of the reduced memory requirements. Against the background of the high degree of redundancy in the models we confirmed in chapter 3, this seems to be particularly promising. Nevertheless there remain challenges that have significant impact on the overall performance of the algorithm, such as the simplification of expressions, elimination of unreachable blocks and last but not least reduction of the time spent on solver calls by preprocessing expressions.

5. Conclusion

The future influences the present just as much as the past.

Friedrich Nietzsche

5.1. Summary

We presented two bisimulation quotienting algorithms in a probabilistic and symbolic setting. The first method computes the bisimulation quotient via the manipulation of Binary Decision Diagrams, while the second method formulates the computation of the quotient as a series of satisfiability queries.

For the BDD-based method, we integrated SIGREF, a symbolic bisimulation tool, into the probabilistic model checker PRISM. We extended the main refinement procedure to account for non-deterministic models before we conducted experiments to examine the impact on probabilistic model checking. Our experiments include a comparison with symmetry reduction and the impact of symmetry reduction on a following bisimulation minimization.

The experiments confirmed that common models contain a lot of redundancy that is eliminated by bisimulation minimization. However, unlike in the explicit setting, in almost all cases minimizing is not worthwhile with respect to runtime and memory requirements. Despite the huge state space reductions, we observe a blow-up of the BDD representation in some cases, both for bisimulation minimization as well as symmetry reduction. We also showed that symmetry reduction quotients tend to be relatively close to the coarsest bisimulation regarding the state space size.

The second method we described is a novel technique that computes the quotient system directly from the underlying probabilistic program without building the full state space representation before. To that end, we formulate the refinement procedure of a block in the partition refinement approach as a series of satisfiability problems of the quantifier-free linear integer arithmetic. We pointed out difficulties we encountered and possible countermeasures we applied during a prototypical implementation in which we employ MathSat, a solver for the Satisfiability Modulo Theories problem, to answer our queries.

We present preliminary experiments that show promising results. For one case study, applying the method pushes the bounds on model checkable systems far beyond what can be verified without abstraction techniques. Furthermore, another case study suggests

that avoiding the construction of the full system might reduce the memory requirements for the quotient extraction and verification significantly at the cost of an increase in runtime.

5.2. Future Work

In this section, we focus entirely on future work for the SMT-based bisimulation minimization method. First of all, there is room for the improvement of the overall performance of the prototype:

1. a C++ implementation could avoid duplicate expression representations as all expressions can be represented using the data structures of the satisfiability solver,
2. an improved expression simplification engine could greatly improve both runtime and used memory; in particular, a conjunctive normal form representation of all expressions could reduce runtime drastically,
3. a more sophisticated method of eliminating unreachable blocks, e.g. by constructing an expression to tightly over-approximate the reachable state space via a static analysis of the probabilistic program, can accelerate the computation and reduce memory requirements at the same time, and
4. exploring the potential of further solver capabilities, e.g., determining unsatisfiable cores or interpolant generation, might reduce the complexity of expressions.

Furthermore, more case studies need to be conducted to determine the potential of the method. As our case studies illustrate that huge state space reductions are possible and the quotient extraction mechanism of our approach could potentially avoid BDD blow-ups, it is of particular interest to study the effectiveness on a broader range of examples.

And last, but not least, the extension of the algorithm to non-deterministic systems needs to be tackled. As the non-deterministic variants of Markov models are strictly more expressive than their deterministic counterparts, it is obviously desirable to support them as a bigger class of models.

Appendix

A. Case Studies for BDD-based Bisimulation

A.1. General

- Model construction (without bisimulation minimization) and verification was done by the PRISM model checker.
- All times are given in seconds.
- The BDD sizes are given in terms of the number of nodes.
- The state space size is the number of reachable states of the system.
- For bisimulation minimization, the construction times include the time PRISM needs to build the system BDD and the numbers in parentheses are the number of iterations of the bisimulation algorithm.
- *MO* and *TO* indicate a memory-out and time-out, respectively.

A.2. Properties

Shared Coin Protocol

The properties we verified are the following:

- the probability that the protocol eventually finishes is at least 1, i.e.

$$\mathbb{P}_{\geq 1}(true \mathbf{U} finished),$$

- what is the minimum probability that the protocol finishes with the value of all coins being 0, i.e.

$$\mathbb{P}_{=?}^{min}(true \mathbf{U} finished \wedge all_coins = 0),$$

- what is the minimum probability that the protocol finishes with the value of all coins being 1, i.e.

$$\mathbb{P}_{=?}^{min}(true \mathbf{U} finished \wedge all_coins = 1),$$

- what is the maximum probability that the protocol finishes without an agreement, i.e.

$$\mathbb{P}_{=?}^{max}(true \mathbf{U} finished \wedge \neg agree),$$

- what is the maximum probability that the protocol finishes within 40 timesteps, i.e.

$$\mathbb{P}_{=?}^{max}(true \mathbf{U}^{\leq 40} finished),$$

-
- what is the minimum probability that the protocol finishes within 40 timesteps, i.e.

$$\mathbb{P}_{=?}^{min} (true \mathbf{U}^{\leq 40} finished),$$

- what is the maximum expected number of steps needed until the protocol is finished, i.e.

$$\mathbb{E}_{=?}^{max} (true \mathbf{U} finished),$$

and

- what is the minimum expected number of steps needed until the protocol is finished, i.e.

$$\mathbb{E}_{=?}^{min} (true \mathbf{U} finished),$$

where

- *finished* characterizes states in which the run of the protocol is finished,
- *agree* characterizes states in which all processes agree, and
- *all_coins = i* for $i \in \{0, 1\}$ characterizes states in which the value of all coins is 0 or 1, respectively.

Firewire

The properties we verified are the following:

- what is the minimum probability of completing within the deadline, i.e.

$$\mathbb{P}_{=?}^{min} (true \mathbf{U} finished).$$

where *finished* characterizes states in which the run of the protocol finished.

Peer-To-Peer Protocol

The properties we verified are the following:

- what is the probability that all clients received all parts of the file within 1 time-unit, i.e.

$$\mathbb{P}_{=?} (true \mathbf{U}^{[0,1]} finished),$$

and

- what is the expected fraction of blocks received by time 1, i.e.

$$\mathbb{E}_{=?} (I = 1),$$

where *finished* characterizes states in which all clients are in possession of all parts.

Wireless Network

The properties we verified are the following:

- what is the maximum probability that the maximal number of collisions occurred, i.e.

$$\mathbb{P}_{=?}^{max}(true \mathbf{U} occurred_collisions = C).$$

Synchronous Leader Election

The properties we verified are the following:

- what is the probability that a leader is eventually elected, i.e.

$$\mathbb{P}_{=?}(true \mathbf{U} elected),$$

- what is the probability that a leader is elected within 2 rounds, i.e.

$$\mathbb{P}_{=?}(true \mathbf{U}^{\leq 2 \cdot (N+1)} elected),$$

and

- what is the expected number of steps needed to elect a leader, i.e.

$$\mathbb{E}_{=?}(true \mathbf{U} elected),$$

where *elected* characterizes states in which a single leader has been elected.

Zeroconf Protocol

The properties we verified are the following:

- what is the minimum probability that the host is using a valid IP address, i.e.

$$\mathbb{P}_{=?}^{min}(true \mathbf{U} using_valid_ip),$$

where *using_valid_ip* characterizes states in which the host is using a valid IP address.

Crowds Protocol

The properties we verified are the following:

- what is the probability that the original sender is observed more than once by bad crowd members, i.e.

$$\mathbb{P}_{=?}(true \mathbf{U} observe_0 > 1),$$

where *observe₀* is the number of instances the original sender was observed by bad crowd members.

A.3. Tables

Shared Coin Protocol

normal					
N	K	states	MTBDD	construction	verification
4	4	43136	2336	0.178	2825.793
5	4	327936	4325	0.299	13536.28
6	4	2376448	7055	0.593	42625.96
7	4	16695808	11031	0.957	99462.187
8	4	114757632	15888	1.455	217338.47

Bisimulation using $\Pi_{AF(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	4	3473	7417	101.532 (106)	2124.593
5	4	8472	21515	1009.142 (135)	9039.668
6	4	17939	44935	9327.026 (164)	28343.306
7	4	MO	MO	MO	MO
8	4	MO	MO	MO	MO

Bisimulation using $\Pi_{AF^{dist}(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	4	3473	4309	460.555 (48)	1966.711
5	4	8472	8449	3500.546 (60)	8955.59
6	4	17939	7853	25898.081 (72)	25192.973
7	4	TO	TO	TO	TO
8	4	TO	TO	TO	TO

Table 5.1.: Shared Coin Protocol (without prior symmetry reduction)

normal					
N	K	states	MTBDD	construction	verification
4	4	4087	2131	0.277	1363.752
5	4	10434	3881	0.551	5636.141
6	4	23233	6347	0.983	15276.103
7	4	46816	9902	1.731	32475.929
8	4	87378	14285	2.702	59132.891

Bisimulation using $\Pi_{AF(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	4	3473	8430	19.305 (106)	1699.343
5	4	8472	18638	75.292 (135)	6862.614
6	4	17939	45626	242.933 (164)	20691.979
7	4	34322	49626	681.559 (193)	41447.138
8	4	60862	142858	1613.902 (222)	85897.595

Bisimulation using $\Pi_{AF^{dist}(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	4	3473	5419	157.016 (48)	1670.149
5	4	8472	11983	840.654 (60)	6305.614
6	4	17939	16028	3261.225 (72)	17472.038
7	4	34322	52290	13763.587 (84)	40115.342
8	4	60862	28354	45562.005 (96)	68286.201

Table 5.2.: Shared Coin Protocol (with prior symmetry reduction)

Firewire

normal					
<i>DE</i>	<i>DL</i>	states	MTBDD	construction	verification
3	200	14824	5163	1.415	0.262
3	400	69683	8453	4.557	2.737
3	600	168411	10902	9.5	9.662
36	200	68056	9711	1.494	0.626
36	400	220565	10394	4.313	5.652
36	600	375765	10421	8.781	13.17
Bisimulation using $\Pi_{AF(P,\Phi)}$					
<i>DE</i>	<i>DL</i>	states	MTBDD	construction	verification
3	200	521	4438	10.4 (162)	0.197
3	400	15740	14096	425.901 (408)	2.233
3	600	19360	20507	799.652 (614)	6.942
36	200	10844	14066	129.878 (205)	0.446
36	400	74042	28296	1633.578 (408)	6.147
36	600	145242	28029	4992.183 (611)	16.508
Bisimulation using $\Pi_{AF^{dist}(P,\Phi)}$					
<i>DE</i>	<i>DL</i>	states	MTBDD	construction	verification
3	200	521	3845	40.491 (161)	0.183
3	400	15740	14920	987.697 (253)	2.399
3	600	19360	12517	3001.348 (613)	6.16
36	200	10844	13217	362.623 (195)	0.44
36	400	74042	24955	4151.514 (283)	9.121
36	600	145242	25749	13545.012 (486)	17.941

Table 5.3.: Firewire Tree Identify Protocol

Peer-To-Peer Protocol

normal					
N	K	states	MTBDD	construction	verification
4	5	1048576	11941	0.069	107.058
5	5	33554432	26266	0.202	914.72
6	5	1073741824	40591	0.181	4617.71
7	5	34359738368	54916	0.398	19936.175
8	5	1099511627776	69241	0.525	TO

Bisimulation using $\Pi_{AF(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	5	126	834	2.607 (3)	2.822
5	5	196	1626	14.963 (3)	14.888
6	5	266	2152	52.882 (3)	33.184
7	5	336	3058	118.273 (3)	91.409
8	5	406	3895	478.318 (3)	109.216

Bisimulation using $\Pi_{AF^{dist}(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	5	126	834	6.8 (3)	3.181
5	5	196	1626	37.974 (3)	15.529
6	5	266	2152	154.191 (3)	41.08
7	5	336	3058	481.835 (3)	85.451
8	5	406	3895	1250.506 (3)	151.221

Table 5.4.: Peer-To-Peer Protocol (without prior symmetry reduction)

normal					
N	K	states	MTBDD	construction	verification
4	5	52360	42166	0.683	181.01
5	5	376992	101630	1.882	1350.597
6	5	2324784	189704	3.428	6917.427
7	5	12620256	306123	5.908	24013.256
8	5	61523748	449599	9.964	TO

Bisimulation using $\Pi_{AF(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	5	126	2682	3.441 (3)	5.23
5	5	196	5522	21.785 (3)	21.833
6	5	266	8344	65.259 (3)	50.841
7	5	336	11718	160.864 (3)	141.391
8	5	406	14778	442.952 (3)	134.394

Bisimulation using $\Pi_{AF^{dist}(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	5	126	2682	8.813 (3)	5.388
5	5	196	5522	43.254 (3)	29.401
6	5	266	8344	146.212 (3)	58.817
7	5	336	11718	378.127 (3)	176.182
8	5	406	14778	821.359 (3)	271.171

Table 5.5.: Peer-To-Peer Protocol (with prior symmetry reduction)

Wireless Network

normal					
N	COL	states	MTBDD	construction	verification
4	4	345120	25145	1.134	12.706
4	6	728990	27997	1.443	150.909
4	8	1339700	28079	1.753	883.069
5	4	1295338	28959	1.483	13.974
5	6	1591710	35153	1.597	169.738
5	8	3927860	35248	2.135	1661.668

Bisimulation using $\Pi_{AF(P,\Phi)}$					
N	COL	states	MTBDD	construction	verification
4	4	2777	51154	248.845 (196)	13.97
4	6	12197	168535	4525.905 (772)	225.677
4	8	24753	173893	7885.25 (822)	1280.47
5	4	2777	52724	312.409 (196)	14.12
5	6	12197	173599	4827.678 (772)	259.521
5	8	37297	334885	30230.154 (1565)	2443.98

Bisimulation using $\Pi_{AF^{dist}(P,\Phi)}$					
N	COL	states	MTBDD	construction	verification
4	4	2777	53206	1236.101 (191)	13.525
4	6	12197	170335	50611.731 (767)	228.413
4	8	24753	201124	94672.551 (767)	1418.992
5	4	2777	46621	1849.978 (191)	13.197
5	6	12197	194550	56156.138 (767)	296.151
5	8	TO	TO	TO	TO

Table 5.6.: Wireless Network Collision Avoidance Protocol

Synchronous Leader Election

normal					
N	K	states	MTBDD	construction	verification
4	5	1933	34246	0.282	0.572
4	8	12400	179139	2.809	3.578
5	5	12709	184363	2.609	4.373
5	8	131521	1540390	123.112	34.011
6	5	78784	837485	42.229	30.953
6	8	1312334	11384030	11970.266	1163.058
Bisimulation using $\Pi_{AF(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	5	10	632	2.288 (9)	0.02
4	8	10	632	11.476 (9)	0.021
5	5	12	920	15.398 (11)	0.033
5	8	12	931	193.607 (11)	0.033
6	5	14	1283	137.698 (13)	0.07
6	8	14	1288	12708.643 (13)	0.053
Bisimulation using $\Pi_{AF^{dist}(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	5	10	632	1.992 (2)	0.022
4	8	10	632	11.01 (2)	0.02
5	5	12	920	12.022 (2)	0.039
5	8	12	931	183.061 (2)	0.039
6	5	14	1283	116.896 (2)	0.06
6	8	14	1288	15617.917 (2)	0.056

Table 5.7.: Synchronous Leader Election Protocol

Zeroconf Protocol

normal					
N	K	states	MTBDD	construction	verification
4	32	26121	134295	6.656	108.35
4	128	98889	558167	36.7	111.213
8	32	552097	547191	24.585	2356.859
8	128	2092513	2316397	141.627	2086.214

Bisimulation using $\Pi_{AF(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	32	940	37970	465.306 (110)	117.294
4	128	940	39708	1044.079 (110)	117.445
8	32	1808	117417	11741.426 (126)	886.207
8	128	1808	148174	20468.535 (126)	723.739

Bisimulation using $\Pi_{AF^{dist}(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	32	940	37193	324.791 (26)	118.873
4	128	940	40053	485.193 (26)	113.689
8	32	1808	102271	5491.823 (30)	858.623
8	128	1808	119204	9923.045 (30)	708.148

Table 5.8.: Zeroconf IP Address Protocol

Crowds Protocol

normal					
N	R	states	MTBDD	construction	verification
10	5	110562	30744	0.376	6.644
10	7	1002451	64334	0.708	16.595
15	5	586242	183669	6.212	11.125
15	7	9028168	676568	107.005	131.938
20	5	2036647	831299	248.088	49.74
20	7	47657605	5133569	9873.882	893.677

Bisimulation using $\Pi_{AF(P,\Phi)}$					
N	R	states	MTBDD	construction	verification
10	5	95	1709	16.295 (18)	0.027
10	7	141	1931	63.288 (26)	0.028
15	5	95	2177	155.554 (18)	0.033
15	7	141	2399	1331.351 (26)	0.041
20	5	95	2699	1079.177 (18)	0.028
20	7	141	2921	17549.656 (26)	0.043

Bisimulation using $\Pi_{AF^{dist}(P,\Phi)}$					
N	R	states	MTBDD	construction	verification
10	5	95	1709	15.29 (16)	0.02
10	7	141	1931	65.2 (24)	0.027
15	5	95	2177	133.539 (16)	0.023
15	7	141	2399	1245.045 (24)	0.042
20	5	95	2699	896.606 (16)	0.026
20	7	141	2921	19528.49 (24)	0.043

Table 5.9.: Crowds Protocol

B. Case Studies for SMT-based Bisimulation

B.1. General

- Model construction (without bisimulation minimization) and verification was done by the PRISM model checker.
- All times are given in seconds.
- The BDD sizes are given in terms of the number of nodes.
- The state space size is the number of reachable states of the system.
- For bisimulation minimization, the construction times include the time PRISM needs to build the system BDD.
- *MO* and *TO* indicate a memory-out and time-out, respectively.

B.2. Properties

Synchronous Leader Election

The properties we verified are the following:

- what is the probability that a leader is eventually elected, i.e.

$$\mathbb{P}_{=?} (true \mathbf{U} elected),$$

and

- what is the probability that a leader is elected within 2 rounds, i.e.

$$\mathbb{P}_{=?} (true \mathbf{U}^{\leq 2 \cdot (N+1)} elected),$$

where *elected* characterizes states in which a single leader has been elected.

Crowds Protocol

The properties we verified are the following:

- what is the probability that the original sender is observed more than once by bad crowd members, i.e.

$$\mathbb{P}_{=?} (true \mathbf{U} observe_0 > 1),$$

where *observe₀* is the number of instances the original sender was observed by bad crowd members.

B.3. Tables

Synchronous Leader Election

normal					
N	K	states	MTBDD	construction	verification
4	5	1933	34246	0.282	0.572
4	8	12400	179139	2.809	3.578
5	5	12709	184363	2.609	4.373
5	8	131521	1540390	123.112	34.011
6	5	78784	837485	42.229	30.953
6	8	1312334	11384030	11970.266	1163.058

Bisimulation using $\Pi_{AF(P,\Phi)}$					
N	K	states	MTBDD	construction	verification
4	5	10	37	11.229	0.002
4	8	10	39	455.976	0.002
5	5	12	41	439.928	0.003
5	8	12	37	77740.161	0.002
6	5	14	41	23908.122	0.003
6	8	TO	TO	TO	TO

Table 5.10.: Synchronous Leader Election Protocol

Crowds Protocol

normal					
N	R	states	MTBDD	construction	verification
10	5	110562	30744	0.376	6.644
10	7	1002451	64334	0.708	16.595
15	5	586242	183669	6.212	11.125
15	7	9028168	676568	107.005	131.938
20	5	2036647	831299	248.088	49.74
20	7	47657605	5133569	9873.882	893.677
25	5	MO	MO	MO	MO

Bisimulation using $\Pi_{AF(P,\Phi)}$ and b_{reach}					
N	R	states	MTBDD	construction	verification
10	5	95	377	4.787	0.005
10	7	141	524	7.122	0.009
15	5	95	375	7.849	0.006
15	7	141	512	12.345	0.009
20	5	95	372	13.456	0.011
20	7	141	512	21.46	0.009
25	5	95	364	458.528	0.007
70	5	95	360	417.466	0.007
100	5	95	359	1448.328	0.007

where $b_{reach} = (\sum_{i=0}^{N-1} observe_i \leq runCount)$.

Table 5.11.: Crowds Protocol

C. Reformulation of the Crowds Protocol

We present the model used for verifying the crowds protocol by means of an example ($N = 2$ and $R = 3$). Note that the actual probabilities remain unchanged compared to the model on the PRISM website.

```

// probability of forwarding
const double PF = 0.8;
// probability that a crowd member is an attacker
const double bad = 0.167;
// total size of the crowd and number of times protocol is run
const int CrowdSize = 2;
const int TotalRuns = 3;

// the phase the protocol is in


p : [0..4] init 0;


// store whether the current crowd member is good


g : bool init false;


// the number of times the protocol was already run


runCount : [0..TotalRuns] init 0;


// observei is the number of times attackers observed i as the sender


observe0 : [0..TotalRuns] init 0;



observe1 : [0..TotalRuns] init 0;


// store the most recently seen crowd member


lastSeen : [0..CrowdSize - 1] init 0;



// start the protocol run


[start] p = 0 1 : (p' = 1)



& runCount < TotalRuns → & (runCount' = runCount + 1)



& (lastSeen' = 0);


// decide whether the current crowd member is good or bad


[pick] p = 1 → (1 - bad) : (p' = 2) & (g' = true)



+ bad : (p' = 2) & (g' = false);


// if crowd member is good, choose the most recently seen index uniformly


[upd] p = 2 & g → 1/2 : (lastSeen' = 0) & (p' = 3)



+ 1/2 : (lastSeen' = 1) & (p' = 3);


// bad crowd members observe the most recently seen index


[obs0] p = 2 & !g



& lastSeen = 0 → (observe0' = observe0 + 1) & (p' = 4);



& observe0 < TotalRuns



[obs1] p = 2 & !g



& lastSeen = 1 → (observe1' = observe1 + 1) & (p' = 4);



& observe1 < TotalRuns


// forward the message with probability PF and deliver it otherwise


[fw] p = 3 → PF : (p' = 1) + (1 - PF) : (p' = 4);


// restart the protocol


[loop] p = 4 → (p' = 0);


```

Bibliography

- [1] CHRISTEL BAIER AND JOOST-PIETER KATOEN. *Principles of Model Checking*, The MIT Press, Cambridge (MA), 2008.
- [2] ROBERT FLOYD. Assigning meanings to programs. In: *Proceedings of Symposium on Applied Mathematics*, volume 19, pages 19-32, 1967.
- [3] C. A. R. HOARE. An axiomatic basis for computer programming. In: *Communications of the ACM*, volume 12(10), pages 576–580, 1969.
- [4] JOOST-PIETER KATOEN AND TIM KEMNA AND IVAN ZAPREEV AND DAVID N. JANSEN. Bisimulation minimisation mostly speeds up probabilistic model checking. In: *13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *Lecture Notes in Computer Science*, pages 87-101, Springer, Berlin, 2007.
- [5] HOLGER HERMANNNS AND BJÖRN WACHTER AND LIJUN ZHANG. Probabilistic CEGAR. In: *20th International Conference on Computer Aided Verification (CAV)*, volume 5123 of *Lecture Notes in Computer Science*, pages 162-175, Springer, Berlin, 2008.
- [6] MARK KATTENBELT AND MARTA KWIATKOWSKA AND GETHIN NORMAN AND DAVID PARKER. A game-based abstraction-refinement framework for Markov decision processes. In: *Formal Methods in System Design*, volume 36, pages 246-280, Springer, Berlin, 2010.
- [7] MARTA KWIATKOWSKA AND GETHIN NORMAN AND DAVID PARKER AND HONGYANG QU. Assume-Guarantee Verification for Probabilistic Systems. In: *16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6015 of *Lecture Notes in Computer Science*, pages 23-37, Springer, Berlin, 2010.
- [8] SUZANA ANDOVA AND HOLGER HERMANNNS AND JOOST-PIETER KATOEN. Discrete-Time Rewards Model-Checked. In: *First International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 2791 of *Lecture Notes in Computer Science*, pages 88-104, Springer, Berlin, 2004.
- [9] MARTA KWIATKOWSKA AND GETHIN NORMAN AND DAVID PARKER. Stochastic Model Checking. In: *7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM)*, volume 4486 of *Lecture Notes in Computer Science*, pages 220-270, Springer, Berlin, 2007.

- [10] ADNAN AZIZ AND VIGYAN SINGHAL AND FELICE BALARIN AND ROBERT K. BRAYTON AND ALBERTO L. SANGIOVANNI-VINCENTELLI. In: *7th International Conference on Computer Aided Verification (CAV)*, volume 939 of *Lecture Notes in Computer Science*, pages 155-165, Springer, Berlin, 1995.
- [11] MARTA KWIATKOWSKA AND GETHIN NORMAN AND DAVID PARKER. PRISM: Probabilistic Symbolic Model Checker. In: *12th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools (TOOLS)*, volume 2324 of *Lecture Notes in Computer Science*, pages 200-204, Springer, Berlin, 2002.
- [12] CHRISTEL BAIER AND BOUDEWIJN HAVERKORT AND HOLGER HERMANNNS AND JOOST-PIETER KATOEN. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE TSE* 29, pages 524-541, 2003.
- [13] JOOST-PIETER KATOEN AND IVAN S. ZAPREEV AND ERNST MORITZ HAHN AND HOLGER HERMANNNS AND DAVID N. JANSEN. The ins and outs of the probabilistic model checker MRMC. In: *Advances in Quantitative Evaluation of Systems (QEST 2009)*, volume 68 of *Performance Evaluation*, pages 90-104, 2011.
- [14] LUCA DE ALFARO AND MARTA KWIATKOWSKA AND GETHIN NORMAN AND DAVID PARKER AND ROBERTO SEGALA. Symbolic Model Checking of Probabilistic Processes Using MTBDDs and the Kronecker Representation. In: *6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1785 of *Lecture Notes in Computer Science*, pages 395-410, Springer, Berlin, 2011.
- [15] MARTA KWIATKOWSKA AND GETHIN NORMAN AND DAVID PARKER. Symmetry Reduction for Probabilistic Model Checking. In: *18th International Conference on Computer Aided Verification (CAV)*, volume 4144 of *Lecture Notes in Computer Science*, pages 234-248, Springer, Berlin, 2006.
- [16] RALF WIMMER AND MARC HERBSTTRITT AND HOLGER HERMANNNS AND KELLEY STRAMPP AND BERND BECKER. Sigref – A Symbolic Bisimulation Tool Box. In: *4th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 4218 of *Lecture Notes in Computer Science*, pages 477-492, Springer, Berlin, 2006.
- [17] CLARK BARRETT AND CHRISTOPHER L. CONWAY. Leveraging SMT: Using SMT Solvers to Improve Verification; Using Verification to Improve SMT Solvers. <https://www.cs.nyu.edu/~cconway/papers/conway-2010-smt.pdf>.
- [18] EDMUND CLARKE AND ARMIN BIERE AND RICHARD RAIMI AND YUNSHAN ZHU. Bounded Model Checking Using Satisfiability Solving. In: *Formal Methods in System Design*, volume 19, pages 7-34, 2001.

-
- [19] ALESSANDRO ARMANDO AND JACOPO MANTOVANI AND LORENZO PLATANIA. Bounded Model Checking of Software Using SMT Solvers Instead of SAT Solvers. In: *13th International SPIN Workshop*, volume 3925 of *Lecture Notes in Computer Science*, pages 146-162, Springer, Berlin, 2006.
- [20] ARMIN BIERE AND ALESSANDRO CIMATTI AND EDMUND CLARKE AND YUNSHAN ZHU. Symbolic Model Checking without BDDs. In: *5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1579 of *Lecture Notes in Computer Science*, pages 193-207, Springer, Berlin, 1999.
- [21] JAMES ASPNES AND MAURICE HERLIHY. Fast randomized consensus using shared memory. In: *Journal of Algorithms*, 15 (1), pages 441-460, 1990.
- [22] MARTA KWIATKOWSKA AND GETHIN NORMAN AND JEREMY SPROSTON. Probabilistic Model Checking of Deadline Properties in the IEEE1394 FireWire Root Contention Protocol. In: *Formal Aspects of Computing*, 14 (3), pages 295-318, 2003.
- [23] Simple Peer-To-Peer Protocol Case Study. <http://www.prismmodelchecker.org/casestudies/peer2peer.php>.
- [24] IEEE 802.11 Wireless LAN Case Study. <http://www.prismmodelchecker.org/casestudies/wlan.php>.
- [25] ALON ITAI AND MICHAEL RODEH. Symmetry breaking in distributed networks. In: *Information and Computation*, 88 (1), 1990.
- [26] S. CHESHIRE, B. ADOBA AND E. GUTTERMAN. Dynamic configuration of IPv4 link local addresses. <http://www.ietf.org/rfc/rfc3927.txt>.
- [27] MICHAEL REITER AND AVIEL RUBIN. Crowds: anonymity for Web transactions. In: *ACM Transactions on Information and System Security (TISSEC)*, volume 1 (1), 1998.