MODEL REDUCTION TECHNIQUES FOR PROBABILISTIC VERIFICATION OF MARKOV CHAINS

by

NISHANTHAN KAMALESON

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
April 2018

Abstract

Probabilistic model checking is a quantitative verification technique that aims to verify the correctness of probabilistic systems. Nevertheless, it suffers from the so-called state space explosion problem. In recent years, many model reduction techniques have been introduced to reduce the impact of this problem in the context of probabilistic verification.

In this thesis, we propose two new model reduction techniques to improve the efficiency and scalability of verifying probabilistic systems, focusing on the commonly used model of discrete-time Markov chains (DTMCs). In particular, unlike most existing approaches, our emphasis is on verifying quantitative properties that bound the time or cost of an execution. We also focus on methods that avoid the explicit construction of the full state space, which can be a bottleneck for some existing techniques.

We first present a finite-horizon variant of probabilistic bisimulation for DTMCs, which preserves a bounded fragment of PCTL, the most widely used temporal logic for specifying properties of this model. The goal is to enable a more aggressive reduction of the model than can be achieved when preserving the full logic. We propose two techniques to perform minimisation with respect to this notion of bisimulation: a standard partition-refinement based algorithm and an on-the-fly finite-horizon approach, based on a backwards traversal of the Markov chain, directly from a high-level model description.

We also propose another model reduction technique that reduces what we call linear inductive DTMCs, a class of models whose state space grows linearly with respect to a parameter. We devise methods that automatically detect and extract such models from a high-level model description, and then perform model checking via construction and solution of a set of recurrence relations. We also show how verifying step-bounded and cost-bounded probabilistic reachability properties on arbitrary DTMCs reduces to the problem of verifying linear inductive DTMCs.

All the techniques presented in this thesis were developed as a complete implementation in the PRISM model checker. We demonstrate the effectiveness of our work by applying it to a selection of existing benchmark probabilistic models, showing that both of our two new approaches can provide significant reductions in model size and in some cases outperform the existing implementations of probabilistic verification in PRISM.

Acknowledgements

Firstly, I would like to thank my supervisor, David Parker, for his impeccable support, guidance and motivation without which this thesis would never have been completed. I am also grateful to my co-supervisor, Jonathan Rowe, for his encouragement and guidance. I am obliged to my dear friend Pietro Consoli for his timely support and valuable comments and suggestions which made a great contribution in improving and developing this thesis. Thanks must also go to Hieratic Project and School of Computer Science, University of Birmingham for funding my studies and providing required resources to carry out this research. Finally, I would like to thank my mother, sister and girlfriend for their positive motivation and support throughout.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Computerised systems are now integral to all aspects of our society, including safety-critical domains such as the embedded systems in everything from cars to planes to medical devices. *Formal verification* is an approach that uses mathematical techniques to confirm the correctness of a computerised system during its design phase. Given a model of a design, some description of the environment where the system will be executed and specifications of correctness properties that are to be satisfied by the system, formal verification enables us to ensure the absence of errors. In addition to this, it often allows us to detect scenarios that could invalidate the specifications. In contrast to *testing*, an alternative non-exhaustive approach to check correctness of a system, formal verification is expensive in terms of computational resources and this can be a potential bottleneck.

In the early stages, most research in formal verification focused on developing techniques for analysing *qualitative* properties of critical systems, for example, whether a message is eventually delivered, or a certain pair of parallel process never violates a mutual exclusion property. However, systems with probabilistic behaviour and real-time delays requires analysis of *quantitative* properties. The following properties: "the probability of an airbag failing to deploy within 0.01s" or "the expected power usage of a sensor network over 1 hour" are examples of quantitative properties. *Quantitative verification* is a formal technique that generalises formal verification and examines correctness of quantitative properties of a complex system.

The term model checking [17] refers to a formal verification technique that automatically verifies whether a finite state model of a system satisfies its specifications, which are typically expressed in temporal logic. In contrast to traditional techniques such as simulation and testing, this technique exhaustively explores through the whole state space. This approach has been successfully applied in industry to benefit in the verification of various complex designs. For example, Clarke et al. have constructed an exact model of the cache coherence protocol, which is described in the IEEE Futurebus1 Standard 896.1-1991, in the SMV input language [69] and concluded that the resulting transition system satisfied a formal specification of cache coherence [15]. In addition to this, they have identified a number of possible errors and ambiguities that had not been detected by the informal techniques which were applied before to validate the protocol. This was the first time that formal methods were used to find non-trivial errors in a proposed IEEE standard.

Probabilistic model checking is a quantitative verification technique and a generalisation of model checking that builds a probabilistic model and analyses it based on the formally specified properties. In this formal method, specifications of complex systems are expressed in quantitative extensions of temporal logic and the systems are modelled as finite state probabilistic models such as Markov chains and Markov decision process. The use of temporal logic gives formal and unambiguous definitions of properties to be verified against the model. This approach allows a user to verify if a finite state probabilistic model satisfies a given specification. A probabilistic model checker is a tool that allows a user to query such models and provides answers or counterexamples to the given queries. The two most widely used probabilistic model checkers are PRISM [59] and MRMC [56].

Probabilistic model checking has been applied in various domains such as communication and security protocols, biological modelling and so on. For example, probabilistic model checking was used as part of a failure analysis for a car airbag system in [1]. The analysis discovered that certain specification were violated in the one-processor variant of the system. Moreover, the use of counterexamples revealed the critical aspect of this violation was the failure of the micro-processor.

Overall, probabilistic model checking has proven to be a powerful formal verification method. Nevertheless, it requires exhaustive state space exploration of a given system, where the state space may increase exponentially along with the complexity of the system. As a consequence, these verification techniques suffer from the so-called *state space explosion problem*. In the context of model checking, many researchers [71, 10, 68, 29, 30, 80, 55, 62] have developed various model reduction techniques to combat this problem. Some examples of model reduction techniques are symmetry reduction, abstraction refinement, bisimulation equivalences and partial order reduction.

In this thesis, we propose two novel model reduction techniques which aim to tackle the state space explosion problem. In particular, we focus on quantitative properties, which impose bounds on either the time or cost needed for some event. Another goal is to avoid the explicit construction of the full state space of the model, which can be a bottleneck for some other model reduction approaches.

We first present a model reduction technique for verifying finite-horizon properties on DTMCs. We have formalised the notion of finite-horizon bisimulation minimisation and clarified the subset of probabilistic computation tree logic that it preserves. The motivation of this technique is to perform more aggressive reduction of the model by restricting the expensive iterations of the minimisation process. We have implemented both a partition-refinement minimisation algorithm and an on-the-fly approach, implemented in both a symbolic (based on satisfiability modulo theories) and explicit-state manner.

We also present an inductive model reduction technique for verifying reachability properties on linearly inductive DTMCs, a class of models whose state space grows linearly with respect to a parameter. We introduce methods to automatically detect and extract such models from a high-level modelling description, in our case, the PRISM modelling language and give techniques for model checking the reduced model by building and solving a set of recurrence relations. The result is a function which can be used to verify the model for any value of the parameter. This approach is extended to verify step-bounded and cost-bounded reachability properties on normal DTMCs, which reduces to the problem of verifying linearly inductive DTMCs.

We have implemented both of the two model reduction techniques in the PRISM model checker. We demonstrate the effectiveness of our approaches by applying them to several benchmark models. We show that both methods can provide significant reductions in model size and are sometimes also able to outperform the existing implementations of probabilistic verification in PRISM.

1.1 Publications

The finite-horizon bisimulation approach, presented in Chapter 4, has been published as a jointly-authored paper [50]. A paper describing the inductive model reduction technique presented in Chapter 5 is currently in preparation.

1.2 Thesis outline

The thesis is structured as follows. In Chapter 2, we introduce the relevant background material that is required to understand the work presented in this thesis. This includes details of DTMCs, PCTL, the computation of probabilistic reachability, probabilistic bisimulation and, fundamental definitions of recurrence relations and generating functions. Chapter 3 provides a literature review of closely related work and discusses the difference between our work, presented in this thesis, and the existing related work. In Chapter 4, we present a finite-horizon bisimulation minimisation technique for DTMCs, which preserves a bounded fragment of PCTL. In Chapter 5, we introduce another model reduction technique that reduces linearly inductive models with respect to a parameter. Finally, we summarise all the work presented in this thesis and point out possible extensions of the current work as future directions in Chapter 6.

CHAPTER 2

Background Material

In this chapter, we present the relevant background material for this thesis. In Section 2.1 and 2.2, we present the formal definitions of discrete-time Markov chains and the probabilistic computation tree logic. We explain the underlying computation for the model checking of reachability properties in Section 2.3. Section 2.4 presents the formal definitions relevant to probabilistic bisimulation. Finally, Section 2.6 and Section 2.7 presents the formal definitions of relevant recurrence relations and generating functions for Chapter 5.

2.1 Discrete-time Markov Chains

A discrete-time Markov chain (DTMC) can be thought of as a state transition system where transitions between states are annotated with probabilities.

Definition 2.1.1 (DTMC) *A DTMC is a tuple* $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$*, where:*

- $\mathcal{S}$ *is a finite set of states and* $\mathcal{S}_{init} \subseteq \mathcal{S}$ *is a set of initial states;*
- $\mathbf{P} : \mathcal{S} \times \mathcal{S} \to [0, 1]$ *is a transition probability matrix, where, for all states* $s \in \mathcal{S}$*, we have* $\sum_{s' \in \mathcal{S}} \mathbf{P}(s, s') = 1$*;*

- *$\mathcal{AP}$ is a set of atomic propositions and $\mathcal{L} : \mathcal{S} \rightarrow 2^{\mathcal{AP}}$ is a labelling function giving the set of propositions from $\mathcal{AP}$ that are true in each state.*

For each pair s, s' of states, $\mathbf{P}(s, s')$ represents the probability of going from s to s'. If there is no outgoing transition from s_i to s_j, $\mathbf{P}(s_i, s_j) = 0$. If $\mathbf{P}(s, s') > 0$, then s is a predecessor of s' and s' is a successor of s. A state $s \in \mathcal{S}$ is called absorbing when $\mathbf{P}(s, s) = 1$. For a state s and set $C \subseteq \mathcal{S}$, we will often use the notation $\mathbf{P}(s, C) := \sum_{s' \in C} \mathbf{P}(s, s')$.

A path σ of a DTMC $\mathcal{D}$ is a finite or infinite sequence of states $\sigma = s_0 s_1 s_2 \ldots$ such that $\forall i \geqslant 0,\ s_i \in \mathcal{S}$ and $\mathbf{P}(s_i, s_{i+1}) > 0$, where $s_0 \in \mathcal{S}_{init}$. The i^{th} state of the path σ is denoted by $\sigma[i]$. We let $Path^{\mathcal{D}}(s)$ denote the set of infinite paths of $\mathcal{D}$ that begin in a state s. To reason formally about the behaviour of a DTMC, we define a probability measure Pr_s over the set of infinite paths $Path^{\mathcal{D}}(s)$ [57]. We usually consider the behaviour from some initial state $s \in \mathcal{S}_{init}$ of $\mathcal{D}$.

DTMCs can also be augmented with *cost structures*, which attach non-negative costs to transitions [64]. In this thesis, we assume that these costs are always integers. Formally, a cost structure is a function $C : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{N}$, where $C(s, s') > 0$ implies $\mathbf{P}(s, s') > 0$. Formally, for a DTMC $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$, a reward structure is defined as a pair (ϱ, ι), where they represent state and transition rewards, respectively. The state rewards are assigned to states by using the reward function $\varrho : \mathcal{S} \rightarrow \mathbb{R}_{\geqslant 0}$, which is a vector. The state reward $\varrho(s)$ is the reward received when a DTMC is in the state s for one time step. Meanwhile, the transition rewards are assigned to transitions by using the rewards function $\iota : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}_{\geqslant 0}$, which is a matrix. The transition reward $\iota(s, s')$ is the reward collected when a transition occurs between states s and s'.

2.2 Probabilistic Computation Tree Logic

Properties of probabilistic models can be expressed using *Probabilistic Computation Tree Logic* (PCTL) [43] which extends Computation Tree Logic (CTL) with time and probabilities. In PCTL, state formulae ϕ are interpreted over states of a DTMC and path formulae ψ are interpreted over paths.

Definition 2.2.1 (PCTL) *The syntax of PCTL is as follows:*

$$\begin{aligned}\phi &::= \text{true} \,\big|\, a \,\big|\, \neg\phi \,\big|\, \phi \wedge \phi \,\big|\, \mathtt{P}_{\bowtie p}[\,\psi\,] \\ \psi &::= \phi_1 \,\mathtt{U}^{\leqslant k}\, \phi_2\end{aligned}$$

where a is an atomic proposition, $\bowtie \in \{<, \leqslant, \geqslant, >\}$, $p \in [0,1]$ and $k \in \mathbb{N} \cup \{\infty\}$.

The main operator in PCTL, in addition to those that are standard from propositional logic, is the probabilistic operator $\mathtt{P}_{\bowtie p}[\psi]$, which means that the probability measure of all the paths that satisfy ψ is within the bound $\bowtie p$. For path formulae ψ, we allow the (bounded) until operator $\phi_1 \,\mathtt{U}^{\leqslant k}\, \phi_2$. If ϕ_2 becomes true within k time steps and ϕ_1 is true until that point, then $\phi_1 \,\mathtt{U}^{\leqslant k}\, \phi_2$ is true. In the case where k equals ∞, the bounded until operator becomes the unbounded until operator and is denoted by $\mathtt{U}$. For simplicity of presentation, in this paper, we omit the $(\mathtt{X}\,\phi)$ operator, but this could easily be added.

Definition 2.2.2 (PCTL semantics) *Let $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$ be a DTMC. The satisfaction relation $\models_{\mathcal{D}}$ for PCTL formulae on $\mathcal{D}$ is defined by:*

- $s \models_{\mathcal{D}} \text{true}$ $\quad \forall s \in \mathcal{S}$
- $s \models_{\mathcal{D}} a$ $\quad$ *iff* $a \in \mathcal{L}(s)$
- $s \models_{\mathcal{D}} \neg\phi$ $\quad$ *iff* $s \not\models_{\mathcal{D}} \phi$
- $s \models_{\mathcal{D}} \phi_1 \wedge \phi_2$ $\quad$ *iff* $s \models_{\mathcal{D}} \phi_1$ *and* $s \models_{\mathcal{D}} \phi_2$
- $s \models_{\mathcal{D}} \mathtt{P}_{\bowtie p}[\psi]$ $\quad$ *iff* $Pr_s\{\sigma \in Path^{\mathcal{D}}(s) \mid \sigma \models_{\mathcal{D}} \psi\} \bowtie p$
- $\sigma \models_{\mathcal{D}} \phi_1 \,\mathtt{U}^{\leqslant k}\, \phi_2$ $\quad$ *iff* $\exists i \in \mathbb{N}.(i \leqslant k \wedge \sigma[i] \models_{\mathcal{D}} \phi_2 \wedge (\forall j. 0 \leqslant j < i. \sigma[j] \models_{\mathcal{D}} \phi_1))$

For example, a PCTL formula $\mathtt{P}_{<0.01}[\neg \mathit{fail}_1 \, \mathtt{U}^{\leqslant k} \, \mathit{fail}_2]$ can be interpreted as the probability of type 2 failure occurring within k time-steps, given that the type 1 failure does not occur until type 2 failure has taken place, is less than 0.01. Common derived operators are $\mathtt{F}\,\phi \equiv \mathrm{true}\,\mathtt{U}\,\phi$, which means that ϕ eventually becomes true, and $\mathtt{F}^{\leqslant k}\,\phi \equiv \mathrm{true}\,\mathtt{U}^{\leqslant k}\,\phi$, which means that ϕ becomes true within k steps.

The reward-based properties of DTMC can be expressed by extending the logic PCTL with additional operators [63].

$$\mathbf{R}_{\bowtie r}[\mathrm{C}^{\leqslant k}] \mid \mathbf{R}_{\bowtie r}[\mathrm{I}^{=k}] \mid \mathbf{R}_{\bowtie r}[\mathrm{F}\phi] \mid \mathbf{R}_{\bowtie r}[\mathrm{S}]$$

where $r \in \mathbb{R}_{\geqslant 0}$, $k \in \mathbb{N}$, $\bowtie$ holds the same definition as before and ϕ is a PCTL formula. The four reward-based operators given above respectively refer to the reward cumulated over k time steps, the state reward at time instant k, the reward cumulated before a state satisfying ϕ and the long-run (steady-state) rate of reward accumulation. However, in this thesis, we will make use of *cost-bounded* properties of the form $\mathtt{P}_{\bowtie p}[\mathtt{F}_{\leqslant c}\ \phi]$, which state that the probability of reaching a state satisfying ϕ and whilst accumulating a cost of at most c satisfies $\bowtie p$. Logics and model checking algorithms for these types of properties can be found in [3].

2.3 Probabilistic Reachability

Probabilistic reachability is the most fundamental property considered for probabilistic models, and is the main computational task required to perform PCTL model checking. In particular, we need to compute *reachability probabilities*, i.e. the probability, from some state s of DTMC $\mathcal{D}$, of reaching a set of states $\mathsf{target} \subseteq \mathcal{S}$, or the *step-bounded reachability probabilities*, i.e. the probability of reaching target within k steps.

For the former, reachability probabilities are computed as follows. We first divide all states into three disjoint sets: $\mathcal{S}^{yes}, \mathcal{S}^{no}, \mathcal{S}^{?}$. The set $\mathcal{S}^{yes}$ represents all the states that has the probability value equal to one; this includes all $s \in \mathsf{target}$. The set $\mathcal{S}^{no}$ contains all the states that cannot reach any $s \in \mathsf{target}$. The set $\mathcal{S}^{?}$ represents all the states $s \in \mathcal{S} \setminus \mathcal{S}^{yes} \cup \mathcal{S}^{no}$. The graph traversal algorithm presented in [38] can be used to identify the sets $\mathcal{S}^{yes}$ and $\mathcal{S}^{no}$. The reachability probability for set $\mathcal{S}^{?}$ can be computed by solving the linear equation system in variables x_s, where $s \in \mathcal{S}$.

$$x_s = \begin{cases} 1 & \text{if } s \in \mathcal{S}^{yes} \\ 0 & \text{if } s \in \mathcal{S}^{no} \\ \sum\limits_{s' \in \mathcal{S}} \mathsf{P}(s, s') \cdot x_{s'} & \text{if } s \in \mathcal{S}^{?} \end{cases}$$

For the case of step-bounded reachability probabilities, we can compute the probability for k steps as the value x_s^k inductively:

$$x_s^k = \begin{cases} 1 & \text{if } s \in \mathsf{target} \\ 0 & \text{if } s \notin \mathsf{target} \text{ and } k = 0 \\ \sum\limits_{s' \in \mathcal{S}} \mathsf{P}(s, s') \cdot x_{s'}^{k-1} & \text{otherwise} \end{cases}$$

2.4 Probabilistic Bisimulation

Larsen and Skou [67] defined (strong) *probabilistic bisimulation* for discrete probabilistic transition systems, which is an equivalence relation used to identify states with identical labellings and (probabilistic) step-wise behaviour.

Definition 2.4.1 (Probabilistic bisimulation) *Let $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$ be a DTMC and $\mathcal{R}$ an equivalence relation on $\mathcal{S}$. Then $\mathcal{R}$ is a (strong) probabilistic bisimulation on*

$\mathcal{D}$ if, for $(s_1, s_2) \in \mathcal{R}$:

$$\textit{(i) } \mathcal{L}(s_1) = \mathcal{L}(s_2) \;\; \textit{and (ii) for all } C \in \mathcal{S}/\mathcal{R} : \;\mathbf{P}(s_1, C) = \mathbf{P}(s_2, C)$$

where $\mathcal{S}/\mathcal{R}$ denotes the set of equivalence classes of set $\mathcal{S}$ by relation $\mathcal{R}$. States s_1, s_2 are bisimilar if there exists a bisimulation on $\mathcal{D}$ containing (s_1, s_2).

Two states that are probabilistically bisimilar will satisfy the same properties, including both infinite-horizon (long-run) and finite-horizon (transient) properties. Aziz et al. [5] proved that any property in the temporal logic PCTL is also preserved in this manner. Thanks to these results, the analysis of the original Markov chain, such as probabilistic model checking of PCTL, can be equivalently performed on the *quotient* Markov chain, in which equivalence classes of bisimilar states are lumped together into a single state.

Usually, we are interested in the coarsest possible probabilistic bisimulation for a DTMC $\mathcal{D}$ (or, in other words, the union of all possible bisimulation relations). We denote the coarsest possible probabilistic bisimulation by $\sim$. The quotient model $\mathcal{D}/\!\sim$ derived using this relation is defined as follows.

Definition 2.4.2 (Quotient DTMC) *Given DTMC $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$, the quotient DTMC is defined as $\mathcal{D}/\!\sim\; = (\mathcal{S}', \mathcal{S}'_{init}, \mathbf{P}', \mathcal{AP}, \mathcal{L}')$ where:*

- $\mathcal{S}' = \mathcal{S}/\!\sim\; = \{[s]_\sim \mid s \in \mathcal{S}\}$
- $\mathcal{S}'_{init} = \{[s]_\sim \mid s \in \mathcal{S}_{init}\}$
- $\mathbf{P}'([s]_\sim, [s']_\sim) = \mathbf{P}(s, [s']_\sim)$
- $\mathcal{L}'([s]_\sim) = \mathcal{L}(s)$

and $[s]_\sim$ denotes the unique equivalence class of relation $\sim$ containing s.

2.5 The PRISM Modelling language

The PRISM modelling language is a simple, state-based language based on the Reactive Modules formalism of Alur and Henzinger [1]. In this section, we briefly explain the relevant components of the language using a simple PRISM model description. Details about this language can be found in [25].

The core components of the PRISM language are modules and variables. A variable can either be boolean or integer and can be defined locally or globally. A PRISM model is made up of one or more modules and they can interact with each other. A module contains a number of variables. These variables are local to the module and their values depend on the state of the module at any given time.

The behaviour of each module is described by a set of commands. A command takes the form:

$$[]\ g \rightarrow p_1 : u_1 + ... + p_n : u_n;$$

The guard g is a predicate over all the variables in the model . An update u_i describes a transition which the module can make if the guard is satisfied. The expression p_i is used to attach the probabilities to their corresponding transitions. A simple DTMC model described using the PRISM language is shown in Figure 2.1. This model includes only one module called *example*. In this model, N is defined as a global integer constant whereas the variable x is defined as a local integer. The upper bound of the variable x is defined using N. The command shown in line 7 can be interpreted as follows: the model is allowed to make two different transitions when the value of the variable x is less than $(N - 1)$. The first transition of this command is represented by the update that increments the x by one with the probability of 0.3. The second update can be interpreted in a similar way.

```
dtmc

const int N;

module example
x : [0..N];
[] x < N − 1      → 0.3 : (x′=x + 1) + 0.7 : (x′=x + 2);
[] x = N          → (x′=0);
endmodule
```

Figure 2.1: The PRISM model description with single module

2.6 Recurrence relations

The nature of a recurrence relation can be classified based on the following properties: order of the relation, homogeneous or non-homogeneous, linear or non-linear and constant coefficients or not. In relation to Chapter 5, we are only interested in the first-order linear recurrence relations with constant coefficients, which can be either homogeneous or non-homogeneous. We formally define the relevant recurrence relations below.

Definition 2.6.1 (Linear Recurrence Relation) *A sequence a_n (for $n \geqslant 0$) satisfies a linear recurrence relations of order k with coefficients $c_1(n), c_2(n), \cdots, c_k(n)$ if*

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + \alpha(n),\ c_k \neq 0, n \geqslant k$$

where $\alpha(n)$ is the particularity function of n. This linear recurrence relation is called homogenous *when $\alpha(n) = 0$ for all $n \geqslant k$ and said to have constant coefficients when $c_1(n), c_2(n), \cdots, c_k(n)$ are constants.*

Definition 2.6.2 (Linear Homogeneous Recurrence Relation) *Homogenous linear recurrence relations of order k with coefficients can also be written in the following form,*

$$c_0 a_n + c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} = 0,\ n \geqslant k$$

where $c_0 \neq 0$ and $c_k \neq 0$. The characteristic equation*, which is solved to find a matrix's eigenvalues, of the corresponding recurrence relation is a polynomial equation of the form*

$$c_0 a^k + c_1 a^{k-1} + \cdots + c_{k-1} a + c_k = 0$$

where the polynomial on the left hand side is called the characteristic polynomial*. The solutions of this characteristic equation are called the characteristic roots of the respective recurrence relation.*

2.7 Generating Functions

Generating functions are very useful as they transform the problems of sequences into problems of functions. These generating functions can be used to solve a large class of recurrence relations as there is a relationship between the denominator of a generating function and a recurrence relation which define the same series. Any infinite sequence $a_0, a_1, \cdots$ can be encoded as a formal infinite power series

$$a_0 x^0 + a_1 x + a_2 x^2 + \cdots$$

where the elements of the sequence are treated as the coefficients of a series expansion. The sum of this infinite series is called the generating function. There are many types of generating functions; we are interested in *ordinary* generating functions. The definition of an ordinary generating function is given below.

Definition 2.7.1 ((Ordinary) Generating Function) *The ordinary function of a sequence a_n is given in the form of*

$$A(x) = \sum_{n=0}^{\infty} a_n x^n$$

where the respective sequence is treated as the coefficients of the formal power series called Maclaurin series [74].

CHAPTER 3

Review of Related Work

This chapter sets the scene for the subsequent chapters by presenting a literature review of the closely related work to this thesis. We start with mentioning about the model reduction techniques that are widely used in the context of model checking. Thereafter, we review the so called "bisimulation minimisation" technique in detail for both the non-probabilistic and probabilistic settings in Section 3.1. We also review the other relevant model reduction techniques in Section 3.2. Afterwards, the work closely related to parametric model checking is discussed briefly with respect to Chapter 5. Finally, we summarise the whole literature review with respect to this thesis.

The state space explosion problem [79], which refers to the size of a state space of a system growing exponentially in the number of its processes and variables, has drawn the attention of many researchers in the field of model checking. As a result of this, numerous novel *model reduction* algorithms have been introduced to mitigate the impact of this problem over the past years. Among them, bisimulation minimisation [29, 52], abstraction refinement[16, 54] and partial-order reduction [2] are three most widely used approaches. All of these algorithms have different characteristics. For example, the resulting reduced model of some algorithms preserves all the properties of the original model whereas others lose some information irrelevant to accomplish the goal of model checking [9].

This thesis considers the work that revolves around the bisimulation minimisation technique as the closely relevant literature to Chapter 4. Nevertheless, the literature about the approximate bisimulation and abstraction refinement are still connected to this chapter as the ultimate goal of them is also alleviating the impact of the state space explosion problem. Therefore, we will review the literature about the bisimulation minimisation in depth while the literature of the other two concepts is briefly reviewed in the following sections.

3.1 Bisimulation Minimisation

A labelled transition system (LTS) consists of a collection of states and a collection of transitions between them [53]. In the context of model checking, the notion of *bisimulation* is an equivalence relation between states in a LTS. States in an LTS are equivalent under bisimulation when the atomic propositions of interest and the transitions to other classes of equivalent states are the same. In the process of *bisimulation minimisation*, equivalent states are merged, i.e. replaced by a single state that exhibits the same behaviour, and the coarsest model is obtained at the end. The notion of bisimulation was initially applied only to non-probabilistic systems. Later, this notion was adapted to apply to probabilistic systems as well. Thus, we will review the literature with regards to non-probabilistic systems in the upcoming subsection which will be immediately followed by the literature of bisimulation for probabilistic systems.

3.1.1 Non-Probabilistic Bisimulation

The notion of bisimulation was introduced by Park [73] and Milner [70] in the context of concurrency theory. Later, Kanellakis and Smolka have successfully used this notion of bisimulation to combat the so called state space explosion problem [51]. The computation of bisimulation minimisation can be processed using two different strategies. One is a *positive strategy* in which the process starts with the finest partition and gradually the quotient model is constructed by merging bisimilar classes. Another is a *negative strategy* in which the processes starts with the coarsest partition and step by step it splits classes until the partition becomes *stable*, i.e when there are no further possible refinements. In [46], Hopcroft has proposed an algorithm for minimising the number of states in a finite automaton and also for determining the equivalence of two automata. This algorithm uses the negative strategy to obtain the quotient model. The time and space complexity of the presented algorithm in [46] is bounded by respectively $O(n \log n)$ and $O(n)$ where n is the number of states.

Later, Kanellakis and Smolka have clearly drawn the boundary between the partition problem and the state space reduction of a finite automaton based on the difference of having a set that contains only states and having a set with states where each of them has many transitions [51]. Henceforth, they have generalised Hopcroft's work to develop an efficient partition refinement algorithm with time complexity $O(mn)$ for minimising the state space of a deterministic finite automaton. In this context, the number of observable state processes and the transitions of each state are denoted as n and m, respectively. The underlying idea of this algorithm is to iterate the splitting process on the initial partition using a splitter with respect to certain actions until there is no further possible refinement in the current partition. Paige et al. [72] have also studied a variant of the coarsest partition problem which was addressed in [46]. As a result of this study, they have successfully presented a new algorithm to solve this problem in $O(n)$ space and time.

In [51], Kanellakis and Smolka have also conjectured that there exists an algorithm that reduces the time complexity of the partition refinement algorithm from $O(mn)$ to $O(m \log n)$. Paige and Tarjan have proposed an algorithm in [71], that has further improved the time bounds of [51] and yielded a time complexity of $O(m \log(n) + n)$ for strong equivalence classes, i.e. the classes that contain states with same transitions to any other equivalence classes. This algorithm refines the larger equivalence classes into smaller classes, i.e. starting with an initial partition of set of states and refines it until no further refinement is possible.

Fernandez [33] has established a relationship between bisimulation equivalence and the relational coarsest partition problem which was solved by Paige and Tarjan in [71]. The process of identifying a suitable coarsest partition for a given initial partition and binary relation is considered to be a challenging task. Fernandez has stated that computing bisimulation equivalence can be seen either as an instance or a generalisation of this problem. He has adapted the algorithm presented in [71] to come up with a more efficient algorithm to minimise the labelled transition systems modulo bisimulation equivalence. Similarly, many other researchers [32, 10, 68, 29] have adapted and optimised the algorithm proposed in [71] that improves the solution to the relational coarsest partition problem.

A state s is reachable when $\mathbf{P}(s_0, s) > 0$, where $s_0 \in \mathcal{S}_{init}$. A block, which is a set of states,is reachable when it contains at least one reachable state. Reachable blocks may also contain unreachable states. A block B_1 is stable with respect to block B_2 iff either all states in B_1 have transitions to states in B_2 or no states in B_1 has a transition to a state in B_2. In the case when B_1 is not stable with respect to B_2, B_2 is called a **splitter** of B_1. The bisimulation minimisation algorithm presented by Paige and Tarjan has stabilised both reachable and unreachable blocks. Afterwards, Bouajjani et al. [10] have improved it by choosing only reachable blocks during each iteration for stabilizing. Nevertheless, this still could stabilize an unreachable block that was separated from the reachable block being processed in the respective iteration.

Later, this problem was addressed by Lee and Yannakakis [68], who have completely avoided stabilising the unreachable blocks. In [36], Fisler and Vardi have studied the three bisimulation minimization algorithms presented in [71, 10, 68] and created a novel on-the-fly model checker (constructs the state space during the exploration) for invariant properties based on these algorithms. Later, they have compared their on the fly model checker with a traditional model checker that leveraged a backward reachability approach. As a result of this study, they have established close correlations between the set of states computed during the minimisation and those computed during invariant property verification through backward reachability. This correlation clearly shows that in the context of testing invariant properties minimisation and backward reachability are similar. In the context of non-probabilistic model checking, Fisler and Vardi have proven that the cost of bisimulation outweighs the model checking through comparing the minimum number of operations of various kinds and an experimental analysis on a suite of designs.

3.1.2 Probabilistic Bisimulation

The notion of bisimulation for discrete probabilistic transition systems was initially defined by Larsen and Skou [67]. *Probabilistic bisimulation* is an equivalence relation where any two related states have the same probability of making a transition to any equivalence class of states. In this case, the term *"minimisation"* is also referred to either as *"lumping"* or *"aggregation"*. Aziz et al. [5] have proved that the logic PCTL is expressive with respect to probabilistic bisimulation equivalence on Markov chains by developing a notion of bisimulation for Markov processes.

In [8], Baier et al. have presented an algorithm for bisimulation equivalence in probabilistic labelled transitions systems. This algorithm has runtime complexity of $O(mn(\log m + \log n))$ and is considered to be efficient enough to be used in verification tools. By using this algorithm, verifications tools do not have to analyse the whole system, instead they analyse only the quotient state space of the respective system. In contrast to non-probabilistic setting, bisimulation minimisation may help probabilistic model checking to cope more with the state space explosion problem.

Later, Cattani and Segala [12] have defined a splitter for both *weak* and *strong* probabilistic bisimulation relations based on [71], where weak bisimulation is a bisimulation with possibly unobservable actions interspersed. Although the authors have obtained a polynomial time algorithm for strong probabilistic bisimulation, they only managed to produce a exponential time algorithm for the weak probabilistic bisimulation (for more detail see [7]).

However, both Baier et al. [8] and, Cattani and Segala [12] could not achieve an efficient algorithm as Paige and Tarjan have attained in non-probabilistic bisimulation. Derisavi et al. [29] have proved that the optimal lumping quotient of a finite Markov chain can be constructed in $O(m \log n)$ time by using statically optimal trees (e.g., splay trees [75]), for n state observable processes and m transitions. Like any other algorithm, this one is also based on the splitting technique of Paige and Tarjan to compute bisimilarity of labelled transition systems. Their research yielded that using the other balanced binary search trees result in the worst case running time of the algorithm, which is $O(m \log^2 n)$. On the other hand, when they used splay trees, they have managed to take the $O(\log n)$ factor out of the time complexity from the previously obtained results with help of the static optimality property of splay trees. Bisimulation minimisation techniques can be grouped into two types. They are:

1) state-level minimisation

2) model-level minimisation

State level lumping technique exploits the lumping properties of the given model and generates the the smallest quotient model (e.g. [29],[80]). On the other hand, model level lumping technique identifies the necessary lumping properties by analysing the higher level formalism and constructs the quotient model directly instead of building the original model and performing the lumping process on it (e.g. [62]). Unfortunately, this technique cannot always find the smallest possible quotient model because this is limited only to the properties that can be identified from the given model description.

Derisavi has implemented two variants of the state level lumping algorithm; one of them uses splay trees while the other one uses red-black trees to represent sub block trees. Although the splay tree variant is proven to be theoretically faster [29], the results of the experiments in [28] show that, in practice and for virtually all cases, the red-black tree variant is 10% faster comparing to the splay tree variant. The state level lumping algorithm presented in [29] with $O(m \log n)$ time complexity is the fastest known algorithm in the context of probabilistic setting. They have also proved a lower bound of $O(m + n \log n)$ on the running time of any state level lumping algorithm. There is a noticeable gap between these two time complexities.

Derisavi's algorithm consists of three general phases. During the first phase, the necessary variables for the bisimulation process will be initialised. Then as a second (main) phase of the algorithm, the refinement of the original partition will take place to produce the coarsest ordinary lumping partition. Finally, the quotient Markov chain will be constructed from the coarsest partition. These three general phases can be reused to construct a new algorithm to perform bisimulation minimisation on probabilistic models. The algorithm presented in [80] also uses these general phases to achieve the goal of bisimulation minimisation.

Derisavi et al. have also conjectured that the time complexity $O(mlog(n))$ could be achieved using a simpler solution than splay trees. In other words, the proposed algorithm for the Markov chain lumping perhaps needs an efficient sorting algorithm for weights. In [80], Valmari and Franceschinis have presented an algorithm that sorts the weights with a combination of so called possible majority algorithm and any $O(k \log k)$ sorting algorithm, where k is the number of items to be sorted. Also they have pointed out an essential issue in the description of the algorithm presented in [29], i.e. if a block is used as a splitter, and then itself split into sub blocks, then it is enough to use all of them as a potential splitter except for one (the largest block will not be used). In the case that the main block is not a splitter then every resulting block must be used as a splitter. The MRMC model checker [56] implements the time-optimal partition refinement algorithm presented in [29]. In this implementation, they have replaced the splay tree with a *heapsort* data structure which has approximately the same performance as the splay tree implementation.

Bisimulation can be performed in two different perspectives, which are forward and backward stochastic bisimulation. Forward stochastic bisimulation determines the equivalence of states by identifying the relation between their *outgoing* transitions. But backwards stochastic bisimulation determines the equivalence between these states by looking at the relations between their *incoming* transitions. The benefit of the latter over forward stochastic bisimulation is: states that reside under an equivalence class have an equal probability, for both the transient and steady-state distributions and for a certain condition on the initial distribution of the concrete system. Thus, both the transient and steady-state probabilities for the concrete system can be computed from the backward stochastic bisimulation quotient. Sproston and Donatelli [77] have presented a study of backward stochastic bisimulation in the context of model checking for continuous-time Markov chains (CTMC) against continuous stochastic logic properties. In this study, it is proven that backward stochastic bisimulation can outperform forward stochastic bisimulation. The property that they have used to reduce the state space of the concrete system is that the states within a equivalence class either should satisfy a temporal logic formula or not. Therefore, it enables to reason at the level of equivalence classes rather than individual states, since these equivalence classes allows to determine the set of states which satisfy a particular formulas.

All of these bisimulation minimisation algorithms require the exploration of the whole state space, which is a stumbling block in this approach. However, in [25] Dehnert et al. have introduced a completely different technique which is directly extracting the bisimulation quotient from a high-level description using satisfiability modulo theories (SMT) solvers [20, 22]. The main focus of this approach is applying the partition refinement bisimulation minimisation on a probabilistic system, which is described using the PRISM language [59] that consists of guarded commands, in a truly symbolic way while avoiding the generation of *whole state space*.

3.2 Other Model Reduction Techniques

In the probabilistic context, the notion of bisimulation can be too restrictive and sensitive when considering processes with approximately identical behaviour, i.e. processes that differ only by a small value ε. It is well-known that the probabilities computed for probabilistic models are often approximate estimations from experiments. Thus, this notion is not always robust in the case of probabilistic models. A small perturbation in the estimation of probabilities can end up resulting two bisimilar states into non-bisimilar states. Thus, the notion of equivalence for stochastic processes is found to be sometimes not suitable to be used in practice. In [39], Giacalone et al. have pointed out that the notion of distance between probabilistic processes is more suitable in practice comparing to the notion of equivalence. Thus, the study of the notion of *approximate bisimulation* has drawn attention in the probabilistic context.

Dean et al. have addressed this problem for the related model of Markov Decision Process (MDP) by introducing a property of state space partitions which is called ε-homogeneity [23]. An ε-homogeneous partition comprises states such that their behaviours are approximately similar under all or some subset of policies. In other words, the states that reside in the same block can have transitions to other blocks with different probabilities, where the difference in probabilities should be less than or equal to ε. Generally

ε-homogeneous partitions are smaller and sometimes much smaller than the actual smallest homogeneous partition. The algorithm presented in [23] allows us to reduce a larger factored MDP into a possibly much smaller *"bounded parameter"* MDP (it is a family of traditional MDPs defined by specifying upper and lower bounds on the transition probabilities and rewards) with the help of homogeneous partitions and provides methods to select actions from the *"bounded parameter"* MDPs, where these methods enable the analysis of the concrete MDPs. Later, Ferns et al. [34] have presented metrics for measuring the similarity of states in a finite MDP, based on the notion of bisimulation for MDPs, with the main focus of addressing the same problem. These metrics allow to aggregate states in the same way they are aggregated in an equivalence relation. In this approach, the states will be clustered together when they are in the same α-neighbourhood, where α is a tolerance parameter.

Chen et al. [14] have proved that the probabilistic bisimilarity pseudometrics can be computed exactly in polynomial time for labelled Markov chains using the ellipsoid algorithm, where they consider pseudometrics as a solution of a linear program. However, the ellipsoid algorithm is considered to be inefficient in practice. In [6], Bacci et al. have proposed an algorithm for exact computation of bisimilarity distances between DTMCs which was introduced in [31]. In this approach, the distance between given states is computed exactly and the exhaustive state space exploration is avoided. This on-the-fly algorithm addresses the problem in [14] by using a greedy strategy. Successively, it refines the over-approximations of the target distances, so that it explores the state space further only when the current approximations are improved.

Abstraction of a concrete model intentionally leaves out some details that are not relevant to the objective of the abstraction, i.e. to the property that is to be verified. *Refinement* is a process that refines an abstraction by adding more detail to it and produces a new abstraction. A typical *abstraction-refinement* process works in the following way. As a first step, the abstract model of the concrete system will be automatically constructed based on the chosen partition of the state space. Afterwards, the constructed abstract model will be checked for the desired property and if the abstract model is error-free then so is the concrete system. If not, an abstract counter-example, that shows the violation of the property, will be produced. Finally, it will be checked whether a corresponding concrete counter-example exists in the original system for the produced abstract counter-example. If one exists, then an error has been found on the original system. Otherwise, more detailed descriptions will be added to the previous abstraction since it does not provide enough information to verify the given property.

The application of abstraction refinement varies for DTMCs and CTMCs. An abstraction can be obtained for DTMCs by replacing the transition probabilities within a class where upper and lower bounds serve as respectively upper and lower approximations. In the case of continuous-time setting, abstraction is performed on a uniform CTMC, where the uniform CTMC is derived from a general CTMC. During the abstraction, the probabilistic transitions will be replaced by intervals. Hence, the model checking can be reduced to verify time-bounded reachability probabilities in continous-time MDPs. The semantics for the abstractions were presented as two-valued semantics [16] and three valued semantics [54]. Recently, a novel abstraction technique based on Erlang's method of stages for CTMCs was proposed in [55], improves the work presented in [54] by persevering a simulation relation on CTMCs.

3.3 Parametric and Incremental Model Checking

In *parametric* model checking, the model checking process depends on *parameters* from the high-level model description (e.g. the PRISM language). The parametric model checking verifies whether a property holds for different values of the parameters. In [19], Daws has first proposed a language-theoretic approach to symbolic probabilistic model checking of PCTL over DTMCs. This approach is based on the conversion of the DTMC to a finite automaton (where the alphabet represents the set of strictly positive transition probabilities), from which a regular expression is derived using the technique called *state elimination* [47]. Later, the regular expression is evaluated to a rational function over the unspecified parameters which defines the probability of reaching the target.

However, Gruber and Johannsen have shown that the conversion of deterministic finite automata accepting finite languages into regular expressions explodes in size of $n^{\Theta(logn)}$, where n denotes the number of states. In practice, the numerical values are largely simplified in the process, thus the length of the expression mostly relies on the number of parameters. In [41], Hahn et al. have provided an improved algorithm that intertwines the state elimination and the computation of the rational functions. As a result, they have avoided the computation of regular expressions; thus, managed to stay within the domain of Markov chains throughout the process. The authors have also presented a strategy that reduces the state space before the main algorithm by applying bisimulation minimisation technique. These techniques were implemented in the model checking tools PARAM [40] and PRISM [59]. A more recent tool, PROPhESY [24] includes further optimisations (e.g. using decomposition into strongly connected components [49]); henceforth, it outperforms the state-of-the-art tools.

An approach to efficiently verifying models with repeated structure is *incremental model checking*, where model checking is performed many times, e.g. over a range of model parameter values, but each run of model checking re-uses results from previous runs. The first incremental algorithm in the context of non-probabilistic systems was presented in [76]. Later, various other incremental techniques have been proposed, e.g. [45, 18, 44], in the same context. In probabilistic verification, the transition probabilities varies due to changes in the real system over time. Thus, the incremental verification techniques are necessary to improve the efficiency of probabilistic verification.

In [66], Kwiatkowska et al. have presented efficient incremental techniques for quantitative verification of MDPs. These techniques divides the state space into strongly connected components and performs verification on them individually. If a change occurs in the transition probabilities, verification is performed only on the affected components and the old results of the unaffected components are reused. This way the efficiency of probabilistic verification is significantly improved. Later, the authors have reported further improvement on these techniques in [61]. Unlike [66], the techniques presented in [37] considers the changes at modelling language description level and allows changes in the model structure.

3.4 Summary

Our work on finite-horizon bisimulation minimisation from Chapter 4 is slightly different from the techniques that we discussed in the literature. The signature-based bisimulation approach computes a fingerprint for each state such that states can only be bisimilar if they have identical fingerprints. Our first partition-refinement algorithm adapts the signature-based approach, which has been studied in [29, 83]. The SMT-based bisimulation minimisation technique proposed in [25] is also relevant which, like our on-the-fly algorithm, avoids the construction of the full model when minimising. Our SMT-based algorithm has an additional benefit in that it works on model descriptions with state-dependent probabilities. Other probabilistic verification methods have been developed based on backwards traversal of a model, for example for probabilistic timed automata [60], but this is for a different class of models and does not perform minimisation. Della Penna et al. considered finite-horizon verification of Markov chains [26], but using disk-based methods, not model reduction.

The techniques in Chapter 5 can be seen as a form of *parametric model checking*. Most work on parametric model checking focuses on the case where the underlying graph structure of the Markov chains is fixed, and the parameters are used to define the probabilities attached to the transitions. Most subsequent work in this area, similar to Daws [19], focuses on the probabilistic reachability problem for DTMCs, as also tackled in Chapter 5. Following a similar motivation to the techniques we propose in Chapter 5, parametric model checking has been successfully used to improve the efficiency of probabilistic model checking in scenarios where it needs to be applied repeatedly with different parameter values, for example in *run-time* verification contexts [35]. These methods all assume that parameters are used only to define transition probabilities, which differs from our approach in Chapter 5, where parameters control the size of an inductively defined model.

Chapter 5 also considers the case where the parameter represents a time-bound in a bounded probabilistic reachability query. The parametric model checking techniques presented in [42, 11, 13] are more relevant as they also work on time-bounded properties, using either discretisation or iterative division techniques. However, these all work on the alternative model of CTMCs, where the problem of computing time-bounded reachability probabilities is very different to the case of DTMCs.

The basic idea behind the methods in Chapter 5 is to exploit the structure of DTMCs that are defined inductively, and where a fragment of the model is repeated multiple times. Similar ideas are often used to define infinite-state models, for example quasi-birth death (QBD) models or matrix-geometric models. Model checking techniques for the former were proposed in [58], by constructing finite abstractions to solve time-bounded properties. However, the underlying model is again different (a CTMC) and the approach is quite varied, based on building abstractions with interval Markov chains, rather than the recurrence relation approach we defined in Chapter 5. Most relevant to the work in this thesis is the idea of *incremental model construction*. The methods proposed in [37, 78], like our work, look for repeated structure in a model described in the PRISM modelling language. However, they focus on MDPs, which is a more general model than

DTMCs, and do not construct a *reduced* model, like in Chapter 5.

CHAPTER 4

Finite-Horizon Bisimulation Minimisation

As discussed earlier, a widely used approach to combat the state space explosion problem is *probabilistic bisimulation* [67], an equivalence relation over the states of a probabilistic model which can be used to construct a smaller *quotient* model that is equivalent to the original one (in the sense that it preserves key properties of interest to be verified). Typically, it preserves both infinite-horizon properties, e.g., "the probability of eventually reaching an error state", finite-horizon (transient, or time-bounded) properties, e.g. "the probability of an error occurring within k time-steps", and, more generally, any property expressible in an appropriate temporal logic such as PCTL [43]. It has been shown that, in contrast to non-probabilistic verification, the effort required to perform bisimulation minimisation can pay off in terms of the total time required for verification [52].

In this chapter, we consider model reduction techniques for finite-horizon properties of Markov chains. We propose a *finite-horizon* variant of probabilistic bisimulation, which preserves stepwise behaviour over a finite number of steps, rather than indefinitely, as in standard probabilistic bisimulation. This permits a more aggressive model reduction, but still preserves satisfaction of PCTL formulae of bounded depth (i.e., whose interpretation requires only a bounded exploration of the model). Time-bounded properties are commonly used in probabilistic verification, e.g., for efficiency ("the probability of task completion within k steps") or for reliabilty ("the probability of an error occurring within time k").

We formalise finite-horizon probabilistic bisimulation, define the subset of PCTL that it preserves and then give a partition-refinement based algorithm for computing the coarsest possible finite-horizon bisimulation relation, along with a corresponding quotient model. The basic algorithm is limited by the fact it requires the full Markov chain to be constructed before it is minimised, which can be a bottleneck. So, we then develop on-the-fly approaches, which construct the quotient model directly from a high-level model description of the Markov chain, based on a backwards traversal of its state space. We propose two versions: one symbolic, based on SMT solvers, and one explicit-state.

We implemented all algorithms in PRISM and evaluated them on a range of examples. First, we apply the partition-refinement based approach to some standard benchmarks to investigate the size of the reduction that can be obtained in a finite-horizon setting. Then, we apply the on-the-fly approach to a class of problems to which it is particularly well suited: models with a large number of possible initial configurations, on which we ask questions such as "from which initial states does the probability of an error occurring within 10 seconds exceed 0.01?". We show that on-the-fly finite-horizon bisimulation can indeed provide significant gains in both verification time and scalability, demonstrated in each case by outperforming the existing efficient implementations in PRISM.

The remaining sections of this chapter are organised as follows. Section 4.1 outlines the mathematical definitions of this work. Section 4.2 discusses corresponding minimisation algorithms and their implementations in depth. Section 4.3 describes the on-the-fly finite-horizon bisimulation and its various implementations in the PRISM model checker. Section 4.4 describes the experiments carried out on both conventional and on-the-fly variants of the finite-horizon bisimulation and presents a discussion based on the results.

4.1 Finite-Horizon Bisimulation Preliminaries

We now formalise the notion of *finite-horizon bisimulation*, a step-bounded variant of standard probabilistic bisimulation for Markov chains [67], which is discussed in Section 2.4. We fix, from this point on, a DTMC $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$. Intuitively, a *$k$-step finite-horizon bisimulation*, for non-negative integer k, preserves the stepwise behaviour of $\mathcal{D}$ over a finite horizon of k steps. We use the following inductive definition.

Definition 4.1.1 (Finite-horizon bisimulation) *A* k-step finite-horizon bisimulation*, for $k \in \mathbb{N}_{\geqslant 0}$, is an equivalence relation $\mathcal{R}_k \subseteq \mathcal{S} \times \mathcal{S}$ such that, for all states $(s_1, s_2) \in \mathcal{R}_k$, the following two conditions are satisfied:*

(i) $\mathcal{L}(s_1) = \mathcal{L}(s_2)$*;*

(ii) $\mathbf{P}(s_1, C) = \mathbf{P}(s_2, C)$ *for each equivalence class $C \in \mathcal{S}/\mathcal{R}_{k-1}$,*

where $\mathcal{R}_{k-1}$ is a $(k-1)$-step finite-horizon bisimulation. A 0-step finite-horizon bisimulation *is an equivalence relation $\mathcal{R}_0$ satisfying only condition (i) above.*

Definition 4.1.2 (Finite-horizon bisimulation equivalent) *We say states s_1, s_2 are (k-step)* finite-horizon bisimulation equivalent *(*bisimilar*), denoted $s_1 \sim_k s_2$, if there exists a k-step finite-horizon bisimulation $\mathcal{R}_k$ such that $(s_1, s_2) \in \mathcal{R}_k$.*

Two states s_1 and s_2 satisfying $s_1 \sim_k s_2$ have the same stepwise behaviour over k steps. The following proposition gives a connection between the relation $\sim_k$ and standard probabilistic bisimulation relation (see 2.4), as well as some simple, but useful properties of $\sim_k$.

Proposition 4.1.1 *Let $s_1, s_2 \in \mathcal{S}$ be two states. Then:*

(a) if $s_1 \sim_k s_2$, then $s_1 \sim_j s_2$ for any $0 \leqslant j \leqslant k$.

(b) if $s_1 \sim s_2$, then $s_1 \sim_k s_2$ for any $k \geqslant 0$.

(c) if $s_1 \sim_k s_2$ and $s_1 \to s_1'$, then $s_1' \sim_{k-1} s_2'$ for some state s_2' such that $s_2 \to s_2'$.

From a model checking perspective, if $s_1 \sim_k s_2$, then s_1 and s_2 satisfy the same PCTL formulae up to a bounded depth k. We formalise this as follows.

Definition 4.1.3 (Formula depth) *The* depth *of a PCTL formula Φ, denoted $d(\Phi)$, is a value in $\mathbb{N} \cup \{\infty\}$ defined inductively as follows:*

- $d(\mathrm{true}) = d(a) = 0$ *for atomic proposition a;*
- $d(\neg\Phi) = d(\Phi)$*;*
- $d(\Phi_1 \wedge \Phi_2) = \max(d(\Phi_1), d(\Phi_2))$*;*
- $d(\mathtt{P}_{\bowtie p}[\Phi_1 \,\mathtt{U}^{\leqslant j}\, \Phi_2]) = j + \max(d(\Phi_1){-}1, d(\Phi_2))$*.*

For example, if a and b are atomic propositions, we have

- $d(\mathtt{P}_{\bowtie p}[\mathrm{true}\,\mathtt{U}^{\leqslant 5}\, a]) = 5$,
- $d(\mathtt{P}_{\bowtie p}[\mathrm{true}\,\mathtt{U}^{\leqslant 5}\, a] \wedge \mathtt{P}_{\bowtie p}[\mathrm{true}\,\mathtt{U}^{\leqslant 6}\, a]) = 6$
- $d(\mathtt{P}_{\bowtie p}[\mathrm{true}\,\mathtt{U}^{\leqslant 5}\, \mathtt{P}_{\bowtie p}[a\,\mathtt{U}^{\leqslant 3}\, b]]) = 8$

If states s_1 and s_2 are (k-step) finite-horizon bisimilar, then they satisfy exactly the same PCTL formulae of depth at most k, which we state formally as follows.

Theorem 4.1.1 *Let s_1 and s_2 be two states such that $s_1 \sim_k s_2$, and Φ be a PCTL formula with depth $d(\Phi) \leqslant k$, then $s_1 \models \Phi$ if and only if $s_2 \models \Phi$.*

Proof 4.1.1 *We prove the result by induction over the structure (see Definition 2.2.1) of PCTL formula Φ. Propositional operators are straightforward since s_1 and s_2 satisfy the* *same atomic propositions, by the definition of* $\sim_k$*, and, for* $\Phi = \neg\Phi_1$ *or* $\Phi = \Phi_1 \wedge \Phi_2$*, the subformulae* Φ_1 *and* Φ_2 *have depth at most* k *so, by induction, we can assume that* $s_1 \models \Phi_i \Leftrightarrow s_2 \models \Phi_i$ *for* $i \in \{1,2\}$*.*

The remaining case to consider is $\Phi = \mathtt{P}_{\bowtie p}[\Phi_1 \,\mathtt{U}^{\leqslant j}\, \Phi_2]$*. We know, from Definition 4.1.3, that the depths* $d(\Phi_1)$ *and* $d(\Phi_2)$ *of the two subformulae are at most* $k-j+1$ *and* $k-j$*. From the semantics of PCTL, we have that, for any state* s*:*

$$s \models \mathtt{P}_{\bowtie p}[\,\Phi_1 \,\mathtt{U}^{\leqslant j}\, \Phi_2\,] \;\Leftrightarrow\; Pr_s(\Phi_1 \,\mathtt{U}^{\leqslant j}\, \Phi_2) \bowtie p$$

which means it suffices to show that:

$$Pr_{s_1}(\Phi_1 \,\mathtt{U}^{\leqslant j}\, \Phi_2) = Pr_{s_2}(\Phi_1 \,\mathtt{U}^{\leqslant j}\, \Phi_2) \tag{1}$$

We in fact show this to be true for any states s_1, s_2*, values* $j \leqslant k$ *and PCTL subformulae* Φ_1, Φ_2 *satisfying* $s_1 \sim_k s_2$ *and* $\max(d(\Phi_1)-1, d(\Phi_2)) \leqslant k-j$*, which we prove inductively over* j*. From the model checking algorithm for PCTL [43], we know that, for any state* s*:*

$$Pr_s(\Phi_1 \,\mathtt{U}^{\leqslant j}\, \Phi_2) = \begin{cases} 1 & \textit{if } s \models \Phi_2 \\ 0 & \textit{if } s \models \neg\Phi_1 \wedge \neg\Phi_2 \\ 0 & \textit{if } s \models \Phi_1 \wedge \neg\Phi_2 \textit{ and } j = 0 \\ \sum_{s' \in \mathcal{S}} \mathbf{P}(s,s') Pr_{s'}(\Phi_1 \,\mathtt{U}^{\leqslant j-1}\, \Phi_2) & \textit{if } s \models \Phi_1 \wedge \neg\Phi_2 \textit{ and } j > 0. \end{cases}$$

For the base case $j = 0$*, only the first three cases of the definition above can apply, and we know that* $s_1 \models \Phi_i \Leftrightarrow s_2 \models \Phi_i$ *for* $i \in \{1,2\}$*, so we have that* $Pr_{s_1}(\Phi_1 \,\mathtt{U}^{\leqslant 0}\, \Phi_2) = Pr_{s_2}(\Phi_1 \,\mathtt{U}^{\leqslant 0}\, \Phi_2)$*. For the inductive case, where* $j > 0$*, we can assume that* $Pr_{s_1}(\Phi_1 \,\mathtt{U}^{\leqslant j-1}\, \Phi_2) = Pr_{s_2}(\Phi_1 \,\mathtt{U}^{\leqslant j-1}\, \Phi_2)$*, as long as* $s_1 \sim_{j-1} s_2$*. Considering again the possible cases in the above definition, the first two follow as for* $j = 0$ *and the third cannot apply since* $j > 0$*. For the fourth case, since* $j > 0$*, we know there exists a* $(j-1)$*-step finite-horizon bisimulation*

$\mathcal{R}_{j-1}$. Let us further assume an (arbitrary) function $rep : \mathcal{S}/\mathcal{R}_{j-1} \to \mathcal{S}$, which selects a unique representative from each equivalence class of $\mathcal{R}_{j-1}$. We have:

$$
\begin{array}{lll}
 & Pr_{s_1}(\Phi_1 \,\mathtt{U}^{\leqslant j}\, \Phi_2) & \\
= & \sum_{s' \in \mathcal{S}} \mathbf{P}(s_1, s') Pr_{s'}(\Phi_1 \,\mathtt{U}^{\leqslant j-1}\, \Phi_2) & \textit{by definition} \\
= & \sum_{C \in \mathcal{S}/\sim_{j-1}} \sum_{s' \in C} \mathbf{P}(s_1, s') Pr_{s'}(\Phi_1 \,\mathtt{U}^{\leqslant j-1}\, \Phi_2) & \textit{since } \sim_{j-1} \textit{ partitions } \mathcal{S} \\
= & \sum_{C \in \mathcal{S}/\sim_{j-1}} Pr_{rep(C)}(\Phi_1 \,\mathtt{U}^{\leqslant j-1}\, \Phi_2) \sum_{s' \in C} \mathbf{P}(s_1, s') & \textit{by induction on } j \\
= & \sum_{C \in \mathcal{S}/\sim_{j-1}} Pr_{rep(C)}(\Phi_1 \,\mathtt{U}^{\leqslant j-1}\, \Phi_2) \mathbf{P}(s_1, C) & \\
= & \sum_{C \in \mathcal{S}/\sim_{j-1}} Pr_{rep(C)}(\Phi_1 \,\mathtt{U}^{\leqslant j-1}\, \Phi_2) \mathbf{P}(s_2, C) & \textit{since } s_1 \sim_j s_2 \\
= & \sum_{C \in \mathcal{S}/\sim_{j-1}} Pr_{rep(C)}(\Phi_1 \,\mathtt{U}^{\leqslant j-1}\, \Phi_2) \sum_{s' \in C} \mathbf{P}(s_2, s') & \\
= & \sum_{C \in \mathcal{S}/\sim_{j-1}} \sum_{s' \in C} \mathbf{P}(s_2, s') Pr_{s'}(\Phi_1 \,\mathtt{U}^{\leqslant j-1}\, \Phi_2) & \textit{since } s' \sim_{j-1} rep(C) \\
= & \sum_{s' \in \mathcal{S}} \mathbf{P}(s_2, s') Pr_{s'}(\Phi_1 \,\mathtt{U}^{\leqslant j-1}\, \Phi_2) & \textit{since } \sim_{j-1} \textit{ partitions } \mathcal{S} \\
= & Pr_{s_2}(\Phi_1 \,\mathtt{U}^{\leqslant j}\, \Phi_2) & \textit{by definition}
\end{array}
$$

which proves (1), as required, and concludes the proof. □

In a similar fashion to the standard (non-finite-horizon) case, we are typically interested in the *coarsest possible* k-step finite-horizon bisimulation relation for a given DTMC (labelled with atomic propositions) and time horizon k, which we denote by $\sim_k$. We can also define this as the union of all possible k-step finite-horizon bisimulation relations. Furthermore, for $\sim_k$ (or any other finite-horizon bisimulation relation), we can define a corresponding *quotient* DTMC, whose states are formed from the equivalence classes of $\sim_k$, and whose k-step behaviour is identical to the original DTMC $\mathcal{D}$.

This is similar, but not identical, to the process of building the quotient Markov chain corresponding to a full minimisation (see Definition 2.4.2). We must take care since, unlike for full bisimulation, given a state $B \in \mathcal{S}/\sim_k$ of the quotient model, the probabilities $\mathbf{P}(s, B')$ of moving to other equivalence classes $B' \in \mathcal{S}/\sim_k$ can be different for each state $s \in B$ (according to the definition of $\sim_k$, probabilities are the same to go to

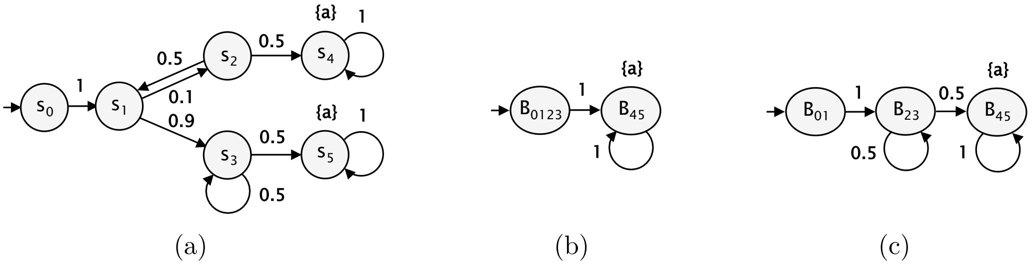


Figure 4.1: (a) Example DTMC; (b-c) Finite-horizon quotient DTMCs for $k = 0, 1$.

states with the same ($k-1$)-step, not k-step, behaviour). However, when they do differ, it suffices to pick an arbitrary representative from B. We formalise the quotient DTMC construction below, and then present some examples.

Definition 4.1.4 (Finite-horizon quotient DTMC) *If $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$ is a DTMC and $\sim_k$ is a finite-horizon bisimulation on $\mathcal{D}$, then a quotient DTMC can be constructed as $\mathcal{D}/{\sim_k} = (\mathcal{S}', \mathcal{S}'_{init}, \mathbf{P}', \mathcal{AP}, \mathcal{L}')$ where:*

- $\mathcal{S}' = \mathcal{S}/{\sim_k} = \{[s]_{\sim_k} \mid s \in \mathcal{S}\}$
- $\mathcal{S}'_{init} = \{[s]_{\sim_k} \mid s \in \mathcal{S}_{init}\}$
- $\mathbf{P}'(B, B') = \mathbf{P}(rep(B), B')$ *for any* $B, B' \in \mathcal{S}'$
- $\mathcal{L}'(B) = \mathcal{L}(rep(B))$ *for any* $B \in \mathcal{S}'$,

where $rep : \mathcal{S}/{\sim_k} \rightarrow \mathcal{S}$ is an arbitrary *function that selects a unique representative from each equivalence class of $\sim_k$, i.e., $B = [rep(B)]_{\sim_k}$ for all $B \in \mathcal{S}'$.*

Example 4.1.1 *Fig. 4.1 illustrates finite-horizon bisimulation on an example DTMC, shown in part (a). Fig.s 4.1 (b) and (c) show quotient DTMCs for 0-step and 1-step finite-horizon bisimulation minimisation, respectively, where quotient state names indicate their corresponding equivalence class (e.g., B_{23} corresponds to DTMC states s_2 and s_3).*

For 2-step minimisation (not shown), blocks B_{23} and B_{01} are both split in two, and only the states s_4 and s_5 remain bisimilar.

From the above, we see that $s_2 \sim_1 s_3$, but $s_2 \not\sim_2 s_3$. Consider the PCTL formula $\Phi = \mathtt{P}_{\bowtie p}[\mathrm{true}\,\mathtt{U}^{\leqslant k}\,a]$, which has depth $d(\Phi) = k$. Satisfaction of Φ is equivalent in states s_2 and s_3 for $k = 1$, but not for $k = 2$. To give another example, for $\Phi' = \mathtt{P}_{>0}[\mathtt{P}_{>0.5}[\mathrm{true}\,\mathtt{U}^{\leqslant 2}\,a]\,\mathtt{U}^{\leqslant 1}\,a]$, which has $d(\Phi') = 1 + 2 - 1 = 2$, we have $s_3 \models \Phi'$, but $s_2 \not\models \Phi'$.

In constructing the 1-step quotient model (Fig. 4.1 (c)), we used s_1 as a representative of equivalence class $B_{01} = \{s_0, s_1\}$, which is why there is a transition to B_{23}. We could equally have used s_0, which would yield a different quotient DTMC, but which still preserves 1-step behaviour.

4.2 Finite-Horizon Bisimulation Minimisation

Bisimulation relations have a variety of uses, but our focus in this thesis is on using them to minimise a probabilistic model prior to verification, in order to improve the efficiency and scalability of the analysis. More precisely, we perform *finite-horizon bisimulation minimisation*, determining the coarsest possible finite-horizon bisimulation relation $\sim_k$, for a given k, and then constructing the corresponding quotient Markov chain. Theorem 4.1.1 tells us that it is then safe to perform verification on the smaller quotient model instead.

We begin, in this section, by presenting a classical *partition-refinement* based minimisation algorithm, which is based on an iterative splitting of an initially coarse partition of the state space until the required probabilistic bisimulation has been identified. In the next section, we will propose on-the-fly approaches which offer further gains in efficiency and scalability.

4.2.1 A Partition-Refinement Based Minimisation Algorithm

The standard approach to partition refinement is to use *splitters* [71, 29], individual blocks in the current partition which show that one or more other blocks contain states that should be split into distinct sub-blocks. An alternative approach is to use a so-called *signature-based* method [27]. The basic structure of the algorithm remains the same, however the approach to splitting differs: rather than using splitters, a *signature* corresponding to the current partition is computed at each iteration for each state s. This signature comprises the probability of moving from s in one step to each block in the partition. In the next iteration, all states with different signatures are placed in different blocks.

Because each iteration of the signature-based algorithm considers the one-step behaviour of every state in the model, it is relatively straightforward to adapt to finite-horizon bisimulation minimisation. Algorithm 4.1 shows the finite-horizon minimisation algorithm MinimiseFiniteHorizon. It takes a DTMC $\mathcal{D}$ and the time horizon k as input. The partition Π is first initialised to group states based on the different combinations of atomic propositions, i.e., states with identical labellings are placed in one block.[1] The partition is then repeatedly split, each time by computing the signatures for each state and splitting accordingly. The loop terminates either when k iterations have been completed or no further splitting is possible. Finally, the quotient model is constructed, as described in the previous section.

Correctness. The correctness of MinimiseFiniteHorizon, i.e. that it generates the coarsest k-step finite-horizon bisimulation, can be argued with direct reference to Definition 4.1.1. For $k = 0$, only the initialisation step at the start of the algorithm is needed. For $k > 0$ the ith iteration of the loop produces a partition Π which groups precisely the

[1] In the algorithm, we store the signatures with the partition, so Π is a list of pairs of blocks (state-sets) and signatures (distributions).

equivalence classes of $\sim_i$, which are constructed from those of $\sim_{i-1}$, as in Definition 4.1.1. It is also clear that we group *all* equivalent states at each step, yielding the coarsest relation. If the algorithm terminates early, at step j, then $\sim_i = \sim_k$ for all $j \leqslant i \leqslant k$.

Algorithm 4.1: MINIMISEFINITEHORIZON

```
Input: $\mathcal{D} = (\mathcal{S}, \mathcal{S}_{init}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$, $k$
$\Pi, \Pi' := \emptyset$ ;                                          // Initialise partition
for $A \subseteq \mathcal{AP}$ do
    $B_A := \{s \in \mathcal{S} \mid L(s) = A\}$
    if $B_A \neq \emptyset$ then $\Pi := \Pi \cup \{(\{B_A\}, \langle\rangle)\}$;
$i := 1$ ;                                                          // Splitting loop
while $i \leqslant k \ \wedge \ \Pi \neq \Pi'$ do
    $\Pi' := \Pi$ ; $\Pi := \emptyset$
    for $s \in \mathcal{S}$ do
        $Sig := \langle\rangle$ ;                                   // Compute signature
        for $B \in \Pi'$ do $Sig(B) := 0$;
        for $s \to s'$ do
            $B_{s'}$ := block of $\Pi'$ containing $s'$
            $Sig(B_{s'}) := Sig(B_{s'}) + \mathbf{P}(s, s')$
        $B_s$ := block of $\Pi'$ containing $s$
        if $\exists (B', Sig) \in \Pi \wedge B' \subseteq B_s$ then
            $B' := B' \cup \{s\}$ ;                                 // New blocks
        else
            $\Pi := \Pi \cup \{(\{s\}, Sig)\}$
    $i := i + 1$
$\mathcal{S}' := \emptyset$ ; $\mathcal{S}'_{init} := \emptyset$ ;  // Build quotient
for $(B, Sig) \in \Pi$ do
    $\mathcal{S}' := \mathcal{S}' \cup \{B\}$
    if $B \cap \mathcal{S}_{init} \neq \emptyset$ then $\mathcal{S}'_{init} := \mathcal{S}'_{init} \cup \{B\}$;
    $\mathbf{P}'(B, \cdot) := Sig$
    $\mathcal{L}'(B) := \mathcal{L}(s)$ for any $s \in B$
return $\mathcal{D}' = (\mathcal{S}', \mathcal{S}'_{init}, \mathbf{P}', \mathcal{AP}, \mathcal{L}')$
```

4.3 On-the-Fly Finite-Horizon Minimisation

A key limitation of the partition-refinement approach presented in the previous section is that it takes as input the full DTMC to be minimised, the construction of which can be expensive in terms of both time and space. This can remove any potential gains in terms of scalability that minimisation can provide.

To resolve this, we now propose methods to compute a finite-horizon bisimulation minimisation in an *on-the-fly* fashion, where the minimised model is constructed directly from a high-level modelling language description of the original model, bypassing construction of the full, un-reduced DTMC. In our case, the probabilistic models are described using the modelling language of the PRISM model checker [59], which is based on guarded commands. Our approach works through a backwards traversal of the model, which allows us to perform bisimulation minimisation on the fly. For simplicity, we focus on preserving the subclass of PCTL properties comprising a single $\mathtt{P}$ operator, more precisely, those of the form $\mathtt{P}_{\bowtie p}[\, b_1 \,\mathtt{U}^{\leqslant k}\, b_2 \,]$ for atomic propositions b_1 and b_2. This is the kind of property most commonly found in practice. The bounded reachability property is a special case of unbounded reachability property $\mathtt{P}_{\bowtie p}[\mathtt{F}\, b_2]$.

4.3.1 The On-the-Fly Minimisation Algorithm

The basic approach to performing finite-horizon minimisation on the fly is shown as FiniteHorizonOnTheFly, in Algorithm 4.2. This takes *model*, which is a description of the DTMC, B_1 and B_2, the sets of states satisfying b_1 and b_2, respectively, in the property $\mathtt{P}_{\bowtie p}[\, b_1 \,\mathtt{U}^{\leqslant k}\, b_2 \,]$, and the time horizon k. The algorithm does not make any assumptions about how sets of states are represented or manipulated. Below, we will discuss two separate instantiations of it.

The algorithm is based on a backwards traversal of the model. It uses a separate algorithm FindMergedPredecessors(*model*, *target*, *restrict*), which queries the DTMC (*model*) to find all (immediate) predecessors of states in *target* that are also in *restrict* (the *restrict* set will be used to restrict attention to the set B_1 corresponding to the left-hand side b_1 of the until formula). The algorithm also groups the predecessor states in

Algorithm 4.2: FINITEHORIZONONTHEFLY

```
Data: $model$, $B_1$, $B_2$, $k$
$P$ := {FINDMERGEDPREDECESSORS($model$, $B_2$, $B_1$)} ; $P'$ := $\emptyset$
$\Pi := \{(B_2, \langle\rangle)\}$
$i := 1$
while $P \neq \emptyset \wedge i \leqslant k$ do
    $(B, D)$ := pop($P$) ;                    // block $B$, (sub)distribution $D$
    for $(B', D') \in \Pi \wedge B \neq \emptyset$ do
        if $B' \cap B \neq \emptyset$ then
            replace $(B', D')$ in $\Pi$ with $(B' \setminus B), D')$ and $(B' \cap B, D' \cup D)$
            $B := B \setminus B'$
            refine all $(B'', D'') \in \Pi$ and $(B, D)$ with respect to the split of $B'$
        end
    end
    if $B \neq \emptyset$ then
        $\Pi := \Pi \cup \{(B, D)\}$
        $P'$ := $P'$ ∪ {FINDMERGEDPREDECESSORS($model$, $B$, $B_1$)}
    end
    if $(P = \emptyset \wedge P' \neq \emptyset)$ then
        $P$ := $P'$ ; $P'$ := $\emptyset$
        $i := i + 1$
    end
end
return FINITEHORIZONQUOTIENT($\Pi$)
```

blocks according to the probabilities with which they transition to *target* and returns these too. As above, each instantiation of Algorithm 4.2 will use a separate implementation of the FINDMERGEDPREDECESSORS algorithm as the merging process is different between explicit and SMT-based approaches.

The main loop of the algorithm iterates backwards through the model: after the ith iteration, it has found all states that can reach the target set B_2 within i steps with positive probability. The new predecessors for each iteration are stored in a set of blocks P. A separate set P' is used to store predecessors of blocks in P, which will then be considered in the next iteration.

More precisely, P (and P') store, like in Algorithm 4.1, a list of pairs (B, D) where B

Algorithm 4.3: FiniteHorizonQuotient

```
Data: $\Pi$
$\mathcal{S}' := \{B_{sink}\}$; $\mathcal{L}'(B_{sink}) = \emptyset$; $\mathcal{S}'_{init} := \emptyset$; $\mathbf{P}'(B_{sink}, \cdot) := \langle B_{sink} \to 1 \rangle$;
for $(B, D) \in \Pi$ do
    $\mathcal{S}' := \mathcal{S}' \cup \{B\}$
    if $B \cap \mathcal{S}_{init} \neq \emptyset$ then $\mathcal{S}'_{init} := \mathcal{S}'_{init} \cup \{B\}$;
    $p_{sink} = 1 - \sum_{(B',D') \in \Pi} D(B')$
    $\mathbf{P}'(B, \cdot) := D \cup \langle B_{sink} \to p_{sink} \rangle$
    $\mathcal{L}'(B) := \mathcal{L}(s)$ for any $s \in B$
end
return $\mathcal{D}' = (\mathcal{S}', \mathcal{S}'_{init}, \mathbf{P}', \mathcal{AP}, \mathcal{L}')$
```

is a block (a set of states) and D is a (partial) probability distribution storing probabilities of outgoing transitions (from B, to other blocks). The set Π, which is used to construct the partition representing the finite-horizon bisimulation relation, is also stored as a list of pairs.

Algorithm 4.2 begins by finding all immediate predecessors of states in B_2 that are also in B_1 and putting them in P. In each iteration, it takes each block-distribution pair (B, D) from P one by one: it will add this to the current partition Π. But, before doing so, it checks whether B overlaps with any existing blocks B' in Π. If so, B' is split in two, and the overlap is removed from B. At this point, the partition Π is refined to take account of the splitting of block B'. We repeatedly recompute the probabilities associated with each block in Π and, if these are then different for states within that block, it is also split.

Each iteration of the main loop finishes when all pairs (B, D) from P have been dealt with. If $i < k$, then newly found predecessors P' are copied to P and the process is repeated. If $i = k$, then the time horizon k has been reached and the finite-horizon bisimulation has been computed.

Finally, the quotient model is built as shown in Algorithm 4.3. The basic construction is as in Algorithm 4.1 but, since on-the-fly construction only partially explores the model, we need to add an extra sink state to complete the DTMC.

Computing predecessors. One of the main challenges in implementing the on-the-fly algorithm is determining the predecessors of a given set of states from the high-level modelling language description. The PRISM language, used here, is based on guarded commands, for example:

$$c > 0 \ \rightarrow \ c/K \ : \ (c' = c - 1) \ + \ 1 - c/K \ : \ (c' = c + 1);$$

The meaning is that, when a state satisfies the *guard* ($c > 0$), the *updates* (decrementing or incrementing variable c) can be executed, each with an associated probability (c/K or $1 - c/K$). We assume here a single PRISM *module* of commands (multiple modules can be syntactically expanded into a single one [84]).

In the following sections, we describe two approaches to finding predecessors: one *symbolic*, which represents blocks (sets of states) as predicates and uses an SMT (satisfiability modulo theories) [21] based implementation; and one *explicit-state*, which explicitly enumerates the states in each block.

4.3.2 Symbolic (SMT-based) Minimisation

Our first approach represents state sets (i.e., blocks of the bisimulation partition) *symbolically*, as predicates over PRISM model variables. If *target* is a predicate representing a set of states, their predecessors, reached by applying some guarded command update *update*, can be found using the *weakest precondition*, denoted **wp**(*update*, *target*). A weakest precondition is an expression which describes the possible valuations of a set of state variables, when an update is reverted on a target states expression. The resulting expression simply represents the set of states that are the predecessors of the given target states expression. More precisely, if the guard of the command is *guard*, and *bounds* represents the lower and upper bounds of all model variables, the following expression captures the set of states, if any, that are predecessors:

$$\mathit{bounds} \;\wedge\; \mathit{guard} \;\wedge\; \mathbf{wp}(\mathit{update},\; \mathit{target})$$

We determine, for each guarded command update in the model description, whether states can reach *target* via that update by checking the satisfiability of the expression above using an SMT solver. FindMergedPredecessors (see Algorithm 4.4) is used to determine predecessors in this way. It also restricts attention to states satisfying a further expression *restrict*.

The probability attached to an update in a guarded command is in general a state-dependent expression *prob* (see the earlier example command) so this must be analysed when FindMergedPredecessors groups states according to the probability with which they transition to *target*. If the SMT query in the algorithm is satisfiable, a valid probability is also obtained from the corresponding valuation (p' in Algorithm 4.4). The conjunction of the expression *predecessor* and $p = \mathit{prob}$ denotes the set of predecessors with the same probability. To obtain all such probabilities, the algorithm adds a *blocking expression* $\mathit{prob} \neq p'$ to the query and repeats the process.

SMT-based methods for probabilistic bisimulation minimisation have been developed previously [25]. One key difference here is that our approach handles transition probabilities expressed as state-dependent expressions, rather than fixed constants, which are needed for some of the models we later evaluate.

4.3.3 Explicit-State Minimisation

As an alternative to the symbolic approach using SMT, we developed an explicit-state implementation of finite-horizon minimisation in which the blocks of equivalent states are represented by explicitly listing the states that comprise them. As in the previous algorithm, the blocks are refined at each time step such that states residing in the same

Algorithm 4.4: FindMergedPredecessors (SMT-based)

```
Data: $model$, $target$, $restrict$
$P := \emptyset$
$bounds$ := variable bounds from $model$
foreach $(guard, updates)$ in model do
    foreach $(prob, update)$ in updates do
        $predecessor := restrict \wedge bounds \wedge guard \wedge \textbf{wp}(update, target)$
        $query := predecessor \wedge (p = prob)$
        while query is satisfiable do
            $p'$ := value of $p$ in $query$
            if $(B, \langle target \to p' \rangle) \in P$ for some $B$ then
                replace $(B, \langle target \to p' \rangle)$ in $P$ with $(B \vee predecessor, \langle target \to p' \rangle)$
            else
                $P := P \cup \{(predecessor, \langle target \to p' \rangle)\}$
            end
            $query := query \wedge (prob \neq p')$
        end
    end
end
return $P$
```

block have equal transition probabilities to the required blocks. To improve performance and store states compactly, we hash them based on the valuation of variables that define them. This is done in such a way that the hash values are bi-directional (one-to-one).

The algorithm explicitly computes the predecessor state for each update and each state in the set *target*, the transition probability is then computed for each predecessor state and these are collected in order to group states into sets. The set *restrict* is not stored explicitly, but rather as a symbolic expression which is then evaluated against each state's variable values to compute the intersection. This symbolic expression simply represents all states that satisfying the predicates.

4.4 Experimental Results

We have implemented the bisimulation minimisation techniques presented in this chapter as an extension of the PRISM model checker [59], and applied them to a range of benchmark models. For both the partition-refinement based minimisation of Section 4.2, and the on-the-fly methods in Section 4.3, we build on PRISM's "explicit" model checking engine. For the SMT-based variant, we use the Z3 solver [20], through the Z3 Java API. All our experiments were run on an Intel Core i7 2.8 GHz machine, using 2 GB of RAM.

Our investigation is in two parts. First, we apply the partition-refinement algorithm to several DTMCs from the PRISM benchmark suite [65] to get an idea of the size of reductions that can be obtained on some standard models. We use: *Crowds* (an anonymity protocol), *EGL* (a contract signing protocol) and *NAND* (NAND multiplexing). Details of all models, parameters and properties used can be found at [85]. A common feature of these models is that they have a single initial state, from which properties are verified. Since on-the-fly approaches explore backwards from a target set, we would usually need to consider time horizons k high enough such that the whole model was explored.

So, to explore in more depth the benefits of the on-the-fly algorithms, we consider another common class of models in probabilistic verification: those in which we need to exhaustively check whether a property is true over a large set of possible configurations. We use *Approximate majority* [4], a population protocol for computing a majority value amongst a set of K agents, and two simple models of *genetic algorithms* [82] in which a population of K agents evolves over time, competing to exist according to a fitness value in the range $0, \ldots, N-1$. In the first variant, *tournament*, the agent with the highest value wins; in the second, *modulo*, the sum of the two scores is used modulo N. Again, details of all models, parameters and properties used can be found at [85].

4.4.1 The partition-refinement algorithm

Fig. 4.2 shows results for the partition-refinement algorithm. The top row of plots shows the number of blocks in the partition built by finite-horizon bisimulation minimisation for different values of k on the first three benchmark examples. For the largest values of

k shown, we have generated the partition corresponding to the full (non-finite-horizon) bisimulation. In most cases, the growth in the number of blocks is close to linear in k, although it is rather less regular for the *NAND* example. In all cases, it seems that the growth is slow enough that verifying finite-horizon properties for a range of values of k can be done on a considerably smaller model than the full bisimulation.

The bottom row of plots shows, for the same examples, the time required to perform

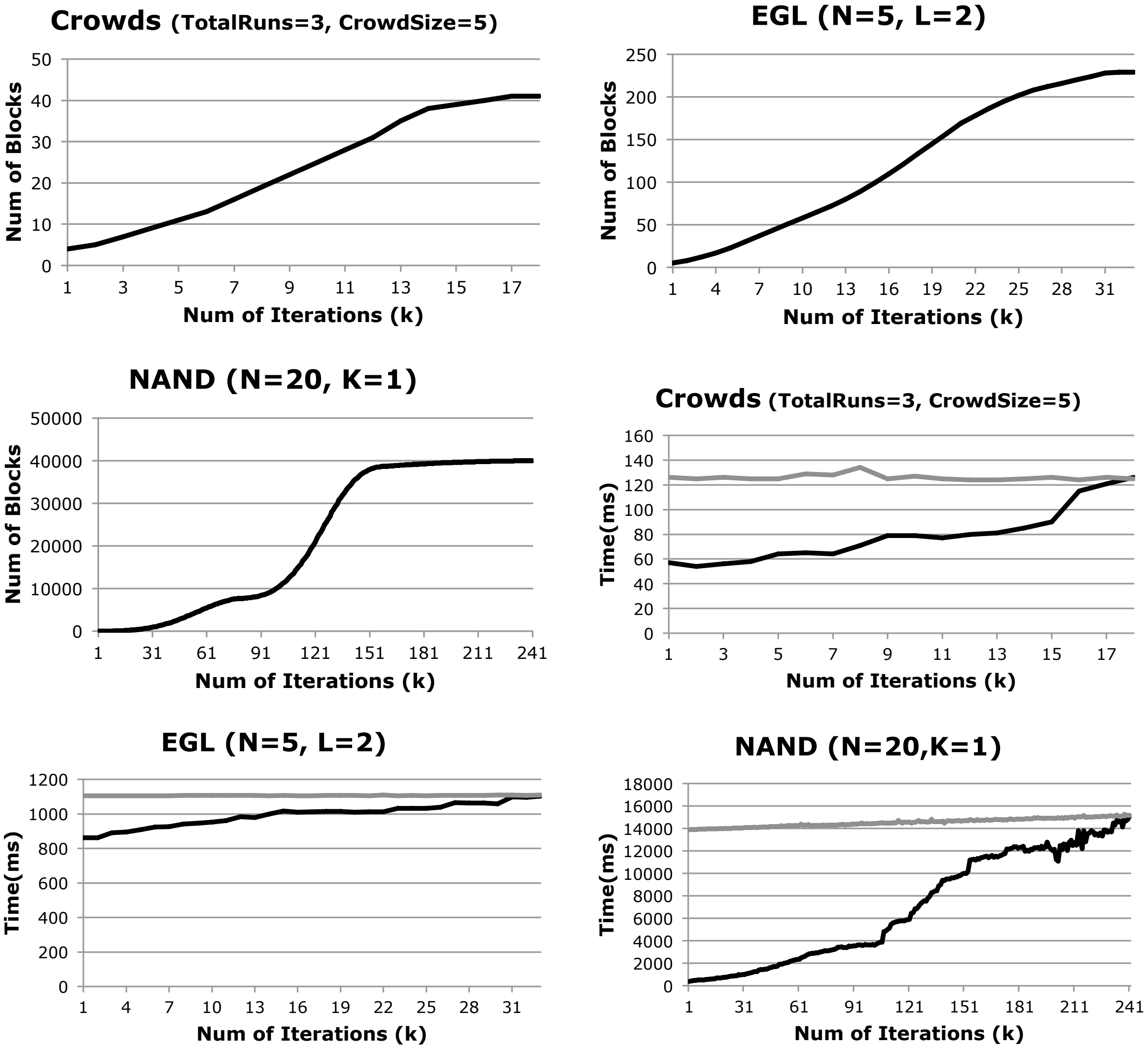


Figure 4.2: Results for partition-refinement. Top: quotient size for varying time horizon k. Bottom: time for finite-horizon (black) and full (grey) minimisation/verification.

bisimulation minimisation and then verify a k-step finite-horizon property (details at [85]). The black lines show the time for finite-horizon minimisation, the grey lines for full minimisation. The latter are relatively flat, indicating that the time for verification (which is linear in k) is very small compared to the time needed for minimisation. However, we see significant gains in the total time required for finite-horizon minimisation compared to full minimisation.

However, despite these gains, the times to minimise and verify the quotient model are still larger than to simply build and verify the full model. This is primarily because the partition refinement algorithm requires construction of the complete model first, the time for which eclipses any gains from minimisation. This was the motivation for the on-the-fly algorithms, which we evaluate next.

4.4.2 On-the-Fly Algorithms

Table 4.1 and Table 4.2 shows model sizes and timings for the on-the-fly algorithms on a range of models and scenarios. The left four columns show the model (and which on-the-fly algorithm was used), any parameters required (N or K) and the time horizon k. Next, under the headings 'Full Red.' and 'Finite Horiz.', we show the reductions in model size obtained using full (non-finite-horizon) and finite-horizon minimisation (for several k), respectively. In the first case, 'States' and 'Blocks' show the size of the full DTMC and the fully reduced quotient model, respectively. For the second case, 'Blocks' is the size of the finite-horizon quotient model and, to give a fair comparison, 'States' is the number of states in the full DTMC that can reach the target of the property within k steps (i.e., the number of states across all blocks). The rightmost three columns show the time required to build the model in three scenarios: 'Finite Horiz.' uses the on-the-fly approach over k steps; 'Full Red.' builds the full (non-finite-horizon) quotient by repeating the on-the-fly algorithm until all states have been found; and 'PRISM' builds the full model using its most efficient (symbolic) construction engine.

Model	Param.s		k	Full Red.		Finite Horiz.		Time (s)		
	N	K		States	Blocks	States	Blocks	PRISM	Full Red.	Finite Horiz.
Approx. majority	n/a	100	20	20300	10201	242	122	11.0	14.2	0.2
			40			882	442			0.3
			60			1922	962			0.4
		150	100	45450	22801	5202	2602	46.1	83.1	1.2
			150			11552	5777			5.1
			200			20402	10202			15.6
		200	250	80600	40401	31752	15877	memout	293.5	40.8
			300			45602	22802			93.9
			350			61952	30977			180.8
		250	375	125750	63001	71064	35533	memout	773.5	247.8
			400			80802	40402			323.2
			425			91164	45583			416.6
Genetic alg. tournament	8	22	8	1184040	22	6435	10	19.2	5.3	0.3
			9			11440	11			0.4
			10			19448	12			0.4
		23	8	1560780	23	6435	10	31.1	7.0	0.3
			9			11440	11			0.4
			10			19448	12			0.4
	10	21	8	10015005	21	24310	10	59.0	43.6	0.5
			9			48620	11			0.6
			10			92378	12			0.7
		22	8	14307150	22	24310	10	61.3	51.3	0.5
			9			48620	11			0.6
			10			92378	12			0.7
Genetic alg. modulus	7	19	8	177100	29565	22179	3638	0.4	475.3	6.8
			9			39404	6491			21.6
			10			66002	10914			64.3
		20	8	230230	38431	22179	3637	0.5	778.6	6.9
			9			39404	6488			20.3
			10			66068	10914			65.9
	9	11	6	75582	12707	24822	3435	0.3	79.9	7.7
			7			51756	8084			32.3
			8			70448	11745			58.3
		12	6	125970	21145	24906	3450	0.3	253.5	7.8
			7			54440	8482			37.4
			8			88642	14207			102.4

Table 4.1: Experimental results for on-the-fly bisimulation minimisation : Explicit variant

Model	Param.s		k	Full Red.		Finite Horiz.		Time (s)		
	N	K		States	Blocks	States	Blocks	PRISM	Full Red.	Finite Horiz.
Genetic alg. tournament	4	9	3	165	9	20	5	0.03	155	4.5
			4			35	6			11.1
			5			56	7			23.5
		10	3	220	10	20	5	0.03	215	9.3
			4			35	6			15.1
			5			56	7			31.1
	5	9	3	330	9	35	5	0.04	723.4	22.1
			4			70	6			70.7
			5			126	7			180.9
		10	3	495	10	35	5	0.04	1998.7	48.8
			4			70	6			82.0
			5			126	7			233.7

Table 4.2: Experimental results for on-the-fly bisimulation minimisation : SMT variant

First, we note that finite-horizon minimisation yields useful reductions in model size in all cases, both with respect to the full model and to normal (non-finite horizon) minimisation. Bisimulation reduces models by a factor of roughly 2 and 5, for the *Approximate majority* and *Modulus* examples, respectively. For *Tournament*, a very large reduction is obtained since, for the property checked, the model ends up being abstracted to only distinguish two fitness values. Finite-horizon minimisation gives models that are smaller again, by a factor of between 2 and 10 on these examples, even for relatively large values of k on the *Approximate majority* models. Comparing columns 7 and 8 in Table 4.1 shows that much of the reduction is indeed due to merging of bisimilar states, not just to a k-step truncation of the state space from the backwards traversal.

Regarding performance and scalability, we first discuss results for the SMT-based implementation, which is shown in Table 4.2. We were only able to apply this to the *Tournament* example, where a very large reduction in state space is achieved. On a positive note, the SMT-based approach successfully performs minimisation here and gives a symbolic (Boolean expression) representation for each block. However, the process is slow, limiting applicability to DTMCs that can already be verified without minimisation.

Our experiments showed that the slow performance was largely caused by testing for overlaps between partition blocks resulting in a very large number of calls to the SMT solver. An example of SMT queries for the tournament game model are provided in Appendix A.

The explicit-state on-the-fly implementation performed much better and Table 4.1 shows results for all three models. In particular, for the *Tournament* example, finite-horizon minimisation and verification is much faster than verifying the full model using the fastest engine in PRISM. This is because we can bypass construction of the full models, which have up to 14 million states for this example. For the *Modulus* example, the model reductions obtained are much smaller and, as a result, PRISM is able to build and verify the model faster. However, for the *Approximate Majority* example, the minimisation approach can be applied to larger models than can be handled by PRISM. For this example, although the state spaces of the full model are manageable, the models prove poorly suited to PRISM's model construction implementation (which is based on binary decision diagram data structures).

4.5 Conclusions

We have presented model reduction techniques for verifying finite-horizon properties on discrete-time Markov chains. We formalised the notion of k-step finite-horizon bisimulation mininisation and clarified the subset of PCTL that it preserves. We have given both a partition-refinement algorithm and an on-the-fly approach, implemented in both a symbolic (SMT-based) and explicit-state manner as an extension of PRISM. Experimental results demonstrated that significant model reductions can be obtained in this manner, resulting in improvements in both execution time and scalability with respect to the existing efficient implementations in PRISM.

CHAPTER 5

Parametric Model Checking of Linear Inductive Models

In the previous chapter, we have presented a novel algorithm to combat the so called "state space explosion problem" for finite-horizon properties. Most of the existing algorithms which we discussed in this thesis so far are using the bisimulation minimisation technique to reduce the size of the state space of the original model, including the novel algorithm discussed in the last chapter.

The bisimulation minimisation technique aggregates the equivalence classes of bisimilar states into single states, hence, reducing the original model into a quotient model. The resulting quotient model is equivalent to the original model as it preserves both long-run and transient properties. This minimisation process requires the exploration of the entire state space of a given model. This whole process can be executed in two different ways. In the first case, the complete exploration of the original model is performed prior to the application of the minimisation. In the second case, the exploration is carried out in parallel to the minimisation (on-the-fly). The minimisation process along with complete exploration of the original model can consume considerable computational resources for larger state spaces. After going through such an expensive process, the reduction factor (i.e. the state space ratio of the quotient and original models) may not be very high for certain models. In such cases, it is not preferable to apply bisimulation minimisation techniques as the gain may be overshadowed when comparing to the performance of conventional model checking. The probabilistic models with such lower reduction factors require different strategies to overcome the state space explosion problem.

In this chapter, we introduce a novel strategy to handle linear inductive DTMCs, i.e. a class of models whose state space grow linearly with respect to a parameter, which is referred to as a recurrence parameter. In these linear inductive models, a particular segment of the state space is repeated multiple times corresponding to a recurrence parameter. We devise methods that automatically detect and extract such models from a high-level model description, and derive underlying recurrence relations from the repeating segment of this model. We then form a function with respect to recurrence parameter using the solutions of derived recurrence relations and perform model checking using this function. A complete implementation of this inductive model reduction technique is developed as an extension of the PRISM model checker. We also show that this technique is extended to verify step-bounded and cost-bounded reachability properties on arbitrary DTMCs, reduces to the problem of verifying linear inductive DTMCs.

The result of the experiments carried out clearly show that the proposed approach contributes to the alleviation of the state space explosion for the given class of models compared to the conventional model checking with and without the application of bisimulation minimisation. In addition to that, this research raises opportunities for more research in the direction of model reduction techniques for inductive models with non-linear state space growth.

The remaining sections of this chapter are structured as follows. Section 5.2 describes the specific class of models, referred to as inductive models, used in this work. Section 5.3

formally explains linear inductive DTMCs. Section 5.4 describes the novel algorithm introduced as the end product of this work. Section 5.5 explains in detail the numerical computations that are used in this work to produce a function with respect to the recurrence parameter. Section 5.6 describes the experiments carried out on two different properties of interest, unbounded and cost-bounded, and analyses the results.

5.1 Terminology

In this section, we explain all the terminology that is introduced in this chapter. A block is a set of states which may assign one of its states as its representative. A **recurrence parameter** λ, which is a constant, acts as either a lower or upper bound for recurrence variables. A **recurrence variable** Λ causes a block to repeat a number of times, where the repeating block is called a **recurrent block**. A recurrent block differs from another recurrent block only by the valuation of Λ. The width of recurrent block B_1, which is $max(\Lambda_{B_1}) - min(\Lambda_{B_1}) + 1$, is called the **recurrent interval**. The term **recurrent borderline** Ω refers to the scope of the (recurrent) region, which is within the range of Λ, where all the recurrent blocks resides.

The region before the recurrent borderline is called the **initial region**, which has a width $\delta_1(= (min(\Omega) - min(\Lambda))/n)$, where n is the number of blocks in the respective region. The width of the **recurrent region** is denoted by $\delta_2(= (max(\Omega)-min(\Omega)+1)/n)$. The region that lies after the recurrent region is called the **end region** which has a width $\delta_3(= (max(\Lambda) - max(\Omega))/n)$. The initial and end regions contain only one block each whereas the recurrent region contains more than one (recurrent) blocks. The recurrence relationship exhibited by the recurrent blocks are solved to form a function $f(\lambda)$, which allows us to verify the property of interest for the various values of λ.

5.2 Inductive Models

The growth of the state space of the inductive models varies with respect to their parameters. In Table 5.1, the state space growth in relation to parameters of both NAND and EGL inductive models, which are from the PRISM benchmark suite, has been presented. In the case of *NAND* model, the state space grows linearly with respect to the parameter K. On the other hand, the state space growth of the *EGL* model grows non-linearly in accordance to the parameter N. In this research, we mainly focus on the inductive models that exhibits linear recurrence behaviour in relation to a single parameter.

Model	Parameters		States	Growth factor
	N	K		
NAND	14	11	98457	10843
		12	109300	
		13	120143	
	15	11	125048	13784
		12	138832	
		13	152616	

NAND

Model	Parameters		States	Growth factor
	L	N		
EGL	5	5	95230	359424
		6	454654	↓
		7	2113534	1658880
	6	5	115710	437248
		6	552958	↓
		7	2572286	2019328

EGL

Table 5.1: The state space growth of inductive models: *NAND* with respect to its parameters N and K, and *EGL* in relation to its parameters L and N

The models whose state space grow non-linearly with respect to a parameter are not the models of our interest as they cannot encompass the linear recurrence behaviour. We can clearly conjecture that only the inductive models whose state space grow linearly, such as *NAND*, may have the potential to comprise the linear recurrence behaviour. In addition to this, we also notice that among the parameters of *NAND*, parameter K induces a linear growth whilst the parameter N causing a non-linear growth in the state space. In this chapter, we refer to parameters that enforce linear state space growth, e.g. the parameter K of *NAND*, whilst encompassing the linear recurrence behaviour in the corresponding inductive model as recurrence parameters.

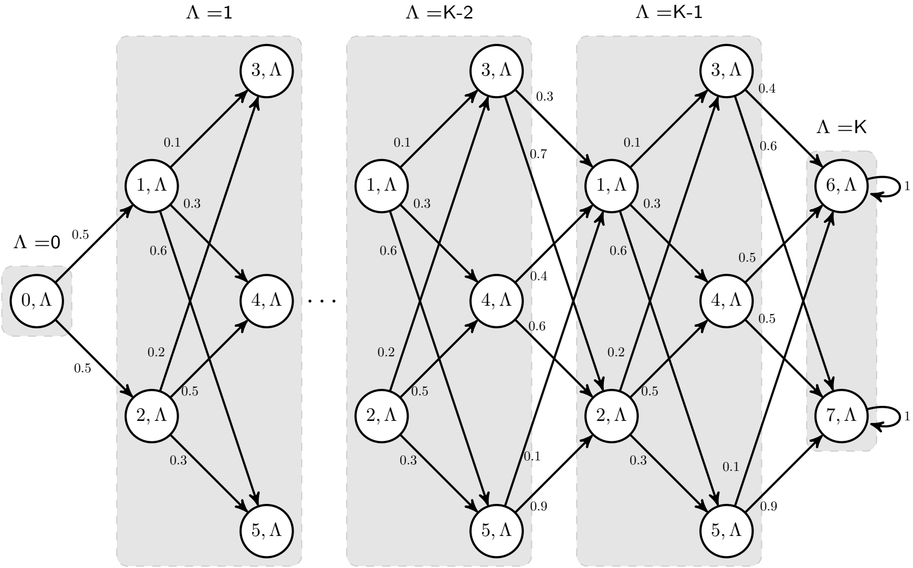


Figure 5.1: An example inductive model that grows linearly with respect to parameter K and encompasses the linear recurrence behaviour

We have introduced an example model in Figure 5.1 with the purpose of illustrating the properties of a linear inductive model that exhibits linear recurrence behaviour. The model description of this example is shown in Figure 5.2 for a clear understanding of the core components of the model description that will be later detailed in Section 5.4.2. This particular model grows linearly with respect to parameter K. In this example model, the *recurrence variable* is denoted as Λ and its upper bound is recurrence parameter K. The *recurrent interval* of this example model is 1 as the value of Λ within a recurrent block is the same. In the case of example model, the recurrent borderline is 1 to $K-2$ as the block of states with similar transition behaviour repeats within this range. Therefore, we claim that this model clearly incorporates the linear recurrence behaviour as a certain block of states are repeated based on Λ, where the repeating set of states are behaving similar in relation to Λ. This example model will be used throughout this chapter to explain the properties of a linear inductive model.

```
dtmc

module inductive_model

const int K;

x : [0..7];
y : [0..K];

[] x=0 & y=0           → 0.5 : (x'=1) & (y'=y+1) + 0.5 : (x'=2) & (y'=y+1);
[] x=1                 → 0.1 : (x'=3) + 0.3 : (x'=4) + 0.6 : (x'=5);
[] x=2                 → 0.2 : (x'=3) + 0.5 : (x'=4) + 0.3 : (x'=5);
[] x=3 & y < (K − 1)   → 0.3 : (x'=1) & (y'=y+1) + 0.7 : (x'=2) & (y'=y+1);
[] x=4 & y < (K − 1)   → 0.4 : (x'=1) & (y'=y+1) + 0.6 : (x'=2) & (y'=y+1);
[] x=5 & y < (K − 1)   → 0.1 : (x'=1) & (y'=y+1) + 0.9 : (x'=2) & (y'=y+1);
[] x=3 & y = (K − 1)   → 0.4 : (x'=6) & (y'=y+1) + 0.6 : (x'=7) & (y'=y+1);
[] x=4 & y = (K − 1)   → 0.5 : (x'=6) & (y'=y+1) + 0.5 : (x'=7) & (y'=y+1);
[] x=5 & y = (K − 1)   → 0.1 : (x'=6) & (y'=y+1) + 0.9 : (x'=7) & (y'=y+1);
[] x>5 & x<8           → true;
endmodule
```

Figure 5.2: The PRISM model description of the inductive example shown in Figure 5.1

5.3 Linear Inductive DTMCs

In this section, we explain more formally the class of models with linear recurrence behaviour that we are able to handle with the techniques in this chapter. We call these *linear inductive DTMCs* and they are divided into three regions: a first (initial) region, a second (inductive) region and a third (final) region. A DTMC is constructed by composing one copy of the first and third regions, with some number (say, $N \geqslant 1$) copies of the second region in between. States in a region can only transition either to the current, or to the next region. That is, states in region 1 can only transition to the same region or to the first instance of region 2; states in an instance of region 2 can only transition to the same or the next instance (or to region 3 if it is the final instance); and states in region 3 can only transition to other states in region 3.

States of a linear inductive DTMC are of the form (s, Λ), where s is a *local* state and Λ is an integer variable. We use δ_1, δ_2 and δ_3 to denote the widths of the three regions, i.e., the number of different (consecutive) values of Λ that can appear in each of the regions. For simplicity, we assume that the lowest value of Λ is 0. So, the range of possible values of Λ is $\{0, \ldots, \delta_1{+}N\delta_2{+}\delta_3{-}1\}$. The model is defined by three separate transition probability matrices $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ and three separate labelling functions $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$. Formally the model is defined as follows.

Definition 5.3.1 (Linear Inductive DTMC) *A* linear inductive DTMC *is a tuple* $\mathcal{D} = (\mathcal{S}, (\delta_1, \delta_2, \delta_3), \mathcal{S}_{\mathit{init}}, (\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3), \mathcal{AP}, (\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3))$*, where:*

- $\mathcal{S}$ *is a finite set of* local *states;*
- δ_1,δ_2 *and* δ_3 *are the widths of the first, second and third regions of* $\mathcal{D}$*;*
- $\mathcal{S}_{\mathit{init}} \subseteq \mathcal{S} \times \{0, \ldots, \delta_1{-}1\}$ *is a set of initial states;*
- $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ *are transition probability matrices for the 3 regions:*
  - $\mathbf{P}_1 : (\mathcal{S} \times \{0, \ldots, \delta_1{-}1\}) \times (\mathcal{S} \times \{0, \ldots, \delta_1{+}\delta_2{-}1\}) \to [0, 1]$
  - $\mathbf{P}_2 : (\mathcal{S} \times \{0, \ldots, \delta_2{-}1\}) \times (\mathcal{S} \times \{0, \ldots, 2\delta_2{-}1\}) \to [0, 1]$
  - $\mathbf{P}_3 : (\mathcal{S} \times \{0, \ldots, \delta_3{-}1\}) \times (\mathcal{S} \times \{0, \ldots, \delta_3{-}1\}) \to [0, 1]$
- $\mathcal{AP}$ *is a set of atomic propositions; and*
- $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ *are labelling functions for the 3 regions:*
  - $\mathcal{L}_1 : (\mathcal{S} \times \{0, \ldots, \delta_1{-}1\}) \to 2^{\mathcal{AP}}$
  - $\mathcal{L}_2 : (\mathcal{S} \times \{0, \ldots, \delta_2{-}1\}) \to 2^{\mathcal{AP}}$
  - $\mathcal{L}_3 : (\mathcal{S} \times \{0, \ldots, \delta_3{-}1\}) \to 2^{\mathcal{AP}}$

In the case of example model shown in Figure 5.1, we have $\mathcal{S} = \{0, 1, 2, 3, 4, 5, 6, 7\}$. The lower and upper bounds of the recurrent borderline Ω is 1 and $K-2$, respectively. The width of initial, recurrent and end regions are $\delta_1 = (1-0)/1 = 1$, $\delta_2 = (K-2-1+1)/(K-2) = 1$, $\delta_3 = (K-(K-2))/1 = 2$ and $N = K-2$.

A DTMC, constructed from this inductive definition, and comprising N copies of region 2, is called an *expanded DTMC* and is defined as follows.

Definition 5.3.2 (Expanded DTMC) *Given a linear inductive DTMC defined by the tuple* $(\mathcal{S}, (\delta_1, \delta_2, \delta_3), \mathcal{S}_{\mathit{init}}, (\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3), \mathcal{AP}, (\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3))$ *and integer* $N \geqslant 1$*, the* expanded DTMC *is* $(S \times \{0, \ldots, \delta_1{+}N\delta_2{+}\delta_3{-}1\}, \mathcal{S}_{\mathit{init}}, \mathbf{P}, \mathcal{AP}, \mathcal{L})$ *where, for any* $s, s' \in \mathcal{S}$*:*

- $\mathbf{P}((s, i), (s', i')) = \mathbf{P}_1((s, i), (s', i'))$ *for any* $0 \leqslant i < \delta_1, 0 \leqslant i' < \delta_1{+}\delta_2$
- $\mathbf{P}((s, \delta_1{+}j\delta_2{+}i), (s', \delta_1{+}j\delta_2{+}i')) = \mathbf{P}_2((s, i), (s', i'))$ *for any* $0 \leqslant i < \delta_2, 0 \leqslant i' < 2\delta_2$ *and* $0 \leqslant j < N$
- $\mathbf{P}((s, \delta_1{+}N\delta_2{+}i), (s', \delta_1{+}N\delta_2{+}i')) = \mathbf{P}_3((s, i), (s', i'))$ *for any* $0 \leqslant i < \delta_3, 0 \leqslant i' < \delta_3$

and $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ *are otherwise 0.*

Similarly, the labelling function $\mathcal{L}$ *is defined, for any* $s \in \mathcal{S}$*, as:*

- $\mathcal{L}((s, i)) = \mathcal{L}_1((s, i))$ *for any* $0 \leqslant i < \delta_1$
- $\mathcal{L}((s, \delta_1{+}j\delta_2{+}i)) = \mathcal{L}_2((s, i))$ *for any* $0 \leqslant i < \delta_2$ *and* $0 \leqslant j < N$
- $\mathcal{L}((s, \delta_1{+}N\delta_2{+}i)) = \mathcal{L}_3((s, i))$ *for any* $0 \leqslant i < \delta_3$

5.4 Main Algorithm

We propose a novel algorithm in this section that overcomes the bottleneck of handling inductive models. As we discussed earlier, the existing algorithms are not promising to handle this class of models effectively for larger values of their parameters. However, this proposed algorithm elegantly forms a function with respect to a recurrence parameter for a particular property of interest as the end result. This algorithm can be broken into six core functionalities as shown in Algorithm 5.1. They are as follows:

- **Preprocess** : DTMC models, with property of interest either time or cost, are preprocessed during this phase such that they can be used as inductive models.

- **DeriveRecurrentIntervalAndBorderline** : The recurrent interval is computed as a first step in this phase. Thereafter, the borderline of the recurrence region is determined using the recurrent interval and the guard of each command.

- **ConstructRegion** : The regions: first, second (recurrence region) and third are constructed during this phase.

- **IsRecurring** : This phase ensures whether the assumption about the recurrence region is valid within the recurrent borderline by comparing a sample repeating set of states in relation to the recurrence parameter.

- **Extract** : The recurrence relations and their base cases are extracted from the constructed regions. This process also differs with respect to the employed approach.

- **Solve** : This phase involves numerical computations to solve the extracted recurrence relations. As the end result of this phase, a function is formed with respect to the recurrence parameter. This function can answer the property of our interest for various values of the recurrence parameter.

As shown in Algorithm 5.1, the model description m, property of interest p and recurrence parameter λ have to be passed as the inputs to start the algorithm. Currently, the algorithm cannot infer λ automatically, however, this can be achieved by testing all the parameters against the conditions mentioned in 5.4.2.

Initially, m goes through the preprocessing phase where the model description m is transformed into an inductive model m' with respect to λ. After the preprocessing phase, the recurrent interval δ and borderline Ω are derived using both m' and λ, where Ω is a pair data structure that is composed of both upper and lower bounds of the recurrent region. Subsequently, the three regions $\gamma_1, \gamma_2, \gamma_3$, which are required for the extraction of recurrence relations, are constructed using m', p and Ω. As γ_2 represents the recurrent region, the algorithm confirms whether γ_2 encompasses the recurrence behaviour. In case γ_2 encompasses it, the recurrence relation set Δ is extracted using the $\gamma_1, \gamma_2, \gamma_3$ regions. Finally, the extracted set Δ is solved numerically to produce the function $f(\lambda)$. In the following subsections, these six core functionalities will be discussed in detail, respectively.

Algorithm 5.1: InductiveReduction

```
Input: $m, p, \lambda$
$m' \leftarrow$ Preprocess($m, \lambda$)
$\Omega \leftarrow$ DeriveRecurrentIntervalAndBorderLine($m', \lambda$)
$\gamma_1, \gamma_2, \gamma_3 \leftarrow$ ConstructRegions($m', p, \Omega$)
if IsRecurring($\gamma_2$) then
    $\Delta \leftarrow$ Extract($\gamma_1, \gamma_2, \gamma_3$)
    $f(\lambda) \leftarrow$ Solve($\Delta$)
end
```

5.4.1 Preprocessing the models

The techniques in this chapter target three different cases. Firstly, models which are, by definition, linear inductive models. Secondly, step-bounded reachability properties for arbitrary DTMCs. Thirdly, cost-bounded properties for arbitrary DTMCs. The model description of such models has to be preprocessed for the inductive unfolding. Algorithm 5.2 shows the pseudocode of the preprocessing phase. Initially, the algorithm makes a copy of the model description m, which is passed in as one of the parameters. The purpose of the copy m' is to be modified in the latter stages of this phase whilst keeping the original m untouched. In order to be an inductive model, it is mandatory for a model description to contain the recurrence parameter λ. Henceforth, the algorithm confirms whether the copy m' contains λ as the next step. In the case when λ is not present in m', the algorithm looks for two special cases of the recurrence parameter λ.

Algorithm 5.2: PREPROCESS

```
Input: $m, \lambda$
Output: $m'$
$m' \leftarrow$ copyOf($m$)
if $m'$ contains $\lambda$ then
    if $\lambda = t$ then
        introduce time variable $t$
        incorporate $t$, $\forall c \in m'$
    else if $\lambda = r$ then
        introduce cost variable $r$
        incorporate $r$, $\forall c \in m'$ that satisfies $\mathbf{R}$
        remove $\mathbf{R}$ from $m'$
    else
        throwError("model $m$ is not supported")
    end
end
return $m'$
```

The first special case is λ being the time variable t. This special case denotes that the user wants to transform a non-inductive DTMC into a timed inductive DTMC. Therefore, the time variable t is introduced in the copy model description m' and every command in m' is incorporated with t. Figure 5.3 shows an example of incorporating t in a command c. A command in the model description represents one time step behaviour of any given state for which the corresponding command is enabled. Therefore, the time variable t must be not only included in the guard of the commands but also has to be in the updates of the corresponding commands. Henceforth, t is incremented by one to represent a single time step in all the updates.

The other special case is λ being the cost variable r. In this case, the user requires the algorithm to preprocess a DTMC model with a cost function into a linear inductive DTMC with costs. Generally, the cost component of a model description will be in the

```
...
[] a=0 → 1.0 : (x'=0);
```

transforms into

```
...
[] a=0 & t<k → 1.0 : (x'=0) & (t'=t+1);
```

Figure 5.3: The illustration of incoporating the time variable t in a command

following form.

$$[\ \textit{action}\ ]\ \ \textit{guard}\ \ :\ \ \textit{cost}\ ;$$

This can be interpreted as a certain *cost* is provided when a transition from a state satisfies the *guard* and is also labelled with the *action*. In this case, the first step is similar to the previous case which is introducing the cost variable r. However in this case, only the commands that match the guard and the action label of the respective cost component will be incorporated with cost variable r. The incorporated r will be updated using the respective *cost* value. Afterwards, the respective cost function will be removed from the model description m'. In addition, the reachability property in the form $\mathsf{P}_{=?}[\mathsf{F}_{\leqslant c}\mathsf{target}]$ is also transformed into the form of $\mathsf{P}_{=?}[\mathsf{F}\,\mathsf{target} \wedge r \leqslant c]$. Finally, the preprocessed m' will be returned as the output, unless the recurrence parameter λ is not any of the special cases.

5.4.2 Derivation of the recurrent interval and borderline

Once the model description m has gone through the preprocessing phase, the preprocessed model description m' needs to be analysed with respect to its various components. As a result of this analysis, the recurrent interval δ and borderline Ω will be derived. In this context, the term *recurrent interval* refers to the difference between the values of recurrence variable Λ of both entry and exit states of a recurrent block whereas the term *recurrent borderline* refers to the particular segment of the model (sub model) where the recurrence behaviour is exhibited. The components of the model description, which allow the derivation of δ and Ω, are variable, guard, probability and update expressions. The presence of the recurrence parameter λ and variable Λ in each of these components determines the existence of recurrence behaviour.

```
Algorithm 5.3: DeriveRecurrentIntervalAndBorderline
Input: $m', \lambda$
Output: $\delta, \Omega$
$\Lambda \leftarrow$ IdentifyRecurrenceVariable($m', \lambda$)
isValidProb $\leftarrow$ CheckProbExprs($m', \Lambda$)
if isValidProb then
    $\delta \leftarrow$ ComputeRecurrentInterval($m', \Lambda$)
    $\Omega \leftarrow$ ComputeRecurrentBorderLine($m', \Lambda, \delta$)
else
    throwError("model $m'$ is not supported")
end
return $\delta, \Omega$
```

The pseudocode of deriving the recurrent interval δ and borderline Ω is shown in Algorithm 5.3. During this phase, the preprocessed model description m' and the recurrence parameter λ are passed as inputs to the algorithm. Initially, the recurrence variable Λ is identified using the inputs m' and λ as it is needed to check whether m' exhibits recurrence behaviour. Thereafter, the probability expressions in m' are checked against the conditions (which will be detailed in the following subsections) that confirm the existence of valid recurrence behaviour. In the case where all probability expressions satisfy the conditions, the recurrent interval δ and borderline Ω are computed. Otherwise, the algorithm terminates as the model description m' does not include the properties of an inductive model of interest. In the following subsections, we will discuss in detail the contribution of the variable, update, probability and guard expressions, respectively.

Variable expressions

Variable expressions, i.e. declarations, are one of the core components of a model description as any unique composition of the values of all variables creates a state of the model.

The variables are declared as a first step in the model description. However, they are not always predefined in the model description as they can also be defined as dependent on a parameter that can be assigned to a value at later stage. The range of a variable has both upper and lower bounds, where a certain parameter can represent only one of them. This confirms that the recurrence parameter can be present in the variable component as either upper or lower bound.

In reality, it is very likely for a state to be made up of more than one variable. As the number of variables depending on the recurrence parameter λ is one of the factors that governs the existence of linear recurrence behaviour, it is important to analyse the underlying relation between them. For linear recurrence behaviour to exist in an inductive model, the growth of the state space with respect to λ must be linear. In the case when there are not any dependent variables as shown in Figure 5.4, it is trivial to understand that the respective model cannot exhibit the linear recurrence behaviour as there is no correlation between the growth of the state space and λ. Meanwhile, when there is more than one dependent variable as shown in Figure 5.6, it is not evident whether the state space growth is linear with respect to λ or not, henceforth, we do not currently handle such models. In fact, the existence of linear recurrence behaviour is trivial only when there is one dependent variable, as shown in Figure 5.5. This variable is referred to as the recurrence variable Λ.

```
dtmc
module non_inductive
const int K;
    x : [0..7];
    y : [1..10];
...
```

Figure 5.4: None of the variables depend on the parameter K

```
dtmc
module inductive
const int K;
    x : [0..7];
    y : [0..K];
...
```

Figure 5.5: Only One variable depends on the parameter K

```
dtmc
module non_inductive
const int K;
    x : [0..K];
    y : [0..K];
...
```

Figure 5.6: More than one variable depends on the paramter K

Probability expressions

A probability expression represents the transition probability of its corresponding update. For the recurrence behaviour to exist in an inductive model, any state in a recurrent block must have a recurrently similar state in all other recurrent blocks. In the case when a probability expression relies on Λ, the transition probability will vary between any such similar states as Λ is different. Thus, this type of probability expression violates the recurrence similarity relation.

Update expressions

An update expression defines the outgoing transition of a state, in other words, the state it can move from a given state. This update expression can be applied on a state only when that state satisfies the guard attached along with the respective update. We have already discussed recurrent block of states in Section 5.3. The existence of recurrence variable Λ in an update expression determines the nature of the transition between recurrent blocks. Algorithm 5.4 shows the pseudocode for computing the recurrent interval using update expressions.

Algorithm 5.4: ComputeRecurrentInterval

```
Input: $m', \Lambda$
Output: $\delta$

$\delta \leftarrow 1$
for each update $u$ in $m'$ do
    if $u$ contains $\Lambda$ then
        if $\Lambda$ is updated by additive increase then
            $\delta \leftarrow$ Max($\delta$, incrementValue)
        else
            throwError("model $m'$ is not supported")
        end
    end
end
return $\delta$
```

This algorithm requires the model description m' and Λ as the inputs to start the phase. Initially, the recurrent interval δ is assigned with the least possible value 1. Thereafter, the algorithm loops through each update $u \in m'$ to analyse the nature of the update u. The possible occurrences of update u with respect to Λ are as follows:

- u does not update Λ: The respective update defines the transition of a state that occurs within the same recurrent interval. Thus, the impact of this form of u does not affect the existence of the recurrence behaviour.

- u updates Λ and updated by additive increase : This is the only possible case where the linear recurrent intervals can be obtained. It is a must to verify that the update is only either increase or decrease. If there is a mixture of both among the updates, the recurrence flow in one direction can be disrupted.The following form of u represents an example of this case:

$$\mathit{\Lambda}' = \mathit{\Lambda} + constant$$

- u updates Λ and updated by multiplicative increase : In this case, the recurrent interval cannot be a constant as the transition interval increases multiplicatively with respect to the value of Λ. In this case, the form of u only denies the linear recurrence behaviour in m'. In other words, it is possible for the exponential recurrence behaviour to exist in this inductive model. The following form of u represents an example of this case:

$$\mathit{\Lambda}' = \mathit{\Lambda} * 2$$

- u contains Λ and use it to update another variable : The following form of u represents an example of this case.

$$a' = a + \mathit{\Lambda}$$

> In such cases, it is evident that another variable is also updated along with the change in Λ. Therefore, it is not possible to obtain recurrent blocks with such updates in m'.

In Algorithm 5.4, when encountering the valid case, the highest common factor of all the additive increases is identified and returned as the recurrent interval δ, otherwise, the algorithm is terminated.

Guard expressions

A command can only be enabled in a state when that state satisfies the guard expression, which is a predicate, of the corresponding command. From the viewpoint of this algorithm, the guard expression is the only component of m' that allows the derivation of the recurrent borderline Ω. In Figure 5.2, a number of commands have been listed that determine the transition behaviour of the example inductive model shown in Figure 5.1. It is clear to see from Figure 5.1 that the guard of each command is a predicate and they fundamentally limit the scope of the variables of a state for which the updates of the respective command can be applied.

This algorithm only focuses on the guards that contain the recurrence variable Λ as they are the only ones capable of contributing to the derivation of Ω. Although the computation of the recurrent borderline is a difficult task comparing to the rest, the fact that all the guards are mostly simple predicates makes the computation relatively easy as they only contain comparison operators like $=, \leqslant, <, >, \geqslant$. Therefore, constructing a number line that encompasses the possible values of Λ and then plotting on it based on the guards will allow the algorithm to find the largest common region in the end, where Ω is the borderline of this largest common region. The pseudocode of computing Ω is shown in Algorithm 5.5.

The first step of computing the recurrent borderline is assigning both the lower and

Algorithm 5.5: ComputeRecurrentInterval

```
Input: $m', \Lambda$
Output: $\Omega$
$\Omega$.upper, $\Omega$.lower $\leftarrow$ $\Lambda$.upper, $\Lambda$.lower
regions $\leftarrow$ $\Omega$
for $g \in m'$ do
    if $\Lambda \in g$ then
        if $g$.operator($\Lambda$) is $\{=\}$ then
            remove related $\Omega$ from regions
            split $\Omega$ wrt. $g$.limit()
            append the split $\Omega$ to regions
        else if $g$.operator($\Lambda$) is $\{<, \leqslant\}$ then
            update related $\Omega$.upper $\in$ regions wrt. the limit
        else if $g$.operator($\Lambda$) is $\{>, \geqslant\}$ then
            update related $\Omega$.lower $\in$ regions wrt. the limit
    end
end
return max $\Omega$ from the regions
```

upper bounds of the recurrence variable Λ to Ω and then appending Ω to the regions list. The purpose of the regions list is to store all possible common regions within the scope of Λ as there can be possibly more than one recurrent region in a model. Thereafter, the algorithm loops through all the guards in the model description m' and updates the regions list only when the guard contains the recurrence variable. If the comparison operator of the expression involving Λ in the guard g is an equal operator then the related Ω is split into two separate borderlines such that the value of right hand side operand is eliminated from the regions. This step is carried out because when the equal operator is present in the expression involving Λ, the application of the corresponding command is only limited to the states with the specific value of Λ. As a consequence of this situation, the recurrence behaviour with respect to Λ is interrupted. Therefore, it is mandatory to remove this specific value of Λ from the regions list. Otherwise, the scope of the respective Ω is narrowed down based on the remaining operators. For example, if the operator is $\geqslant$ then the lower bound of Ω is updated with respect to the right hand side operand.

In this work, we assume an inductive model that encompasses the recurrence behaviour only contains a single recurrence region. However, this is not always the case in practice. Currently, we only choose the region with the largest Ω as our recurrence region when encountering many of them and the rest are considered as residing in the first or third region.

5.4.3 Construction of the regions

As we have pointed out before, the requirement for the complete exploration or construction of the original model seems to be a bottleneck in the existing minimisation algorithms for certain classes of models. The inductive models fall under these class of models. As the inductive models show a pattern in their state space growth with respect to a parameter, the algorithm leverages this knowledge to address the corresponding bottleneck. Henceforth, the state space of a linear inductive model is divided into three individual regions: first, second and third. The borderlines of each of these regions are as follows,

- **First region** : represents the state space starting from the the initial states of the given model until the entry states of the first recurrent block. In the case of example model, the first region represents $\Lambda = 0$.

- **Second region** : is the special region as it contains only the representative blocks for the entire region. This region represents the state space between the entry states of the first recurrent block and the exit states of the last recurrent block. In the case of example model, the second region represents $\Lambda \geqslant 1$ and $\Lambda < K - 1$.

- **Third region** : represents the state space after the second region. This starts from the exit states of the last recurrent block and ends at the target states of the remaining state space. In the case of example model, the third region represents $\Lambda \geqslant K - 1$ and $\Lambda \leqslant K$.

From this point onwards, this algorithm will be considered from the viewpoint of a backward approach, which computes the recurrence relations with respect to incoming transitions. Henceforth, the roles of each of these regions and the construction strategies will also be explained with respect to the backward approach. However, the difference between the backward and forward approach (computes the recurrence relations with respect to outgoing transitions) will be explained where necessary.

The construction of the first region

Initially, the first region is constructed based on the intervals listed previously. The lower bound is the initial state of the original model which is already defined in the model description m'. However, the entry states of the first recurrent block, which is the upper bound of this region, is not defined in m'. Therefore, these entry states must be defined at this stage using the recurrent borderline Ω. Henceforth, the predicate $\{\Lambda = \Omega.lower\}$ is introduced to limit the state space exploration. During the exploration, the first set of states that satisfy this predicate are the entry states of the first recurrent block. In other words, the model construction process starts the exploration from the initial state to the point where the current state satisfies this predicate.

In the backward approach, the probabilities computed from the first region are only used when forming the function $f(\lambda)$. These computed probabilities will become the coefficients of the corresponding terms in the function. On the other hand, these probabilities are considered as the initial conditions of the recurrence relations when employing the forward approach.

Figure 5.7 shows the first partial model of the example shown in Figure 5.1. Unlike the theory discussed above, the respective figure not only includes the first region ($\Lambda = 0$) but also the entry states ($\Lambda = 1$) of the first recurrent block. The only reason for the inclusion of entry states is to simplify the computation of transition probabilities from the initial state to the entry states. On the other hand, if we have chosen the exit states

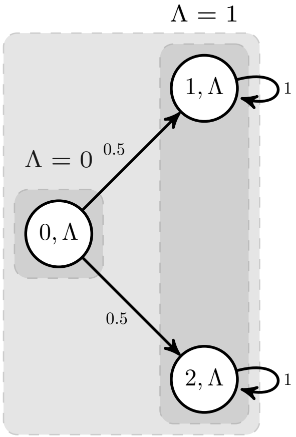


Figure 5.7: The model represents the first region and the entry states of the first recurrent block

to represent the recurrent block then this model would have included the complete first recurrent block along with the first region as the transition probabilities are computed from the initial states to the exit states.

The construction of the second region

The construction of the second region is followed immediately after the first model. It was discussed before that the the scope of this region starts from the entry states of the first recurrent block and ends at the exit states of the last recurrent block. However, unlike the previous model construction, the whole state space that lies within the second region, is not required to be constructed. In other words, this algorithm only necessitates the construction of the representative blocks for the entire region as this region demonstrates the linear recursive state space growth.

In Figure 5.8, the model that represents the second (recurrence) region of the original state space, is presented. This model is made up of $(n-2)^{th}$ and $(n-1)^{th}$ recurrent blocks, and the entry states of the $(n)^{th}$ recurrent block. In this case, the $(n-1)^{th}$ block and the entry states of the $(n)^{th}$ block are sufficient to compute the required probabilities to

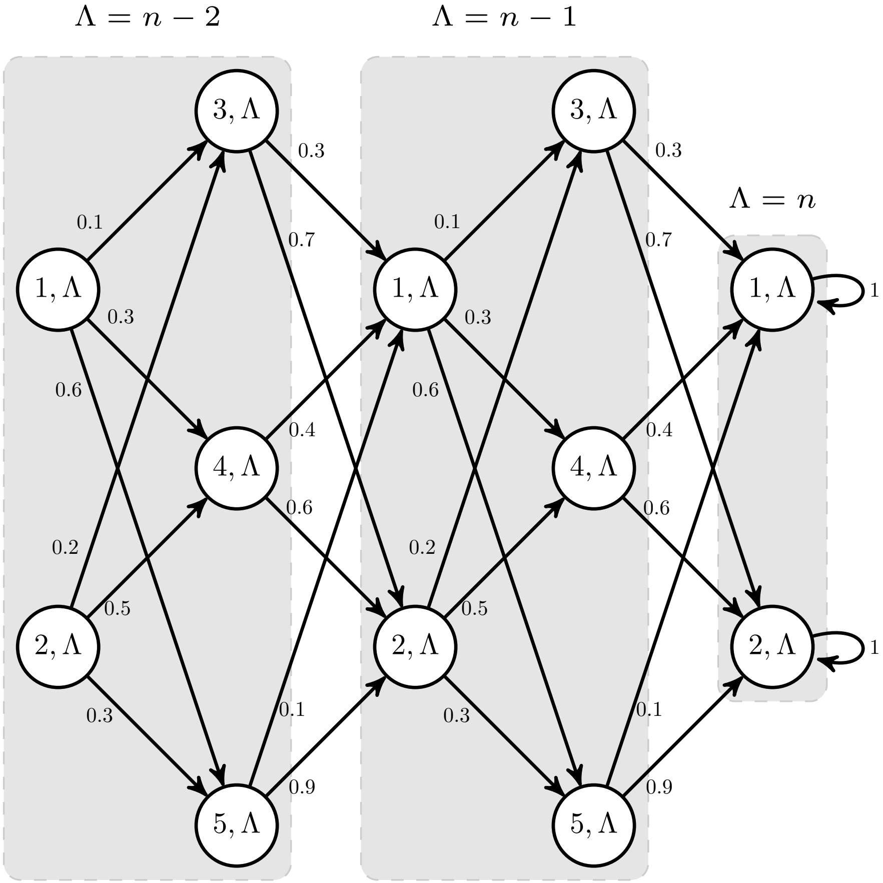


Figure 5.8: The model depicts the $(n-2)^{th}$, $(n-1)^{th}$ and $(n)^{th}$ recurrent blocks as the representative for the second region.

form the recurrence relations. The purpose of including the $(n-2)^{th}$ block is to introduce a second layer of validation that ensures a block of states are recurring with respect to Λ in the recurrent interval δ. In this validation process, this algorithm compares both the $(n-2)^{th}$ and $(n-1)^{th}$ recurrent blocks to ensure the recurrence property. Since the entry states of the first recurrent block were already defined during the previous phase, the algorithm uses that knowledge to deduce the $(n-2)^{th}$ entry states because the only difference between the recurrent blocks is the valuation of the recurrence variable Λ. The predicate $\{\Lambda = n\}$ is introduced to terminate the state space exploration as soon as the

$(n)^{th}$ entry states are encountered.

This region is considered to be the core of this algorithm due to the fact that it avoids the construction of the entire state space. The recurrence relations are formed using this region by computing the transition probabilities in between the entry states of two consecutive recurrent blocks. However, in the case when target (defined in the PCTL property) resides within the second region, this model needs to be manipulated to maintain the integrity with respect to the property of interest. Therefore, all the target states in this model are transformed into absorbing states, i.e. all the outgoing transitions of a target state are replaced by a single transition directed to itself.

The construction of the third region

The construction of the third region is very similar to the first region. However, the transition probabilities computed from this region are used as initial conditions of the extracted recurrence relations, in the backward approach. In the case of forward approach, these probabilities are considered to be the coefficients of the terms in the end function.

Since entry states are used as the representative states for recurrent blocks, the third model also includes the entire last recurrent block along with the end region to simplify the model checking. The model that represents the third region of the example model along with the $(n)^{th}$ recurrent block, is shown in Figure 5.9. Although the $(n)^{th}$ recurrent block and $(K-1)^{th}$ recurrent blocks are very similar, the transition behaviour of the exit states are different from one another. Henceforth, the states with $\Lambda = (K-1)$ are classified as the states residing in the end region.

5.5 Numerical Computation

This section discusses the formation of the end function using three crucial regions discussed in the previous section. Initially, the extraction of the relevant transition probabilities, to form the recurrence relations, is explained using the example model. The derived

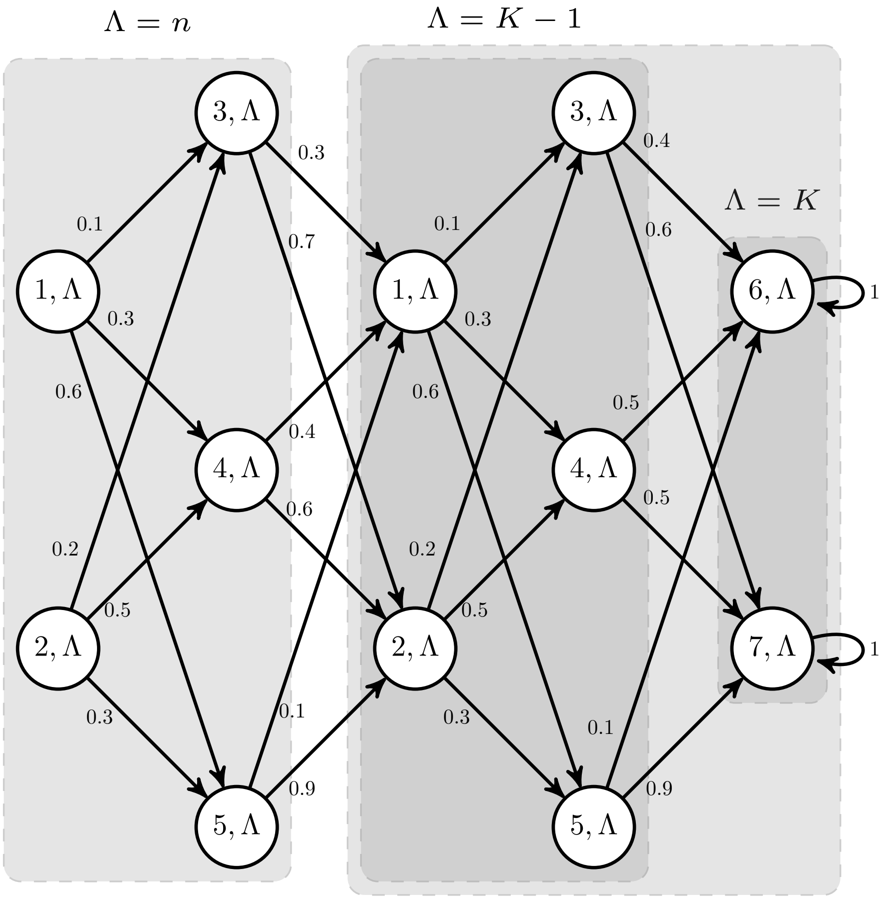


Figure 5.9: The model depicts the $(n)^{th}$ recurrent block, $(K-1)^{th}$ and $(K)^{th}$ blocks as the representative for the third region.

recurrence relations are multivariate recurrence relations. Thereafter, the mathematical computations carried out to solve these recurrence relations are explained in detail with respect to the example model. Finally, an end function is formed, which has the ability to answer the property p passed at the beginning of the algorithm for various values of recurrence parameter λ.

5.5.1 Extraction of recurrence relations

The selection of forward or backward approach influences not only the construction of the regions but also the extraction of recurrence relations. However, the focus of this discussion is about extracting the recurrence relations with respect to the backward approach. Nevertheless, the difference between both of these approaches in terms of the extraction is also briefly explained where necessary.

As a first step, the transition probabilities between two consecutive recurrent blocks are computed to form the multivariate recurrence relations. The selection of two consecutive blocks from the recurrence region as representatives is sufficient to compute the required probabilities. During the computation of the probabilities using the representatives, it is not necessary to compute the probabilities for every states in that representative blocks. Henceforth, the selection of the representative states for a recurrent block becomes mandatory. There are two possible sets of representative states for a recurrent block, which are entry and exit. In this context, the entry set refers to the states that allow incoming transitions to the current recurrent block whereas the exit set refers to the states that have outgoing transitions from the current recurrent block.

The selection of the forward or backward approach does not influence the selection of the entry or exit set as the representative. However, the number of states in the entry and exit set influences the selection of the block representative as it affects both the time for numerical computation and space for the storage of recurrence relations. In this algorithm, selecting the representative with the least number of states is the beneficial case because a recurrence relation is formed for every state in the representative set.

In the case of the example model, the number of states in the entry set is less than the exit set. Thus, the extraction process will select the entry set as the representative for all recurrent blocks. The model required for this computation is shown in Figure 5.8. From the corresponding model, both the $(n-1)^{th}$ and $(n)^{th}$ recurrent blocks are chosen for the current computation. Let $a_{(n-1)}$ be the state $(1, \Lambda)$ and $b_{(n-1)}$ be the $(2, \Lambda)$, where

$\Lambda = (n-1)$.

$$a_{n-1} = (0.21 \times a_n) + (0.79 \times b_n) \tag{1}$$

$$b_{n-1} = (0.29 \times a_n + (0.71 \times b_n) \tag{2}$$

In the equations 1 and 2, both a_n and b_n represents the entry states of the $(n)^{th}$ recurrent block and all the outgoing probabilities from $(n-1)^{th}$ entry states to $(n)^{th}$ entry states are represented by the coefficients of $(n)^{th}$ entry states. In the case of the forward approach, the equations would represent all the incoming transitions, i.e. the coefficients of the terms on the right hand side would represent the incoming transition probabilities towards the term on the left hand side. Although the recurrence relations are extracted from the example model, initial conditions have to be defined so that the exact sequence represented by the example model can be reduced into a closed form. As we discussed in the earlier section, initial conditions are derived from the third model when employing the backward approach. For both $a_{(n-1)}$ and $b_{(n-1)}$ terms, the initial condition represents the transition probability to reach the target states (typically defined in the PCTL property). According to Figure 5.9, assuming that the target state is $(6, \Lambda)$, the initial conditions are as follows.

$$a_n = 0.33$$

$$b_n = 0.32$$

where a_n and b_n represents the initial conditions for recurrence relations 1 and 2, respectively. That is the probabilities of reaching the target state $(6, K)$ from the entry states $(1, n)$ and $(2, n)$. Finally, the coefficients of each term in the final function is computed by performing model checking on the submodel shown in Figure 5.7. These coefficients represent the probabilities of reaching the entry states of the first recurrent block from the initial state. Let x_1 and x_2 be the coefficients of the terms (solutions) corresponding to the equations 1 and 2, respectively. The computed values (probabilities) of the coefficients x_1 (the probability of reaching the state $(1,1)$ from $(0,0)$) is 0.5 and x_2 (the probability of reaching the state $(2,1)$ from $(0,0)$) is 0.5.

5.5.2 Extracting recurrence relations using parametric model checking

The inductive models are split into three sub models in this algorithm with the reason of avoiding the construction of whole state space. Later, we perform model checking on these sub models to compute the required probabilities for forming the recurrence equations. One of the constraints on this implementation is the number of calls to the model checker. In the default approach, given a model and its property, the model checking is done such that it computes all probabilities $\mathtt{P}_{reach}(s, target)$ to reach the target, which is defined in the property, from every non-target state.

$$\mathtt{P}_{reach}(s, target) = \sum_{s' \in S} \mathtt{P}(s, s') \cdot \mathtt{P}_{reach}(s', target), \quad \forall s \in S \setminus target$$

The inductive model reduction classifies the targets into two types: local and global. The local targets, which refer to the representative states of the recurrent blocks (e.g. the entry states of n^{th} recurrent block in Figure 5.8), are created with the purpose of forming the recurrence relations whereas the global target refers to the target of the original mode (e.g. the state $(6, K)$ in Figure 5.9). The local targets are defined during the construction

Submodel	Initial states	Target states	Calls
First	1 or more	x	x
Second	x	x	x
Third	x	1	1

Table 5.2: The number of calls to the conventional model checker for each sub models.

of the sub models and are attached to first and second sub models. The global target lies in the third sub model. Based on this information, the number of calls to the conventional model checker is summarised in Table 5.2. In the case of local target, we are considering every state lies in it as an individual target for the purpose of computing probabilities.

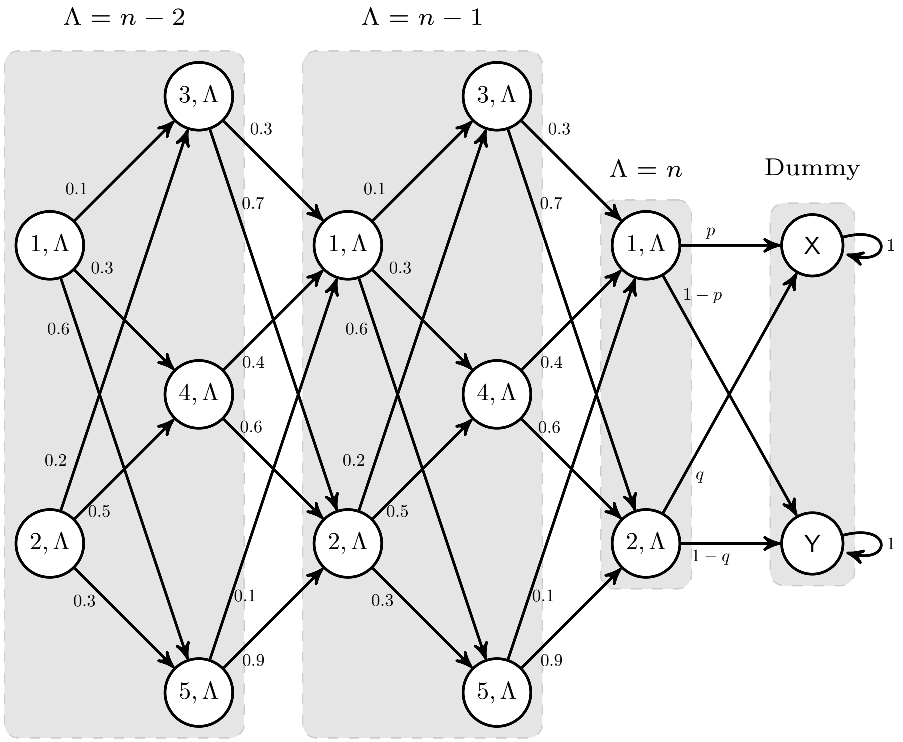


Figure 5.10: The inclusion of dummy states in the second sub model.

We can clearly see that the number of calls to the model checker is depending on the number of individual targets defined in the model. We have to reduce these calls to improve the performance of the implementation. Thus, we came up with a strategy to introduce dummy states in the first and second sub models then use the parametric model checking to tackle this issue. The inclusion of the dummy states into the second sub model is shown as an example in Figure 5.10. In this model, X and Y are the dummy states. We assign the dummy state X as the local target and the parameters p and q as the transition probabilities to reach X from the states $(1, n)$ and $(2, n)$. Henceforth, when performing the parametric model checking, the transition probability to reach the target from the state $(1, n-1)$ is resulted as follows,

$$\mathtt{P}_{reach}((1, n-1), X) = 0.21p + 0.29q$$

where coefficients of p and q represents the transition probabilities to reach the states $(1, n)$ and $(2, n)$ from the state $(1, n-1)$. This way the number of calls to the model checker has been reduced to one for all three sub models.

5.5.3 Solving the recurrence relations

As soon as all of these required computations are carried out, the process of solving the recurrence relations will start. This process leverages linear algebra and generating functions to systematically solve these recurrence relations.

In Chapter 2, we have indicated that the terms in a sequence of numbers correspond to the coefficients of a formal power series (Maclaurin series) represented by a generating function. Therefore, the generating functions can be used to solve the recurrence relations due to their potential of encoding these sequence of numbers.

In the case of inductive models, the type of extracted recurrence relations is linear homogeneous recurrence relation of degree 1 with constant coefficients. However, linear non-homogeneous recurrence relations of degree 1 with constant coefficients can also be encountered when the target lies within the second region. Nevertheless, we can encode both of these types using ordinary generating functions. The recurrence relations 1 and 2 belong to the first type and are encoded as follows. Initially, we multiply both sides of the recurrence equations 1 and 2 by x^n (For simplicity, assume the sequence moves forward, therefore, a_n becomes a_0 and vice versa).

$$a_n x^n = (0.21 \times a_{n-1} x^n) + (0.79 \times b_{n-1} x^n) \tag{3}$$

$$b_n x^n = (0.29 \times a_{n-1} x^n) + (0.71 \times b_{n-1} x^n) \tag{4}$$

Thereafter, we sum both sides of equations 3 and 4 over the same limits where we set the lower bound as possibly the smallest value and the upper bound to infinity. In this case, we have to set the lower bound of n to 1 as any lower value of n would result the subscript of a or b to be negative value on the right side of the equations.

$$\sum_{n=1}^{\infty} a_n x^n = (0.21 \times \sum_{n=1}^{\infty} a_{n-1} x^n) + (0.79 \times \sum_{n=1}^{\infty} b_{n-1} x^n) \tag{5}$$

$$\sum_{n=1}^{\infty} b_n x^n = (0.29 \times \sum_{n=1}^{\infty} a_{n-1} x^n) + (0.71 \times \sum_{n=1}^{\infty} b_{n-1} x^n) \tag{6}$$

Referring back to Section 2.7, we can see that

$$A(x) = \sum_{n=0}^{\infty} a_n x^n = \sum_{n=2}^{\infty} a_n x^n + a_1 x + a_0$$

Therefore, we can substitute an algebraic expression that includes generating function $A(x)$ and $B(x)$ for all infinite sum in the equations 5 and 6. Aftewards, we have to factor the terms on the right of each infinite sum such that the power of x in them matches the lower bound of the infinite summation.

$$\sum_{n=1}^{\infty} a_{n-1} x^{n-1} - a_0 = (0.21x \times \sum_{n=1}^{\infty} a_{n-1} x^{n-1}) + (0.79x \times \sum_{n=1}^{\infty} b_{n-1} x^{n-1})$$

$$\sum_{n=1}^{\infty} b_{n-1} x^{n-1} - b_0 = (0.29x \times \sum_{n=1}^{\infty} a_{n-1} x^{n-1}) + (0.71x \times \sum_{n=1}^{\infty} b_{n-1} x^{n-1})$$

After the factorisation, we replace all the infinite sums with their corresponding generating functions.

$$A(x) - a_0 = (0.21x \times A(x)) + (0.79x \times B(x)) \tag{7}$$

$$B(x) - b_0 = (0.29x \times A(x)) + (0.71x \times B(x)) \tag{8}$$

In equations 7 and 8, the terms a_0 and b_0 denotes the initial conditions of a sequence belongs to $A(x)$ and $B(x)$, respectively. Since we considered that these sequences move forward for the simplicity of calculation, a_0 and b_0 refer to the a_n and b_n which we have computed previously in Subsection 5.5.1.

We rearrange these equations in the following matrix form, so that Gaussian elimination can be applied to solve these linear equations.

$$\begin{array}{cc|c} \scriptstyle A(x) & \scriptstyle B(x) & \\ 0.21x - 1 & 0.79x & -a_0 \\ 0.29x & 0.71x - 1 & -b_0 \end{array}$$

As a result of applying Gaussian elimination, we end up with a rational function for both $A(x)$ and $B(x)$, where the polynomial is expressed in terms of x. The corresponding rational function has to be decomposed into a sum of polynomial fractions with simple denominators such that all partial fractions are in the following form $\frac{I}{(1-Rx)}$. After the partial fraction decomposition, the closed forms for a_n and b_n are derived directly from these partial fractions. The simplified solutions of the recurrence relations 1 and 2 are given below.

$$a_n = 0.32(1)^n - 0.07(-0.08)^n$$

$$b_n = 0.32(1)^n + 0.02(-0.08)^n$$

Finally, these solutions are used to form the end function that can answer the property p for various values of K, which is the recurrence parameter λ. The end function of the example model with respect to p is in the following form,

$$F(n) = 0.5\ (0.32 - 0.07(-0.08)^n) + 0.5\ (0.32 + 0.02(-0.08)^n)$$
$$F(n) = 0.32 - 0.045\ (0.08)^n \tag{9}$$

where $n = (K - 2)$ and $K \geqslant 2$. The reason for K is not allowed to be less than 2 is because the recurrence region would not exist otherwise.

5.6 Experimental Results and Analysis

The inductive model reduction technique presented in this chapter is also implemented as an extension of the PRISM model checker. This implementation has been applied to both the *NAND multiplexing* [81] and synchronous leader election protocol [48] DTMC models from PRISM's benchmark suite [65]. The PRISM model checker has various engines to perform model checking. The implementation of the presented algorithm leverages the *hybrid* and *parametric* engines to perform model checking on all three crucial sub-models. All the experiments are carried out on a MacBook Pro, Late 2013 with Core i5 2.4 GHz processor and 8GB of RAM.

The investigation with regards to this implementation has been divided into two sections, see Section 5.5.2. First, we use both *hybrid* and *parametric* engines individually on this implementation to compare the positive and negative contributions on the performance of this implementation. We have introduced another example model with the purpose of using it for testing the performances of the engines.

Afterwards, we compare the backward approach variant against conventional model checking with and without the application of bisimulation minimisation to clearly show the advantage of the inductive model reduction in the context of probabilistic verification. The *NAND multiplexing* model has two parameters N and K that represent the number of inputs in each bundle and number of restorative stages. We have selected this model for the corresponding experiment because it is a linear inductive model with respect to its parameter K and encompasses the recurrence behaviour.

Apart from the *NAND multiplexing* model, we also use the DTMC model that describes the synchronous leader election protocol. However, this DTMC model is not a suitable candidate as it does not encompass the recurrence behaviour with respect to any of its parameters. Nevertheless, we are using this model for the purpose of verifying the cost-bounded properties.

5.6.1 Parametric vs. Conventional Model Checking

We have generated a few DTMC models with different number of entry states and performed model checking on them using both *hybrid* and *parametric* engines to measure the performance of the respective engines. These DTMC models have the following properties : same number of entry and exit states, all the entry states have outgoing transitions to all the exit states and every exit state has outgoing transitions towards the two dummy states. The results for the experiment are shown in Table 5.3. In the results, the time for the *hybrid* engine represents the total time taken for n number of calls, where n represents the number of entry states. During this experiment, the time-out (TO) was set as 20 seconds. Although the number of calls to the model checker was reduced to one when using the *parametric* engine, its performance was poor compare to the *hybrid* engine. The actual cause of the poor performance is the use of big rational data structure for computations. Therefore, we have decided to leverage the *hybrid* engine over the *parametric* engine.

Exit/Entry States	Transitions		Time(s)	
	Original	Dummy	Hybrid	Parametric
10	100	20	0.08	0.33
11	121	22	0.09	0.6
12	144	24	0.1	1
13	169	26	0.11	1.49
14	196	28	0.11	4
15	225	30	0.13	10
16	256	32	0.13	TO

Table 5.3: The model checking comparison between parametric and hybrid engines of the PRISM model checker

5.6.2 Inductive Model Reduction

First, we compare the performance and scalability of inductive model reduction technique against conventional model checking. In this experimental study, we are using the *hybrid* engine of the PRISM model checker for the conventional approach as it leads to overall best performance in practice (thus, used as default engine in the PRISM model checker). Table 5.4 shows the experimental results of applying these approaches on the *NAND multiplexing* DTMC model. The sub columns 'PRISM' and 'Inductive' of the column 'States' represent the state space of the original model and the required state space for the inductive model reduction, respectively. Meanwhile, the sub columns 'PRISM' and 'Inductive' under the column 'Time' shows the total time taken to yield the model checking result for the respective approaches. In this experiment, the time is shown in seconds and the time-out has been set to 10 minutes.

The size of the original state space varies with respect to both parameters K and N. In contrast, the size of the state space constructed by our technique is a constant for all values of K attached with a particular N, due to the fact that K is the recurrence parameter of *NAND multiplexing* model. We can clearly observe that inductive model reduction technique yields a huge reduction in the state space for larger values of K as it only explores the necessary states space for its computations. Henceforth, we can state that our novel approach is not susceptible for the state space explosion for larger values of K. The inductive model reduction technique involves very expensive computations to derive an end function for a given reachability property in the form of $\mathtt{P}_{=?}[\mathtt{F\ target}]$. Thus, this approach is slower compare to the conventional model checking for smaller values of the recurrence parameter. As the performance of our approach is independent of the value of recurrence parameter, it dominates the conventional model checking for larger values of K which is evident from the results shown in Table 5.4.

Parameters		States		Time(s)	
N	K	PRISM	Inductive	PRISM	Inductive
4	1600	340677	1165	113.74	4.56
	1800	383277		145.00	
	2000	425877		173.26	
5	1600	638133	2139	316.17	19.82
	1800	717933		435.66	
	2000	797733		510.50	
6	1200	823895	554	390.04	136.23
	1400	961295		508.01	
	1600	1098695		TO	

Table 5.4: The experimental results of PRISM and inductive model reduction on *NAND multiplexing* DTMC model

Secondly, we compare the performance and scalability of the current approach against model checking that leverages bisimulation minimisation. Table 5.5 shows the experimental results for employing the aforementioned approaches to perform model checking on the *Nand multiplexing* model. As the bisimulation minimisation technique is only implemented in the PRISM's explicit engine, we are using that engine to perform model reduction. We have claimed earlier that the model reduction gained from the bisimulation minimisation technique is not always plausible for certain models. This claim becomes evident in the case of *Nand multiplexing* model as the reduction is only 30% of the original state space. In addition to this, the performance of the inductive model reduction technique also significantly better over the bisimulation minimisation approach for this class of models.

Parameters		States	Reduction		Time(s)	
N	K		Bisim	Inductive	Bisim	Inductive
3	400	39158	28337	554	53.33	2.65
	500	48958	35437		78.51	
	600	58758	42537		107.15	
4	400	85077	58638	1165	157.22	4.56
	500	106377	73338		215.30	
	600	127677	88038		306.73	
5	400	159333	110052	2139	331.11	19.82
	500	199233	137652		549.20	
	600	239133	165252		TO	

Table 5.5: The experimental results of bisimulation minimisation and inductive model reduction on *NAND multiplexing* DTMC model

5.6.3 Inductive Model Reduction on Reward/Cost Models

The novel technique presented in this chapter not only applicable on inductive models but also on models that have been preprocessed into inductive form to solve a cost-bounded property. We have discussed in Section 5.4.1 about preprocessing these DTMC models. The preprocessing phase results an inductively growing DTMC model that can be used to verify the cost-bounded property in the following form.

$$\mathtt{P}_{=?}[\mathtt{F}_{\leqslant \mathsf{cost}}\ \mathsf{target}]$$

The corresponding property is interpreted as "the probability of reaching the target within the given cost limit". The experiments for this special case were carried out on the *Leader Sync* DTMC model. The results, compared against the conventional model checking and the model checking with bisimulation minimisation, were shown in Table 5.6 and Table 5.7, respectively. The columns of the Tables represent the exact meaning as in the previous section.

From Table 5.6, it is evident that the inductive model reduction outperforms the conventional model checking for larger values of the recurrence parameter. In the case when $N = 5$, $K = 5$ and $\mathsf{Costs} = 2000$, the size of the original state space has reached around 25 million, henceforth, the conventional model checker has timed-out. On the other hand, the inductive model reduction technique constructed a significantly smaller state space. However, the bisimulation minimisation yielded a better reduction comparing to our technique due to the fact that the original DTMC is already a greatly reducible model. However, the implementation of bisimulation minimisation runs out of memory (MO) when the state space of the preprocessed model reaches 1 million states whereas our technique was able to handle it.

Our technique can also produce an end function for the time-bounded properties of a typical DTMC by preprocessing it in the similar way. Unlike the previous case, the PRISM model checker already has a way to verify the time-bounded properties using the original model, which outperforms the current approach. However, we hope that the end function produced by the inductive model reduction technique can be used for mathematical analysis on the original with respect to the selected time-bounded property.

Parameters			States		Time(s)	
N	K	Costs	PRISM	Inductive	PRISM	Inductive
4	3	1000	274001	987	8.2	2.62
		1500	411001		12.62	
		2000	548001		16.8	
4	5	1000	1933001	7052	42.019	7
		1500	2899501		64.72	
		2000	3866001		98.25	
5	5	1000	12709001	44380	438.7	30.45
		1500	19063501		599.3	
		2000	25418001		TO	

Table 5.6: The experimental results of PRISM and inductive model reduction on *Leader Sync* DTMC model

Parameters			States	Reduction		Time(s)	
N	K	Costs		Bisim	Inductive	Bisim	Inductive
4	5	400	773201	302	7052	42.019	7
		500	966501	452		64.72	
		600	1159801	-		MO	
5	5	50	635451	2002	44380	438.7	30.45
		75	953176	2502		599.3	
		100	1270901	-		MO	

Table 5.7: The experimental results of bisimulation minimisation and inductive model reduction on *Leader Sync* DTMC model

5.7 Summary

In this chapter, we have presented an inductive model reduction technique that focuses on linear inductive DTMCs. We have introduced methods to automatically detect and extract such models from a high-level modelling description, and then forming recurrence relations from the extracted models. The solutions of these recurrence relations are used to form an end function of recurrence parameter which can be reused to perform model checking for various values of the recurrence parameter, without any further construction of the state space. We also show how the inductive model reduction can be extended to verify step-bounded and cost-bounded reachability properties on arbitrary DTMCs, which reduces to the problem of verifying linear inductive DTMCs.

CHAPTER 6

Conclusions

6.1 Summary and Evaluation

The main aim of this thesis was to develop scalable and efficient model reduction techniques for probabilistic model checking. In particular, we focused on verifying reachability properties of probabilistic models such as step-bounded, cost-bounded and unbounded properties. The techniques presented in this thesis avoid explicit construction of the original state space and therefore, the scalability of these techniques relies mainly on the size of the reduced model. We present two different ways to construct the reduced model directly from the high-level modelling language, in this case PRISM language, such that the reachability properties can be verified. This way we support the probabilistic model checking to cope with the state space explosion problem.

Another aim of this thesis was to make these technique available in practice so that they can be applied on real world problems. Hence, we have implemented them as an extension of the PRISM model checker. These implementations have been evaluated against a number of benchmark models.

The main contributions of this thesis have been presented in Chapter 4 and Chapter 5.

Below, we summarise each of these contributions.

In Chapter 4, we have proposed *finite-horizon* bisimulation minimisation technique, which is a variant of probabilistic bisimulation, with the aim of reducing the model more aggressively. This technique preserves stepwise behaviour over a finite number of steps.We have formalised this variant and defined the subset of PCTL properties that it preserves. We have presented a partition-refinement based algorithm for computing the coarsest finite-horizon bisimulation quotient. We have also proposed two versions of an on-the-fly approach, symbolic (based on SMT solvers) and explicit-state, which prevents the construction of full Markov chain prior to minimisation, unlike the former. Finally, we have applied the on-the-fly approach to a class of problems: models with a large number of possible initial configurations, and showed that finite-horizon bisimulation can provide significant gains in both verification time and scalability.

Many bisimulation minimisation algorithms [29, 80, 25] have been presented in the context of probabilistic verification, however, they all focus on the complete minimisation. We believe, to the best of our knowledge, we are the first to consider minimisation in the finite-horizon setting.

In Chapter 5, we have presented an *inductive model reduction* technique, which focuses on a class of models whose state space linearly grows with respect to a parameter, which is referred to as a recurrence parameter. This technique considers reachability properties such as step-bounded, cost-bounded and unbounded. The main aim of this work is to form an end function, for a given linear inductive DTMC, corresponding to the recurrence parameter such that it can be reused on the expanded DTMCs. We devise methods that automatically detect and extract such models from a high-level mode description, and then perform model checking via construction and solution of a set of recurrence relations. We show that the inductive model reduction technique outperforms conventional model checking with and without the application of bisimulation minimisation. The experimental results also show that this technique is scalable as it requires the same segments of the original state space for the computation for any values of the recurrence parameter. Finally, we also show that this technique is further extended to verify cost-bounded and step-bounded properties by preprocessing the model description of the original model.

6.2 Future Work

The work presented in this thesis has several limitations and can be extended in a number of ways. In this section, we discuss possible extensions of the work presented in Chapter 4 and Chapter 5.

The model reduction techniques presented in Chapter 4 are currently considering only DTMCs. As one of the possible future extensions, these techniques can be extended to other classes of probabilistic models, e.g. CTMCs and MDPs. Moreover, the symbolic variant of the on-the-fly approach does not perform very well due to the fact that the computation of detecting the intersection between partitions is expensive. We believe that there must be a better way to present the symbolic variant as it can provide potentially a good reduction in state space using the symbolic representation. Finally, we show that the finite-horizon bisimulation preserves partially the step-bounded fragment of PCTL. As a future step, the finite-horizon bisimulation can be adapted to preserve the full step-bounded fragment of PCTL, including nested formulae.

In Chapter 5, we again focus on DTMC models. The techniques presented in this chapter can be also extended for other classes of probabilistic models such as CTMCs and MDPs. We first assume that there is only one recurrence variable in the inductive model that depends on the recurrence parameter; thus, we do not handle the case where more than one variable depends on the recurrence parameter. We believe this could lead to non-linear recurrence behaviour and can be investigated further to reveal the actual underlying problem. Moreover, we also assume that the recurrence behaviour can only be imposed on the inductive models by a single recurrence parameter. Henceforth, the end function is derived only with respect to that particular parameter. We think that the current approach can be extended with respect to this perspective such that the end function depends on more than one recurrence parameters.

6.3 Conclusion

In conclusion, we have successfully introduced two different techniques that alleviate the impact of state space explosion problem in the context of probabilistic verification. We have formally defined each of these techniques, implemented them as an extension of the PRISM model checker and evaluated them over standard benchmarks.

We have also showed that our work outperforms the implementations of the PRISM model checker (state-of-the-art techniques) for specific class of probabilistic models. Finally, We have proposed number of possible directions in which our work can be extended.

APPENDIX A

SMT Queries for Tournament Game

We illustrate the SMT-based approach with an example. We use a PRISM model of a *Tournament* game, which comprises K particles labelled with values from a range $0, \ldots, N-1$. Particles interact at random, and when doing so, the particle with the larger state value wins, and copies its value to the other. Figure A.1 shows the PRISM model, for $N = 3, K = 5$. For modelling convenience, we actually describe the model as a continuous-time Markov chain (CTMC), and consider its embedded Markov chain as the DTMC to be analysed.

Figure A.2 shows an example of an SMT query that is constructed for the input target expression $(c2 = 5)$ & $(c0 + c1 = 0)$ and for the following command:

$$c1 > 0 \;\&\; c2 > 0 \;\&\; c2 < K \;\rightarrow\; 2 \times c1 \times c2 : (c1' = c1 - 1) \;\&\; (c2' = c2 + 1)$$

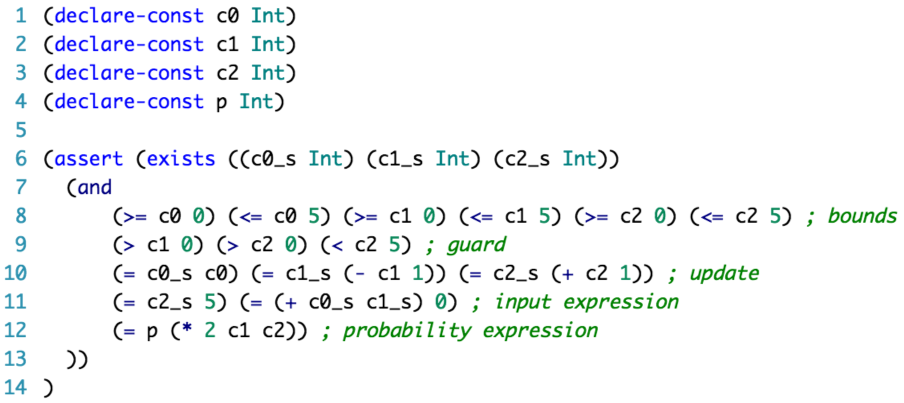
In the model, variable ci counts the number of particles in state i. The command represents the interaction between particles in states 1 and 2.

The SMT query above will be dispatched as an input to the solver to check for its satisfiability. If this query is satisfiable, a value for p will be retrieved from the solution

```
ctmc

// Number of fitness levels: N
const int N = 3;

// Total number of agents/particles: K
const int K = 5;

module tournament

    // Counters: ci = number of agents/particles with fitness i
    c0 : [0..K];
    c1 : [0..K];
    c2 : [0..K];

    // Possible reactions between agents/particles
    // Each possible pairwise collision
    [r01] c0>0 & c1>0 & c1<K → 2 * c0 * c1 : (c0'=c0 − 1) & (c1'=c1 + 1);
    [r02] c0>0 & c2>0 & c2<K → 2 * c0 * c2 : (c0'=c0 − 1) & (c2'=c2 + 1);
    [r12] c1>0 & c2>0 & c2<K → 2 * c1 * c2 : (c1'=c1 − 1) & (c2'=c2 + 1);
    // Collision between 2 identical agents/particles
    [r00] c0>1 → c0 * (c0 − 1) : true;
    [r11] c1>1 → c1 * (c1 − 1) : true;
    [r22] c2>1 → c2 * (c2 − 1) : true;

endmodule

// Initial states
init c0 + c1 + c2=K & c2>0 endinit

// Labels (atomic propositions) for properties:

// Finished: all agents/particles have maximum fitness
label "done" = c2⩾K;
label "target" = c2=K & c0 + c1=0;

// Reward structure used to reason about passage of time (discrete steps)
rewards "time"
   true : 1;
endrewards
```

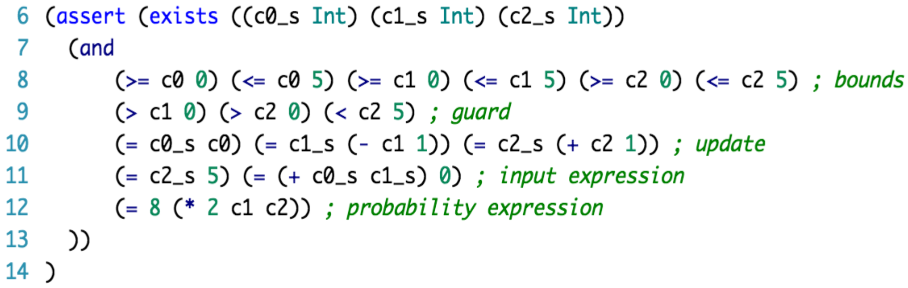
Figure A.1: PRISM modelling language description of the *Tournament* game ($N = 3$).

produced by the solver. As presented in Figure A.3, an expression for a set of predecessor states that satisfies the value p will be generated. Afterwards, the initial SMT query will be updated as shown in Figure A.4, i.e., *not* (= p 8) is used as a blocking expression, to rule out the previous solution.



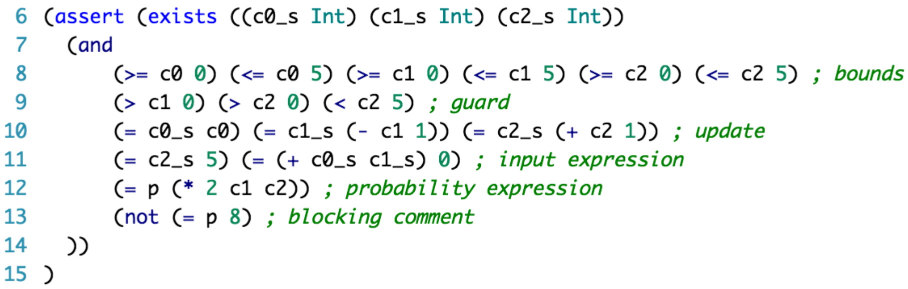
```
(declare-const c0 Int)
(declare-const c1 Int)
(declare-const c2 Int)
(declare-const p Int)

(assert (exists ((c0_s Int) (c1_s Int) (c2_s Int))
  (and
      (>= c0 0) (<= c0 5) (>= c1 0) (<= c1 5) (>= c2 0) (<= c2 5) ; bounds
      (> c1 0) (> c2 0) (< c2 5) ; guard
      (= c0_s c0) (= c1_s (- c1 1)) (= c2_s (+ c2 1)) ; update
      (= c2_s 5) (= (+ c0_s c1_s) 0) ; input expression
      (= p (* 2 c1 c2)) ; probability expression
  ))
)
```

Figure A.2: SMT query representing a guarded command



```
(assert (exists ((c0_s Int) (c1_s Int) (c2_s Int))
  (and
      (>= c0 0) (<= c0 5) (>= c1 0) (<= c1 5) (>= c2 0) (<= c2 5) ; bounds
      (> c1 0) (> c2 0) (< c2 5) ; guard
      (= c0_s c0) (= c1_s (- c1 1)) (= c2_s (+ c2 1)) ; update
      (= c2_s 5) (= (+ c0_s c1_s) 0) ; input expression
      (= 8 (* 2 c1 c2)) ; probability expression
  ))
)
```

Figure A.3: SMT query to find predecessors for a specific probability value



```
(assert (exists ((c0_s Int) (c1_s Int) (c2_s Int))
  (and
      (>= c0 0) (<= c0 5) (>= c1 0) (<= c1 5) (>= c2 0) (<= c2 5) ; bounds
      (> c1 0) (> c2 0) (< c2 5) ; guard
      (= c0_s c0) (= c1_s (- c1 1)) (= c2_s (+ c2 1)) ; update
      (= c2_s 5) (= (+ c0_s c1_s) 0) ; input expression
      (= p (* 2 c1 c2)) ; probability expression
      (not (= p 8) ; blocking comment
  ))
)
```

Figure A.4: SMT query updated with a blocking expression to find further matches

List of References

[1] H. Aljazzar, M. Fischer, L. Grunske, M. Kuntz, F. Leitner, and S. Leue. Safety analysis of an airbag system using probabilistic FMEA and probabilistic counterexamples. In *Proc. QEST'09*, 2009. One citation in section 1.

[2] Rajeev Alur, Robert K Brayton, Thomas A Henzinger, Shaz Qadeer, and Sriram K Rajamani. Partial-order reduction in symbolic state space exploration. In *International Conference on Computer Aided Verification*, pages 340–351. Springer, 1997. One citation in section 3.

[3] S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model-checked. In *Proc. Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *LNCS*, pages 88–104. Springer, 2003. One citation in section 2.2.

[4] Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008. One citation in section 4.4.

[5] Adnan Aziz, Vigyan Singhal, Felice Balarin, Robert K Brayton, and Alberto L Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *CAV'95*, 1995. 2 citations in sections 2.4 and 3.1.2.

[6] Giorgio Bacci, Giovanni Bacci, Kim G Larsen, and Radu Mardare. On-the-fly exact computation of bisimilarity distances. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 1–15. Springer, 2013. One citation in section 3.2.

[7] Christel Baier, Edmund M Clarke, Vasiliki Hartonas-Garmhausen, Marta Kwiatkowska, and Mark Ryan. *Symbolic model checking for probabilistic processes.* Springer, 1997. One citation in section 3.1.2.

[8] Christel Baier, Bettina Engelen, and Mila Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *Journal of Computer and System Sciences*, 60(1):187–231, 2000. One citation in section 3.1.2.

[9] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT press, 2008. One citation in section 3.

[10] Ahmed Bouajjani, Jean-Claude Fernandez, and Nicolas Halbwachs. Minimal model generation. In *Computer-Aided Verification*, pages 197–203. Springer, 1991. 2 citations in sections 1 and 3.1.1.

[11] L. Brim, M. Češka, and David Šafránek S. Dražan. Exploring parameter space of stochastic biochemical systems using quantitative model checking. In *Proc. 25th Int. Conf. Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 107–123. Springer, 2013. One citation in section 3.4.

[12] Stefano Cattani and Roberto Segala. Decision algorithms for probabilistic bisimulation*. In *CONCUR 2002—Concurrency Theory*, pages 371–386. Springer, 2002. One citation in section 3.1.2.

[13] M. Češka, F. Dannenberg, M. Kwiatkowska, and N. Paoletti. Precise parameter synthesis for stochastic biochemical systems. In P. Mendes, J. Dada, and K. Smallbone, editors, *Proc. 12th Int. Conf. Computational Methods in Systems Biology (CMSB'14)*, volume 8859 of *LNCS/LNBI*, pages 86–98. Springer, 2014. One citation in section 3.4.

[14] Di Chen, Franck van Breugel, and James Worrell. On the complexity of computing probabilistic bisimilarity. In *Foundations of Software Science and Computational Structures*, pages 437–451. Springer, 2012. One citation in section 3.2.

[15] Edmund Clarke, Orna Grumberg, Hiromi Hiraishi, Somesh Jha, David Long, Kenneth McMillan, and Linda Ness. Verification of the Futurebus+ cache coherence protocol. Technical report, DTIC Document, 1992. One citation in section 1.

[16] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer aided verification*, pages 154–169. Springer, 2000. 2 citations in sections 3 and 3.2.

[17] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking.* MIT press, 1999. One citation in section 1.

[18] Christopher L Conway, Kedar S Namjoshi, Dennis Dams, and Stephen A Edwards. Incremental algorithms for inter-procedural analysis of safety properties. In *CAV*, volume 5, pages 449–461. Springer, 2005. One citation in section 3.3.

[19] C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. In Z. Liu and K. Araki, editors, *Proc. 1st Int Coll. Theoretical Aspects of Computing (ICTAC'04)*, volume 3407 of *LNCS*, pages 280–294. Springer, 2004. 2 citations in sections 3.3 and 3.4.

[20] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proc. TACAS'08*, 2008. 2 citations in sections 3.1.2 and 4.4.

[21] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: Introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011. One citation in section 9.

[22] Leonardo De Moura and Grant Passmore. *The Strategy Challenge in SMT Solving*, volume 7788 of *Lecture Notes in Computer Science*, pages 15–44. Springer Berlin Heidelberg, 2013. One citation in section 3.1.2.

[23] Thomas Dean, Robert Givan, and Sonia Leach. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pages 124–131. Morgan Kaufmann Publishers Inc., 1997. One citation in section 3.2.

[24] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J-P. Katoen, and E. Ábrahám. PROPhESY: A PRObabilistic ParamEter SYnthesis tool. In *Proc. 27th Int. Conf. Computer Aided Verification (CAV'15)*, volume 9206 of *LNCS*, pages 214–231. Springer, 2015. One citation in section 3.3.

[25] Christian Dehnert, Joost-Pieter Katoen, and David Parker. SMT-based bisimulation minimisation of Markov models. In *Proc. VMCAI'13*, pages 28–47, 2013. 4 citations in sections 3.1.2, 3.4, 18, and 6.1.

[26] Giuseppe Della Penna, Benedetto Intrigila, Igor Melatti, Enrico Tronci, and Marisa Venturini Zilli. Finite horizon analysis of markov chains with the murϕ verifier. *STTT*, 8(4-5):397–409, 2006. One citation in section 3.4.

[27] S. Derisavi. Signature-based symbolic algorithm for optimal Markov chain lumping. In *Proc. QEST'07*, pages 141–150. IEEE Computer Society, 2007. One citation in section 4.2.1.

[28] Salem Derisavi. *Solution of Large Markov Models Using Lumping Techniques and Symbolic Data Structures*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2005. AAI3198968. One citation in section 3.1.2.

[29] Salem Derisavi, Holger Hermanns, and William H Sanders. Optimal state-space lumping in Markov chains. *Information Processing Letters*, 87(6):309–315, 2003. 8 citations in sections 1, 3, 3.1.1, 3.1.2, 3.1.2, 3.4, 4.2.1, and 6.1.

[30] Salem Derisavi, Peter Kemper, and William H Sanders. Lumping matrix diagram representations of Markov models. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 742–751. IEEE, 2005. One citation in section 1.

[31] Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004. One citation in section 3.2.

[32] Agostino Dovier, Carla Piazza, and Alberto Policriti. An efficient algorithm for computing bisimulation equivalence. *Theoretical Computer Science*, 311(1):221–256, 2004. One citation in section 3.1.1.

[33] Jean-Claude Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13(2):219–236, 1990. One citation in section 3.1.1.

[34] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite Markov decision processes. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 162–169. AUAI Press, 2004. One citation in section 3.2.

[35] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *Proc. 33rd ACM/IEEE Int. Conf. Software Engineering (ICSE'11)*. ACM, 2011. One citation in section 3.4.

[36] Kathi Fisler and Moshe Y Vardi. Bisimulation and model checking. In *Correct Hardware Design and Verification Methods*, pages 338–342. Springer, 1999. One citation in section 3.1.1.

[37] V. Forejt, M. Kwiatkowska, D. Parker, H. Qu, and M. Ujma. Incremental runtime verification of probabilistic systems. In S. Qadeer and S. Tasiran, editors, *Proc. 3rd Int. Conf. Runtime Verification (RV'12)*, volume 7687 of *LNCS*, pages 314–319. Springer, 2012. 2 citations in sections 3.3 and 3.4.

[38] Vojtech Forejt, Marta Z Kwiatkowska, Gethin Norman, and David Parker. Automated verification techniques for probabilistic systems. In *SFM*, volume 11, pages 53–113. Springer, 2011. One citation in section 2.3.

[39] Alessandro Giacalone, Chi-Chang Jou, and Scott A Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proc. IFIP TC2 Working Conference on Programming Concepts and Methods*. Citeseer, 1990. One citation in section 3.2.

[40] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. PARAM: A model checker for parametric Markov models. In *Proc. 22nd Int. Conf. Computer Aided Verification (CAV'10)*, volume 6174 of *LNCS*, pages 660–664. Springer, 2010. One citation in section 3.3.

[41] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric markov models. *International Journal on Software Tools for Technology Transfer*, 13(1):3–19, 2011. One citation in section 3.3.

[42] T. Han, J-P. Katoen, and A. Mereacre. Approximate parameter synthesis for probabilistic time-bounded reachability. In *Proc. IEEE Real-Time Systems Symp. (RTSS 08)*, pages 173–182. IEEE Computer Society Press, 2008. One citation in section 3.4.

[43] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *FAC*, 6(5):512–535, 1994. 3 citations in sections 2.2, 4, and 4.1.1.

[44] Keijo Heljanko, Tommi A Junttila, and Timo Latvala. Incremental and complete bounded model checking for full pltl. In *CAV*, volume 5, pages 98–111. Springer, 2005. One citation in section 3.3.

[45] Thomas A Henzinger, Ranjit Jhala, Rupak Majumdar, and Marco AA Sanvido. Extreme model checking. In *Verification: Theory and Practice*, pages 332–358. Springer, 2003. One citation in section 3.3.

[46] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical report, DTIC Document, 1971. One citation in section 3.1.1.

[47] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *ACM SIGACT News*, 32(1):60–65, 2001. One citation in section 3.3.

[48] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1), 1990. One citation in section 5.6.

[49] Nils Jansen, Florian Corzilius, Matthias Volk, Ralf Wimmer, Erika ÃĄbrahÃąm, Joost-Pieter Katoen, and Bernd Becker. Accelerating parametric probabilistic verification. In *Proc. 11th Int. Conf. Quantitative Evaluation of Systems (QEST'14)*, pages 404–420, 2014. One citation in section 3.3.

[50] Nishanthan Kamaleson, David Parker, and Jonathan E Rowe. Finite-horizon bisimulation minimisation for probabilistic systems. In *International Symposium on Model Checking Software*, pages 147–164. Springer, 2016. One citation in section 1.1.

[51] Paris C Kanellakis and Scott A Smolka. Ccs expressions, finite state processes, and three problems of equivalence. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 228–240. ACM, 1983. One citation in section 3.1.1.

[52] J.-P. Katoen, T. Kemna, I. Zapreev, and D. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In *Proc. TACAS'07*, volume 4424, 2007. 2 citations in sections 3 and 4.

[53] Joost-Pieter Katoen. 22 labelled transition systems. *Model-Based Testing of Reactive Systems*, page 615, 2005. One citation in section 3.1.

[54] Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf. Three-valued abstraction for continuous-time Markov chains. In *Computer Aided Verification*, pages 311–324. Springer, 2007. 2 citations in sections 3 and 3.2.

[55] Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf. *Abstraction for stochastic systems by ErlangâĂŹs method of stages*. Springer, 2008. 2 citations in sections 1 and 3.2.

[56] Joost-Pieter Katoen, Ivan S Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N Jansen. The ins and outs of the probabilistic model checker MRMC. *Perform. eval.*, 68(2):90–104, 2011. 2 citations in sections 1 and 3.1.2.

[57] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. Springer-Verlag, 2nd edition, 1976. One citation in section 2.1.

[58] D. Klink, A. Remke, B. Haverkort, and J-P. Katoen. Time-bounded reachability in tree-structured QBDs by abstraction. In *Proc. 6th Int. Conf. Quantitative Evaluation of Systems (QEST'09)*, pages 133–142. IEEE Computer Society Press, 2009. One citation in section 3.4.

[59] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proc. CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011. 5 citations in sections 1, 3.1.2, 3.3, 4.3, and 4.4.

[60] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 205(7), 2007. One citation in section 3.4.

[61] M. Kwiatkowska, D. Parker, H. Qu, and M. Ujma. On incremental quantitative verification for probabilistic systems. In Andrei Voronkov and Margarita Korovina, editors, *HOWARD-60: A Festschrift on the Occasion of Howard Barringer's 60th Birthday*, pages 245–257. Easychair, 2014. One citation in section 3.3.

[62] Marta Kwiatkowska, Gethin Norman, and David Parker. Symmetry reduction for probabilistic model checking. In *Computer Aided Verification*, pages 234–248. Springer, 2006. 2 citations in sections 1 and 3.1.2.

[63] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 220–270. Springer, 2007. One citation in section 2.2.

[64] Marta Kwiatkowska, Gethin Norman, and David Parker. Advances and challenges of probabilistic model checking. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 1691–1698. IEEE, 2010. One citation in section 2.1.

[65] Marta Kwiatkowska, Gethin Norman, and David Parker. The PRISM benchmark suite. In *Proc. QEST'12*, pages 203–204, 2012. 2 citations in sections 4.4 and 5.6.

[66] Marta Kwiatkowska, David Parker, and Hongyang Qu. Incremental quantitative verification for markov decision processes. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 359–370. IEEE, 2011. One citation in section 3.3.

[67] Kim G Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991. 4 citations in sections 2.4, 3.1.2, 4, and 4.1.

[68] David Lee and Mihalis Yannakakis. Online minimization of transition systems. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 264–274. ACM, 1992. 2 citations in sections 1 and 3.1.1.

[69] Kenneth L McMillan. The SMV system. In *Symbolic Model Checking*, pages 61–85. Springer, 1993. One citation in section 1.

[70] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. One citation in section 3.1.1.

[71] Robert Paige and Robert E Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987. 4 citations in sections 1, 3.1.1, 3.1.2, and 4.2.1.

[72] Robert Paige, Robert E Tarjan, and Robert Bonic. A linear time solution to the single function coarsest partition problem. *Theoretical computer science*, 40:67–84, 1985. One citation in section 3.1.1.

[73] David Park. *Concurrency and automata on infinite sequences.* Springer, 1981. One citation in section 3.1.1.

[74] James Frederick Sexton. *The Development of Taylor's and MacLaurin's Series.* PhD thesis, Seattle University, 1939. One citation in section 2.7.1.

[75] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, 1985. One citation in section 3.1.2.

[76] Oleg V Sokolsky and Scott A Smolka. Incremental model checking in the modal mu-calculus. In *International Conference on Computer Aided Verification*, pages 351–363. Springer, 1994. One citation in section 3.3.

[77] Jeremy Sproston and Susanna Donatelli. Backward bisimulation in Markov chain model checking. *Software Engineering, IEEE Transactions on*, 32(8):531–546, 2006. One citation in section 3.1.2.

[78] M. Ujma. *On Verification and Controller Synthesis for Probabilistic Systems at Runtime.* PhD thesis, University of Oxford, 2015. One citation in section 3.4.

[79] Antti Valmari. The state explosion problem. In *Advanced Course on Petri Nets*, pages 429–528. Springer, 1996. One citation in section 3.

[80] Antti Valmari and Giuliana Franceschinis. Simple o(m logn) time Markov chain lumping. In *Proc. TACAS'10*, pages 38–52. Springer, 2010. 3 citations in sections 1, 3.1.2, and 6.1.

[81] J. von Neumann. Probabilistic logics and synthesis of reliable organisms from unreliable components. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 43–98. Princeton University Press, 1956. One citation in section 5.6.

[82] Michael Vose. *The simple genetic algorithm: Foundations and theory.* MIT Press, 1999. One citation in section 4.4.

[83] Ralf Wimmer and Bernd Becker. Correctness issues of symbolic bisimulation computation for markov chains. In *Proc. MMB'10*, 2010. One citation in section 3.4.

[84] http://www.prismmodelchecker.org/doc/semantics.pdf. One citation in section 9.

[85] http://www.prismmodelchecker.org/files/spin16fh. 2 citations in sections 4.4 and 4.4.1.