Decision Algorithms for Probabilistic Simulations

Dissertation zur Erlangung des Grades des Doktors der Naturwissenschaften der Naturwissenschaftlich-Technischen Fakultäten der Universität des Saarlandes

Lijun Zhang

Saarbrücken, 2009

Tag des Kolloquiums	05.12.2008
Dekan	Prof. Dr. Joachim Weickert

Prüfungsausschuss

Vorsitzender	Prof. Dr. Bernd Finkbeiner
Berichterstatter	Prof. Dr. Holger Hermanns Prof. Dr. Rance Cleaveland Prof. Dr. Friedrich Eisenbrand Prof. Dr. Roberto Segala
Akademischer Mitarbeiter	Dr. Philipp Lucas

Abstract

Probabilistic phenomena arise in embedded, distributed, networked, biological and security systems, and are accounted for by various probabilistic modeling formalisms based on labelled transition systems. Among the most popular ones are homogeneous discrete-time and continuous-time Markov chains (DTMCs and CTMCs) and their extensions with nondeterminism, which we will consider in this thesis. Simulation relations admit comparing the behavior of two models and provide the principal ingredients to perform abstractions of the models while preserving interesting properties. Intuitively, one model simulates another model if it can imitate all of its moves. Simulation preorders are compositional, thus allowing hierarchical verification and decomposition of difficult verification tasks into several subproblems. Recently, variants of simulation relations, such as simulatability and polynomially accurate probabilistic simulations, have been introduced to prove soundness of security protocols. The focus of this thesis lies in decision algorithms for various simulation preorders of probabilistic systems. We propose efficient decision algorithms and provide also experimental comparisons of these algorithms.

Zusammenfassung

In einem breiten Spektrum von Systemen, etwa bei eingebetteten, verteilten, netzwerkbasierten und biologischen System sowie im Bereich Security, treten Phänomene auf, die sich sehr gut durch Probabilismus beschreiben lassen. Als Modellierungsformalismus dienen dabei verschiedene probabilistische Erweiterungen von Transitionssystemen. Zu den wohl populärsten Formalismen dieser Art zählen hier homogene Markovketten (Markov chains) mit diskreter Zeit und Markovketten mit kontinuierlicher Zeit, bzw. deren Erweiterungen mit Nichtdeterminismus. Genau diese Klasse von Modellen betrachten wir in dieser Dissertation. Simulationsrelationen erlauben es, das Verhalten zweier Modelle in Beziehung zu setzen und liefern den grundlegenden Baustein, um Abstraktionen so zu betreiben, daß interessante Eigenschaften erhalten bleiben. Intuitiv gesprochen simuliert ein Modell ein anderes, wenn es alle Zustandsübergänge des anderen imitieren kann. Derartige Simulationsordnungen sind kompositional, daher erlauben sie hierarchische Verifikation und Zerlegung von Verifikationsaufgaben in kleinere Unterprobleme. Kürzlich wurden Simulationsrelationen eingeführt, wie etwa Simulatability und Polynomiell Akkurate Probabilstische Simulationen, um Korrektheit von Sicherheitsprotokollen zu zeigen. Der Schwerpunkt dieser Dissertation liegt auf Entscheidungsalgorithmen für verschiedene Simulationsordnungen auf probabilistischen Systemen. Wir stellen neue, effiziente Entscheidungsalgorithmen vor und vergleichen diese in Experimenten mit existierenden Algorithmen.

Erweiterte Zusammenfassung

In einem breiten Spektrum von Systemen, etwa bei eingebetteten, verteilten, netzwerkbasierten und biologischen System sowie im Bereich Security, treten Phänomene auf, die Markovsystemen als unterliegende semantische Modelle haben. Simulationsrelationen erlauben es, das Verhalten zweier solcher Modelle in Beziehung zu setzen und liefern den grundlegenden Baustein, um Abstraktionen so zu betreiben, daß interessante Eigenschaften erhalten bleiben. Derartige Simulationsordnungen sind kompositional, daher erlauben sie hierarchische Verifikation und Zerlegung von Verifikationsaufgaben in kleinere Unterprobleme. Simulationsrelationen können auch direkt angewendet in der Modellprüfung. Kürzlich wurden Simulationsrelationen eingeführt, wie etwa Simulatability und Polynomiell Akkurate Probabilstische Simulationen, um Korrektheit von Sicherheitsprotokollen zu zeigen.

In dieser Dissertation stellen wir Entscheidungsalgorithmen für verschiedene Simulationsordnungen auf Markovsystemen vor. Die Modellklasse die wir betrachten sind homogene Markovketten (Markov chains) mit diskreter Zeit und Markovketten mit kontinuierlicher Zeit, bzw. deren Erweiterungen mit Nichtdeterminismus: Markov Entscheidungsprozesse mit diskreter Zeit und kontinuierlicher Zeit.

Intuitiv gesprochen simuliert ein Zustand s' einen anderen Zustand s ($s \precsim s'$), wenn s' alle Zustandsübergänge von s imitieren kann. Für Transitionssysteme ohne Wahrscheinlichkeit bedeutet dies, dass wenn s einen Nachfolgezustand t hat, dann muss s' auch einen Nachfolgezustand t' haben, so dass t' den Zustand t simuliert. Für Markovmodelle ist dies komplizierter, da nach einem Zustand immer eine Verteilung folgt. Diese entsprechende Bedingungen läßt sich durch eine gewichtete Funktion ausdrücken.

Der Standard-Entscheidungsalgorithmus für Simulationsrelationen funktioniert wie folgt: Wir fangen mit einer Relation R, die gröber als die Simulationsrelation $\precsim$ ist. Dann versuchen wir die Paare von R zu entfernen, die die Bedingungen von Simulationrelationen bezüglich der Relation R verletzen. Dieser Prozess läuft bis keine Paare mehr entfernt werden können. Als Fixpunkt ergibt sich so die Simulationsrelation $\precsim$. Im schlimmsten Fall wird pro Iteration nur ein Paar entfernt. Die Anzahl von Iterationen

ist beschänkt durch n^2, wobei n die Anzahl der Zustände ist.

In einem Transitionssystem ohne Wahrscheinlichkeit verletzt ein Paar (s, s') die Bedingung bezüglich R, wenn s einen Nachfolgezustand t hat, aber s' keinen Nachfolgezustand t' hat, so dass (t, t') auch in R liegt. Für Markovmodelle muss diese Bedingung, die durch eine gewichtete Funktion beschrieben wird, überprüft werden. Diese Bedingung kann mit Hilfe von Algorithmen für maximalen Fluss überprüft werden, was aber eine Zeitkomplizität von $\mathcal{O}(n^3)$ ergibt. Dies führt zu einem Algorithmus mit Zeitkomplexität $\mathcal{O}(n^7)$. Inspiriert von Algorithmen für parametrischen maximalen Fluss überprüfen wir die Bedingungen bezüglich gewichteter Funktionen inkrementell, d.h. nachdem wir einmal das Algorithmus für maximalen Fluss aufgerufen haben, speichern wir das Netzwerk, den Fluss und andere wichtige Informationen. In der nächsten Iteration, nutzen wir die gespeichert Informationen aus, um den Fluss effizient zu bekommen, anstatt den Algorithmus neu aufzurufen. Auf diese Art und Weise erhalten wir einen Algorithmus mit Zeitkomplexität $\mathcal{O}(m^2 n)$, wobei m die Anzahl von Transitionen ist. Als Speicherkomplexität haben wir aber $\mathcal{O}(m^2)$ anstatt $\mathcal{O}(n^2)$ wegen des zusätzlichen Speicherbedarfs.

Desweiteren betrachten wir Entscheidungsalgorithmen für schwache Simulationen für Markovketten und probabilistische Simulationen für Markoventscheidungsprozesse. Sowohl schwache Simulationen als auch probabilistische Simulationen sind gröber als Simulationen. Sie erlauben daher noch stärkere Abstraktion. Für schwache Simulation präsentieren wir einen effizienten Algorithmus basierend auf dem Breakpoint-Algorithmus. Bis jetzt gibt noch keinen Entscheidungsalgorithmus für probabilistische Simulationen. Wir reduzieren das Problem auf Lineare Programmierung, welche polynomielle Laufzeit hat.

Um den Speicherbedarf zu reduzieren, betrachten wir Algorithmen, welche auf Partitionsverfeinerung basieren, so erhalten wir platzeffizientere Entscheidungsalgorithmen. In Experimenten zeig sich, dass in manchen Fällen so auch die Laufzeit verbessert werden kann.

Acknowledgments

I would first thank my supervisor Holger Hermanns, who also guided my master thesis. He is always ready for discussions, and provided me valuable ideas and various directions. He taught me how to write papers, read my drafts, gave a lot suggestions for improvement, and corrected my English mistakes. Without his endless support this thesis would not have been completed. His deep passion for research, and his broad view of the research area will always influent me in the future.

I am grateful to David N. Jansen for his advising of my master thesis. The research collaborations with him, which appear in this thesis, have been very fruitful. His ability to find problems and to suggest elegant solutions helped me greatly, especially if I got stuck. His very careful reading of our papers has improved this thesis a lot. I am also grateful to Friedrich Eisenbrand for several insightful discussions about maximum flow problems, especially about the parametric maximum flow problems, which play a very important role in this thesis.

Björn Wachter deserves a special thank for his multiple roles: he was my excellent study partner during our computer science study in Saarbrücken; he is a very good friend of mine who has been helping me a lot during my stay in Germany; and he is a very great coauthor of several papers with me. We have been having so many wonderful discussions, which lead to wonderful ideas and nice papers. I am also thankful to Jonathan Bogdoll, Pepijn Crouzen and E. Moritz Hahn for our great scientific collaborations. It was a pleasure to work with you. A special thank to Jonathan Bogdoll for helping me with the implementation of the algorithms in this thesis.

I thank all my group colleges Sven Johr, Christian Eisentraut, Reza Pulungan, and again Pepijn Crouzen and E. Moritz Hahn. I enjoyed very much in our amazing discussions. Thanks also to our group secretary Christa Schäfer, who helped me to fill out thousand of formulas.

I am grateful for the support received from the NWO-DFG bilateral project VOSS, the DFG as part of the Transregional Collaborative Research Center SFB/TR 14 AVACS, and the European Community's Seventh Framework Programme under grant agreement

n^o 214755.

I thank my parent Guiping Liu, Hualin Zhang for their endless love. Without their support from the very beginning I would not be able to finish the school and study in Germany. I also thank my parents-in-law, Huiling Zhang and Junda Wen, for their great support in taking care of my son.

Last but not least, a special thank goes to my wife Chao Wen and my son Tianqi: This thesis is dedicated to them.

Contents

Acknowledgments **v**

1 Introduction **1**

1.1 Model Checking 1

1.2 Simulation Relations 3

1.3 Decision Algorithms 5

1.3.1 Contributions 5

1.4 Organisation of the Thesis 8

2 Markov Models **9**

2.1 Fully Probabilistic Systems 10

2.2 Discrete-time Markov Chains 12

2.3 Continuous-time Markov Chains 12

2.4 Probabilistic Automata 13

2.5 Continuous-time Probabilistic Automata 14

2.6 Bibliographic Notes 15

3 Computing Maximum Flows **17**

3.1 Maximum Flow Problems 17

3.2 Augmenting Path Algorithm 18

3.3 The Preflow Algorithm 19

3.4 Feasible Flow Problem 21

3.5 Minimum Cut 22

4 Simulation Relations **23**

4.1 Standard Definitions 24

4.1.1 Weight Functions 25

4.1.2 Strong Simulation . . . 27
4.1.3 Strong Probabilistic Simulations . . . 32
4.2 Alternative Simulation Definitions . . . 35
4.2.1 Characterising Weight Functions . . . 36
4.2.2 Strong (Probabilistic) Simulations . . . 39
4.3 Weak Simulations . . . 42
4.3.1 Weak Simulation for DTMCs . . . 42
4.3.2 Weak Simulation for CTMCs . . . 46
4.3.3 Weak Simulation for FPSs . . . 47
4.4 Bibliographic Notes . . . 51
4.5 Summary . . . 52

5 Algorithms for Strong Simulations 53
5.1 Strong Simulation up to R . . . 54
5.2 Strong Simulation . . . 55
5.2.1 Basic Algorithm to Decide Strong Simulation . . . 55
5.2.2 An Improved Algorithm for FPSs . . . 56
5.2.3 Algorithms for PAs and CPAs . . . 63
5.3 Strong Probabilistic Simulations . . . 68
5.3.1 An Algorithm for PAs . . . 68
5.3.2 An Algorithm for CPAs . . . 70
5.4 Experimental Results . . . 72
5.4.1 Optimisation Options . . . 72
5.4.2 Case studies . . . 74
5.5 Bibliographic Notes . . . 83
5.6 Summary . . . 84

6 Algorithms for Weak Simulations 85
6.1 Weak Simulation up to R . . . 86
6.2 An Algorithm for DTMCs . . . 88
6.2.1 The Parametric Network $\mathcal{N}(\gamma)$ . . . 88
6.2.2 Breakpoints . . . 91
6.2.3 The Algorithm . . . 94
6.2.4 An Improvement . . . 99

6.3 An Algorithm for CTMCs . . . 102
6.4 Experimental Results . . . 103
6.5 Bibliographic Notes . . . 104
6.6 Summary . . . 104

7 Simulation Based Minimisation **105**
7.1 The Quotient Automata . . . 106
7.1.1 The Minimal Quotient Automaton . . . 109
7.1.2 Safety and Liveness Properties . . . 112
7.2 Simulation Characterised by Partition Pairs . . . 113
7.3 Solving the GCPP . . . 118
7.3.1 Refinement of the Partition . . . 119
7.3.2 Refinement of the Partition Relations . . . 122
7.3.3 Correctness . . . 124
7.3.4 On Acyclicity . . . 125
7.3.5 Complexity . . . 126
7.4 Simulation Quotient for CPAs . . . 128
7.4.1 Safety Properties . . . 128
7.4.2 The Minimal Quotient Automaton for CPAs . . . 130
7.4.3 Algorithm for CPAs . . . 131
7.5 Experimental Results . . . 131
7.6 Bibliographic Notes . . . 135
7.7 Summary . . . 135

8 Conclusion and Future Works **137**
8.1 Conclusion . . . 137
8.2 Future works . . . 138

Bibliography **139**

Chapter 1

Introduction

With the ubiquity of computing systems in our world, the need arises to assess the correctness of those systems. The past has shown that undiscovered errors in safety-critical systems may result in enormous economic costs (The Explosion of the Ariane 5) or even physical harm to humans (Therac-25 Accidents).

1.1 Model Checking

Model Checking. Many verification methods have been introduced to prove the correctness of systems exploiting rigorous mathematical foundations. As one of the automatic verification techniques, model checking [16] has successfully been applied to automatically find errors in complex systems. Recently, Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis received the 2007 A.M. Turing Award for their original and leading research. Model checking answers the question whether the system satisfies a property. The system is usually given as a labelled transition system (Kripke structure), and the property can be specified in terms of a temporal logical formula.

Models. Most of the systems such as embedded, distributed, networked, randomized, and biological systems, exhibit probabilistic phenomena. Based on labelled transition systems, various probabilistic models have been proposed in the literature. We consider homogeneous discrete-time and continuous-time Markov chains (DTMCs and CTMCs) [77, 106] and their extensions with nondeterminism. Associated with each DTMC or CTMC is a set of system states. We assume that the system we are modelling occupies exactly one state at any moment. To model the evolution of the system, a transition matrix is used to represent transitions from one state to another. In DTMCs, the transition matrix assigns each state a probability distribution, which provides the probability of going to state s' from state s at discrete time point t. Since we are considering homogeneous Markov chains, the probability of going to s' from s is independent of the time point t. In CTMCs, the corresponding transition probability is exponentially distributed over time with rates given by the rate matrix. If we denote the sum of the outgoing rates of a state s as its *exit rate*, the probability that starting from s within time t any transition is taken is exponentially distributed with the exit rate of s. A

CTMC induces a time-abstract DTMC, also called the *embedded* DTMC: the transition probability from state s to s' is obtained by taking the fraction of the rate of this transition over the exit rate of s. Then, in a CTMC, starting from s, a particular transition to s' is triggered within time t is given by the product of the probability of this transition in its embedded DTMC, and the probability that any transition will be triggered within time t. The fundamental property of Markov chains is the *Markov property* (or called the memoryless property), which states that the future depends only on the current state, not on the history.

Probabilistic automata (PAs) [98] extend transition systems with probabilistic selection, or, viewed differently, extend DTMCs with nondeterminism. They constitute a natural model of concurrent computation involving random phenomena. We consider probabilistic automata in the style of Segala and Lynch [101]. In a nutshell, a labelled transition in some PA leads to a probability distribution over the set of states, rather than a single state. Informally, starting from state s, a distribution is associated with an action α, which is also called the α-successor distribution. The resulting model exhibits nondeterministic choice (as in labelled transition systems) and afterwards probabilistic choice (as in DTMCs). Once such nondeterminism is resolved by choosing some α-successor distribution deterministically (called *deterministic schedulers* in the literature), the PA behaves then just like a DTMC. In PAs there may exist nondeterminism between equally-labelled transitions (also called *internal nondeterminism*), i.e., more than one α-distributions from the same state. Markov decision processes (MDPs) [46] are an important subclass of PAs, which do not allow internal nondeterminism.

Continuous-time probabilistic automata (CPAs) [79, 93, 20, 120] are obtained by extending CTMCs with nondeterminism, just as PAs extend DTMCs. In CPAs, for state s, a rate function, which assigns each state a rate, is associated with an action alphabet. There may exist more than one rate function for state s associated with the same action. Similar to PAs, once the nondeterminism is resolved by choosing some transition deterministically, the CPA behaves then as a CTMC. Continuous-time Markov decision processes (CTMDPs) are special CPAs where no internal nondeterminism exists. CPAs are a natural semantic model for various performance and dependability modelling formalisms including stochastic activity networks [95], generalised stochastic Petri nets [84] and interactive Markov chains [65].

Properties. Temporal logic (or tense logic) has been introduced by Arthur Prior about fifty years ago to describe system of rules. In model checking, it is widely used to reason about transition systems in terms of temporal operators. Various temporal logics have been introduced, including linear temporal logic (LTL) [83] and (CTL) [35]. These logics have also been extended to capture probabilistic properties: A discrete probabilistic variant of CTL, called PCTL [62] (probabilistic CTL), is interpreted over discrete-time models. For the continuous-time case, we consider continuous stochastic logic (CSL) [5], which is the continuous stochastic extension of PCTL tailored to CTMCs or CPAs. As an example, a safety property such as *"the probability within k steps (t time units) that the system reach a set of unsafe sates is bounded by ϵ"* can be expressed as a PCTL (CSL) formula.

1.2 Simulation Relations

Simulations for LTSs. The power of model checking is limited by the infamous state space explosion problem. Notably, minimizing the system to the bisimulation [86, 89] quotient is a favorable approach. As a more aggressive attack to the problem, simulation relations [85] have been proposed for these models. In particular, they provide the principal ingredients to perform abstractions of the models, while preserving safe CTL properties (formulas with positive universal path-quantifiers only) [36].

While bisimulation relations are equivalence relations on the state space, simulation relations ($\precsim$) are preorders such that if $s \precsim s'$ ("s' simulates s") state s' can mimic all stepwise behaviour of s; the converse, i. e., $s' \precsim s$ is not guaranteed, so state s' may perform steps that cannot be matched by s. Thus, if $s \precsim s'$ then every successor of s has a corresponding related successor of s', but the reverse implication does not necessarily hold. Simulation relations are often used for verification purposes to show that one system correctly implements another, more abstract system. One of the interesting aspects of simulation relations is that they allow a verification by "local" reasoning. In the context of model checking, simulation relations can be used to combat the well-known state explosion problem, owed to the preservation of certain classes of temporal formulas. For instance, if $s \precsim s'$ then for all safe CTL formulas Φ it follows that $s' \models \Phi$ implies $s \models \Phi$ [36]. Simulation can be lifted to the level of transition systems by considering their corresponding initial states: system Q simulates system P, denoted by $P \precsim Q$ if the initial state of P can be simulated by the initial state of Q. Simulation can then be used for verification purposes [1, 71, 64]: Assume that the implementation is represented by the system P and the specification is represented by the system Q. Q simulates P would mean that P correctly implements Q.

Simulations for Probabilistic Systems. Bisimulation and simulation relations have been extended to DTMCs [72, 80]. In correspondence to the non-probabilistic setting, a simulation preorder provides the principal ingredients to perform abstraction of DTMCs, while preserving safe fragments of the safe PCTL formulas. For DTMCs, $s \precsim s'$ requires that the distribution of s' can match the distribution of s. Correspondence of distributions is naturally defined with the concept of weight functions [72]. Bisimulation and simulation relations [101] have been further extended to compare the stepwise behaviour of states in PAs: For $s \precsim s'$, it is required that every successor distribution of s via action α (called α-successor distribution) has a corresponding α-successor distribution at s'. As for DTMCs, in the context of model checking, simulation relations preserve safe PCTL formulas [101]. Probabilistic simulation [101] is a relaxation of simulation in the sense that it enables convex combinations of multiple distributions belonging to equally labelled transitions. More concretely, it may happen that a state s has no α-successor distribution which can be related to an α-successor distribution of s', yet there exists a so-called *α-combined transition*, a convex combination of several α-successor distributions. Probabilistic simulation accounts for this and is thus coarser than simulation, but still preserves the same class of PCTL-properties as simulation does.

Weak simulation for DTMCs is proposed in [18]. In weak simulation, the successor states are split into visible and invisible parts, and the weight function conditions are

only imposed on the transitions leading to the visible parts of the successor states. Weak simulation is strictly coarser than the afore-mentioned simulation (will be called strong simulation) for Markov chains, thus allows further reduction of the state space. It preserves the safe PCTL-properties without the next state formulas [18].

In [18], simulation and weak simulation relations are also introduced for CTMCs. If $s \precsim s'$, the following two conditions must hold. The first condition requires that $s \precsim s'$ holds in the embedded DTMC of the CTMC. The other part, called the rate condition, requires that state s' is "faster" than s which manifests itself by a higher exit rate. While simulation preserves the safety fragment of CSL formulas, weak simulation preserves the safe CSL-properties without the next state formulas. Simulation for CPAs [122] combines the simulation for the induced PAs and an additional rate condition. In [122], the notion of probabilistic simulation relations is also introduced for CPAs, which preserve the safe CSL formulas.

Verifying Systems with Simulation Relations. In many applications, a system can be specified concisely by means of a probabilistic model rather than in terms of the logics PCTL or CSL. Examples of this kind include various recent wireless network protocols, such as ZigBee [59], Firewire Zeroconf [25], or the novel IEEE 802.11e, where the central mechanism is selecting among different-sided dice, readily expressible as a probabilistic automaton [82]. For such cases, a decision algorithm for simulation preorder serves as specification checker: The model satisfies the specification if the automaton for the specification *simulates* the automaton for the model. Given the emergence of ever more wireless standards of that sort, there is an obvious motivation to study the principal technological basis: decision algorithms for simulation preorders. Just as standard simulation for LTSs, the main advantage of simulation methods for probabilistic models is that it is compositional [101]. This allows hierarchical verification which decomposes difficult verification tasks into several intermediate refinements.

Probabilistic automata have been considered as formal model for security protocols [90, 91] which involve probabilistic choices. The notion of *simulatability* is introduced which allows to prove soundness of implementation relations, i.e., an ideal system in the Dolev-Yao [50] style can be implemented by a real system. Proof techniques based on simulatability allow *"error sets"*, i.e., a subset of states of the real system which cannot be proved to be simulated by the ideal system; however the probability of such is negligible. Based on probabilistic simulation, Segala and Turrini [104] proposed the notion of *polynomially accurate probabilistic simulations* which incorporate the error sets into the definition.

Simulation and probabilistic simulation relations can also be directly used in model checking: The kernel of the relation, i. e., the corresponding simulation equivalence $\precsim \cap \succsim$, can be constructed to build a quotient automaton. Then, model checking can be performed on the quotient automaton which might be significantly smaller than the original one. The quotient construction preserves both safety and liveness fragments of PCTL and CSL for discrete-time and continuous-time models respectively. This means that, as long as one is interested in safety or liveness properties, it is favorable to perform model checking on the simulation equivalence quotient. To obtain the quotient, an algorithm for deciding simulation preorder is needed.

1.3 Decision Algorithms

The core of the thesis are decision algorithms for simulation preorders for probabilistic systems. We briefly review the existing algorithms. We give complexity results in terms of the number of states n and the number of transitions m of probabilistic systems under analysis.

In the non-probabilistic setting, efficient algorithms for deciding simulation preorders have been proposed in [22, 64]. The complexity is $\mathcal{O}(mn)$ where n and m denotes the number of states and transitions of the transition system respectively. For probabilistic automata, Baier *et al.* [9] introduced a polynomial decision algorithm with time complexity $\mathcal{O}((mn^6 + m^2n^3)/\log n)$ and space complexity $\mathcal{O}(m^2)$, by exploiting a network flow algorithm. For DTMCs or MDPs, their algorithm has complexity $\mathcal{O}(n^7/\log n)$ in time and $\mathcal{O}(n^2)$ in space. For Markov chains, it is proved that probabilistic weak simulation is decidable in polynomial time [13] by reductions to LP problems.

1.3.1 Contributions

This thesis investigates effective algorithms for probabilistic simulations. The common scheme of decision algorithms for simulations is as follows. The algorithm starts with the relation R which is guaranteed to be coarser than the simulation preorder $\precsim$. Then, the relation R will be refined. In each iteration of the refinement loop, pairs (s, s') are eliminated from the relation R if the corresponding simulation conditions are violated with respect to the current relation. In the context of labelled transitions systems, this means that s has a successor state t, but we cannot find a successor state t' of s' such that (t, t') is also in the current relation R. For DTMCs, this correspondence is formulated by the existence of a weight function for distributions $(\mathbf{P}(s, \cdot), \mathbf{P}(s', \cdot))$ with respect to the current relation R. Checking this weight function condition reduces to checking whether there is a maximum flow over the network constructed out of $(\mathbf{P}(s, \cdot), \mathbf{P}(s', \cdot))$ and the current relation R. The complexity for one such check is however rather expensive, it has time complexity $\mathcal{O}(n^3/\log n)$. If the iterative algorithm reaches a fix-point, the strong simulation preorder is obtained. The number of iterations of the refinement loop is in worst case $\mathcal{O}(n^2)$, and the overall complexity [9] amounts to $\mathcal{O}(n^7/\log n)$ in time and $\mathcal{O}(n^2)$ in space.

Fixing a pair (s, s'), we observe that the networks for this pair across iterations of the refinement loop are very similar: They differ from iteration to iteration only by deletion of some edges induced by the successive clean up of R. We exploit this by adapting a parametric maximum flow algorithm [53] to solve the maximum flow problems for the arising sequences of similar networks, hence arriving at efficient simulation decision algorithms. The basic idea is that all computations concerning the pair (s, s') can be performed in an incremental way: after each iteration we save the current network together with maximum flow information. Then, in the next iteration, we update the network, and derive the maximum flow while using the previous maximum flow function. The maximum flow problems for the arising sequences of similar networks with respect to the pair (s, s') can be computed in time $\mathcal{O}(|V|^3)$ where $|V|$ is the number of nodes of the network. This leads to an overall time complexity $\mathcal{O}(m^2n)$ for deciding the simulation

preorder. Because of the storage of the networks, the space complexity is increased to $\mathcal{O}(m^2)$. Especially in the very common case where the state fanout of a model is bounded by a constant g (and hence $m \leq gn$), our strong simulation algorithm has time and space complexity $\mathcal{O}(n^2)$.

The above algorithm can be extended easily to handle CTMCs. Recall in a CTMC $s \precsim s'$ holds if it holds in the embedded DTMC of the CTMC, and s' is faster than s, which is called the rate condition. We can ensure that additional rate condition by incorporating it into the initial relation R. In the refinement steps afterwards, only the weight function conditions needs to be checked in the embedded DTMC. Thus, we achieve an algorithm for CTMCs with same time and space complexity.

For weak simulation on Markov chains, the parametric maximum flow technique cannot be applied directly. Nevertheless, we manage to incorporate the parametric maximum flow idea into a decision algorithm with time complexity $\mathcal{O}(m^2n^3)$ and space complexity $\mathcal{O}(n^2)$. An earlier algorithm [13] uses LP problems [74, 96] as subroutines. The maximum flow problem is a special instance of an LP problem but can be solved much more efficiently [2].

The above joint work is with Holger Hermanns, Friedrich Eisenbrand and David N. Jansen, and has been published in [123, 124].

We extend the algorithm to compute the strong simulation preorder to also work on PAs. It takes the skeleton of the algorithm for Markov chains: It starts with a relation R which is coarser than $\precsim$, and then refines R until $\precsim$ is achieved. In the refinement loop, a pair (s, s') is thrown out of the pair if the corresponding simulation conditions are violated with respect to the current relation. For PAs, this means that there exists an α-successor distribution μ of s, such that for all α-successor distributions μ' of s', we cannot find a weight function for (μ, μ') with respect to the current relation R. Again, as for Markov chains, the existence of such weight functions can be reduced to maximum flow problems. Combining with the parametric maximum flow algorithm [53], we arrive at the same time complexity $\mathcal{O}(m^2n)$ and space complexity $\mathcal{O}(m^2)$ as for Markov chains.

The above maximum flow based procedure cannot be applied to deal with strong *probabilistic* simulation for PAs. The reason is that α-combined transition of state s is a convex combination of several α-successor distributions of s, thus results in uncountable many such possible combined transitions. The computational complexity of deciding strong probabilistic simulation has not been investigated before. We show that it can be reduced to solving LP problems. The idea is that we introduce for each α-successor distribution a variable, and then reformulate the requirements concerning the combined transitions by linear constraints over these variables. This allows us to construct a set of LP problem such that whether a pair (s, s') should be thrown out of the current pair R is equivalent to whether each of the LP problem has a solution.

The algorithms for PAs are then extended to handle their continuous-time analogue, CPAs. In the algorithm, for each pair (s, s') in the refinement loop, the additional rate condition is ensured by an additional check via comparing the appropriate rates of s and s'. The resulting algorithm has the same time and space complexity.

The above joint work is with Holger Hermanns, Friedrich Eisenbrand and David N. Jansen, and has been published in [122, 124].

In addition to developing efficient algorithms, and establishing their worst-case complexities, we present an experimental evaluation of these algorithms, together with various optimisations. The evaluation is carried out on both standard example cases as well as randomly generated models. Experimental results show that for sparse models, the parametric maximum flow based algorithm is only slightly more efficient, but uses more memory instead. The strength of the new algorithms are dense models, which seems seldom in models commonly used for case studies. The gap between theoretical and practical efficiency is not caused by "the constant factors" but by the fact that the corner cases that blow up the worst case complexity are rare in practice.

We also consider a few simple optimisations for the algorithm by exploiting the structure of networks. To our surprise, such simple optimisations produce promising results in general in our practical studies, in comparison to the theoretically better algorithm.

The above joint work is with Jonathan Bogdoll and Holger Hermanns, and has been published in [24].

Sometimes, the space complexity becomes the bottleneck of the decision algorithm. Inspired by the work of [54], we study space efficient algorithms for deciding the simulation preorder for PAs and CPAs. We incorporate partition refinement techniques into the computation. In the algorithm, along a partition of the state space, we maintain also a relation between the blocks in the partition. Then, we check the weight function conditions in the quotient automaton induced by this partition, instead on the original automaton. This reduces the space complexity to $\mathcal{O}(n_\diamond^2 + n \log n_\diamond)$ where $n_\diamond$ denotes the number of simulation equivalence classes. The time complexity, however, is rather heavy: $\mathcal{O}(mn_\diamond + m_\diamond^2 n_\diamond^4 + m_\sim^2 n_\diamond^2)$ where $m_\diamond$ and $m_\sim$ denote the number of transitions in the simulation equivalent quotient and the strong bisimulation quotient respectively. We exploit the parametric maximum flow technique to get a better time bound $\mathcal{O}(mn_\diamond + m_\sim^2 n_\diamond^2)$, but with space penalty $\mathcal{O}(m_\diamond^2 + n \log n_\diamond)$. Experimental results show that the partition refinement based method is very effective in time and memory: not only the space-efficiency is improved drastically, often orders of magnitude less time are required.

This work has been published in [121].

As a side result, we propose a new equivalent definition of strong (probabilistic) simulations. In our new definition, instead of using weight functions to relate distributions, we compare the distributions by considering an arbitrary subset of states. Informally, for states s, s' in a DTMC, $s \precsim s'$ implies the probability of going to a set of states A from s is smaller than or equal to the probability of going to a related set (with respect to the strong simulation relation) of states from s'. This alternative characterisation makes the simulation relations for Markov chains more understandable for those who are familiar with labelled transition systems. An alternative definition of strong simulation is also provided for CTMCs, PAs and CPAs. Using the same idea, we also provide an equivalent definition of strong probabilistic simulation for PAs and CPAs.

1.4 Organisation of the Thesis

Chapter 2 In this chapter we recall the definition of discrete-time and continuous-time Markov chains, as well as discrete-time and continuous-time probabilistic automata.

Chapter 3 We recall the maximum flow problem, and algorithms for solving the problem. We also recall some basic results needed in subsequent chapters.

Chapter 4 This chapter first repeats the standard definition of strong and strong probabilistic simulations. As we mentioned in the introduction, the definition uses the notion of weight functions to relate the distributions. In the second part of this chapter, we propose a new equivalent definition of strong simulation which provides a rather intuitive view. We also recall the definition of weak simulation for Markov chains. We show that for fully probabilistic systems (FPSs) there are some intricacies in the original definition of weak simulation.

Chapter 5 In this chapter we introduce our parametric maximum flow based decision algorithm for strong and strong probabilistic simulations. For both Markov chains and probabilistic automata, we arrive at an algorithm with overall time complexity $\mathcal{O}(m^2 n)$ and space complexity $\mathcal{O}(m^2)$. In this chapter we also report on experimental comparisons, with various useful optimisations.

Chapter 6 In this chapter, we propose a decision algorithm for weak simulation for Markov chains. We incorporate the parametric maximum flow algorithm into a decision algorithm with time complexity $\mathcal{O}(m^2 n^3)$ and space complexity $\mathcal{O}(n^2)$.

Chapter 7 In this chapter, we incorporate the partition refinement technique into the computation of the simulation preorder for PAs and CPAs. The resulting algorithm has space complexity $\mathcal{O}(n_\diamond^2 + n \log n_\diamond)$ where $n_\diamond$ denotes the number of simulation equivalence classes. We provide also experimental results showing that this method is very effective in time and memory.

Chapter 8 This chapter concludes the thesis.

Chapter 2

Markov Models

This chapter introduces the models considered in this thesis. We consider homogeneous discrete-time and continuous-time Markov chains (DTMCs and CTMCs) and their extensions with nondeterminism. Associated with each DTMC or CTMC is a set of system states. We assume that the system we are modelling occupies exactly one state at any moment. To model the evolution of the system, a transition matrix is used to represent the transitions from one state to another. In DTMCs, the transition matrix is probabilistic, which provides the probability of going to state s' from s at discrete time point t. For homogeneous Markov chains, the probability of going to s' from s is independent of the time point t. In CTMCs, the corresponding transition probability is exponentially distributed with rates given by the rate matrix. The fundamental property of Markov chains is the *Markov property* (also called the memoryless property), which states that the future depends only on the current state, not on the history.

For each state s in a DTMC, the sum of probabilities from s is either 0 (s is called absorbing) or 1 (s is called stochastic). In this chapter, we consider another type of states in which the sum of probabilities from s is between 0 and 1 (s is then called substochastic). In this case, the transitions out of s are under-specified. As indicated in [47], under-specification can be used to model behaviour which is unknown or of no interest. The resulting model is called fully probabilistic systems (FPSs). DTMCs are special FPSs in which states are either stochastic or absorbing.

Probabilistic automata (PAs) extend labelled transition systems with probabilistic selection, or extend FPSs with nondeterminism. As for FPSs, we also allow substochastic behaviour to model under-specification. A labelled transition in some PA leads to a probability distribution over the set of states, rather than a single state. The resulting model thus exhibits both nondeterministic choice and probabilistic choice. We also consider the Continuous-time probabilistic automata (CPAs), which extend CTMCs with nondeterminism.

Organisation of this Chapter. We recall the definition of fully probabilistic systems in Section 2.1. Discrete- and continuous-time Markov chains will be presented in Sections 2.2 and 2.3 respectively. In Sections 2.4 and 2.5, we repeat the definitions of nondeterministic extensions of the discrete-time and continuous-time models respectively. In Section 2.6, we discuss other probabilistic models from the literature.

2.1 Fully Probabilistic Systems

Notations. Let X, Y be finite sets. For a two-argument function $f : X \times Y \to \mathbb{R}$, let $f(A, y) = \sum_{x \in A} f(x, y)$ for all $A \subseteq X$ and $y \in Y$. Analogously, let $f(x, B) = \sum_{y \in B} f(x, y)$ for all $x \in X$ and $B \subseteq Y$. The extension to functions with one or three arguments is obvious. Let AP be a fixed, finite set of atomic propositions.

Distributions. For a finite set S, a distribution μ over S is a function $\mu : S \to [0, 1]$ satisfying the condition $\mu(S) \leq 1$. The *support* of a distribution μ is the set of states on which μ assumes a non-zero value, i.e. $Supp(\mu) = \{s \in S \mid \mu(s) > 0\}$. The size of μ is defined by $|\mu| = |Supp(\mu)|$. The distribution μ is called *stochastic* if $\mu(S) = 1$, *absorbing* if $\mu(S) = 0$. Otherwise, i.e. $\mu(S) \leq 1$, we say μ is *substochastic*. We sometimes use an auxiliary state (not a *real* state) $\perp \notin S$ and set $\mu(\perp) = 1 - \mu(S)$. If μ is not stochastic we have $\mu(\perp) > 0$. Further, let $S_\perp$ denote the set $S \cup \{\perp\}$, and let $Supp_\perp(\mu) = Supp(\mu) \cup \{\perp\}$ if $\mu(\perp) > 0$ and $Supp_\perp(\mu) = Supp(\mu)$ otherwise. We let $Dist(S)$ denote the set of distributions over the set S.

Definition 2.1.1. *A labelled fully probabilistic system (FPS) is a tuple $\mathcal{M} = (S, \mathbf{P}, L)$ where:*

- *S is a finite set of states,*
- *$\mathbf{P} : S \times S \to [0, 1]$ is a probability matrix such that for all $s \in S$, $\mathbf{P}(s, \cdot) \in Dist(S)$,*
- *$L : S \to 2^{AP}$ is a labelling function.*

A state $s \in S$ is called stochastic, absorbing, and substochastic if the distribution $\mathbf{P}(s, \cdot)$ is stochastic, absorbing, and substochastic respectively. Intuitively, $\mathbf{P}(s, s')$ denotes the probability of moving from s to s' in a single step. For $s \in S$, let $post(s) = Supp(\mathbf{P}(s, \cdot))$, i. e., the set of successor states of s. Let $post_\perp(s) = Supp_\perp(\mathbf{P}(s, \cdot))$, i. e., $post(s)$ plus the auxiliary state $\perp$ in case that $\mathbf{P}(s, \perp) > 0$.

A state s' is reachable from s if it holds that: there exists a sequence of states $s_1, \ldots, s_n$ with $n \geq 1$, $s_1 = s$, $s_n = s'$, and for each $i = 1, \ldots, n - 1$, $\mathbf{P}(s_i, s_{i+1}) > 0$.

Example 2.1.1. As an example consider the FPS in Figure 2.1. The states are represented by circles, and transitions by edges equipped with transition probabilities. Labelling of states is indicated by colours in the circles. For state s_1, we have that $post(s_1) = \{s_1, p_1, p_2\}$ and $post_\perp(s_1) = \{s_1, p_1, p_2, \perp\}$.

If s is not stochastic, it holds that $\mathbf{P}(s, S) < 1$. In this case, the transitions out of s are under-specified. As indicated in [47], under-specification can be used to model behaviour which is unknown or of no interest. We have introduced the auxiliary state $\perp$ to represent this under-specification. $\perp$ is not a real state, it simply represents unknown behaviour. We could think that from state s, with probability $\mathbf{P}(s, \perp) = 1 - \mathbf{P}(s, S) > 0$, the behaviour is under-specified. The following example gives an intuition why this is useful.

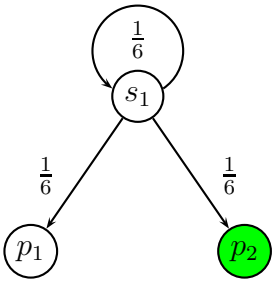

Figure 2.1: A simple FPS.

Example 2.1.2. Assume that a fair die is thrown and two players are observing the outcome of the experiment:

- Player 1 wins if the outcome is 1, loses if the outcome is 2,
- Player 2 wins if the outcome is 2, loses if the outcome is 1,
- The behaviour is unspecified if the outcomes are $3, 4, 5$ or 6, i.e., neither of the players wins or loses.

Consider the FPS depicted on the left part of Figure 2.2. Assume that in state s_1 the die is thrown. Assume that the set of atomic propositions is $AP = \{win_1, win_2\}$, where win_i holds if player i wins for $i = 1, 2$. The labelling function is: $L(s_1) = \emptyset$, $L(p_1) = \{win_1\}$ and $L(p_2) = \{win_2\}$. The outcomes $3, 4, 5, 6$ are of no interest for both player 1 and player 2, thus can be under-specified. Starting from s_1, the probability that win_1 holds in the next state is $\frac{1}{6}$. This probability corresponds to the transition to state p_1, which is the only successor state satisfying win_1. Similarly, the probability that win_2 holds in the next state is $\frac{1}{6}$. Thus, we conclude that the probability that either win_1 or win_2 holds is $\frac{1}{3}$, which corresponds the probability that either player 1 or player 2 wins.

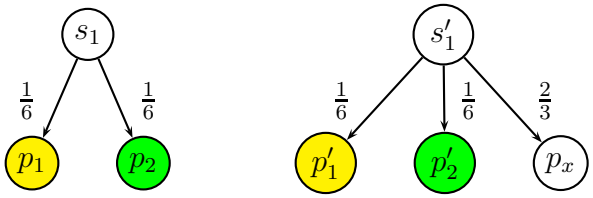

Figure 2.2: A model showing why underspecification is useful in FPSs[1].

Now consider the negation of the atomic propositions $\neg win_1$ and $\neg win_2$. Starting from s_1, the probability that in the next state $\neg win_1$ holds is also $\frac{1}{6}$, since only p_2 satisfies $\neg win_1$. Similarly, the probability that $\neg win_2$ holds in the next state is also $\frac{1}{6}$. Observe the probability that in the next state win_i or $\neg win_i$ holds is $\mathbf{P}(s_1, S) = \frac{1}{3}$ for both $i = 1, 2$.

[1] Although this graph is not connected, it shows a single FPS. Similarly, if not stated explicitly, later figures will show a single DTMC, CTMC etc.

Consider the FPS on the right part of Figure 2.2. We added another state p_x to model the outcomes $3, 4, 5, 6$. Now the question is how should we assign $L(p_x)$? We consider the atomic proposition win_1. Assume first that we have $win_1 \in L(p_x)$. In this case the probability that player 1 wins is $\frac{5}{6}$, which is undesired. It remains to set $win_1 \notin L(p_x)$, which implies, however, that the probability that win_1 or $\neg win_1$ holds would be 1, which contradicts our analysis above (this probability should be $\frac{1}{3}$). Hence, it makes sense to use substochastic distribution to model the outcomes for this situation.

2.2 Discrete-time Markov Chains

If we allow only stochastic or absorbing states in an FPS, we arrive at the notion of discrete-time Markov chains:

Definition 2.2.1. *A labelled discrete-time Markov chain (DTMC) is an FPS $\mathcal{M} = (S, \mathbf{P}, L)$ where s is either absorbing or stochastic for all $s \in S$.*

FPSs and DTMCs are *time-abstract*, since the duration between triggering transitions is disregarded. We observe the state of it only at a discrete set of time points $0, 1, 2, \ldots$. Consider again the FPS in Figure 2.2. Observe that the submodel reachable from state s_1' is a DTMC.

2.3 Continuous-time Markov Chains

Definition 2.3.1. *A labelled continuous-time Markov chain (CTMC) is a tuple $\mathcal{M} = (S, \mathbf{R}, L)$ with S and L as before, and $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ is a rate matrix.*

For CTMC $\mathcal{M}$, let $post(s) = \{s' \in S \mid \mathbf{R}(s, s') > 0\}$ for all $s \in S$. CTMCs are *time-aware*: The rates give the average delay of the corresponding transitions. Starting from state s, the probability that within time t any successor state is chosen is $1 - e^{-\mathbf{R}(s,S)t}$, which is exponentially distributed with rate $\mathbf{R}(s, S) = \sum_{s' \in S} \mathbf{R}(s, s')$. The rate $\mathbf{R}(s, S)$ is also called the exit rate of state s. If $\mathbf{R}(s, s') > 0$ for more than one state s', the outcome of a choice of the successor state is governed by a *race condition*. The probability that the transition from s to s' wins the race, i.e., s move to s' within time t before other successor states of s, is given by:

$$\frac{\mathbf{R}(s, s')}{\mathbf{R}(s, S)} \cdot (1 - e^{-\mathbf{R}(s,S)t}).$$

Embedded DTMCs. A CTMC induces an embedded DTMC, which captures the time-abstract behaviour of it:

Definition 2.3.2. *Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC. We define the embedded DTMC of $\mathcal{M}$ by: $emb(\mathcal{M}) = (S, \mathbf{P}, L)$ where $\mathbf{P}(s, s') = \frac{\mathbf{R}(s,s')}{\mathbf{R}(s,S)}$ if $\mathbf{R}(s, S) > 0$ and 0 otherwise.*

The transition probability $\mathbf{P}(s, s')$ can be considered as the probability of choosing s' as a successor state from s within infinite time. We will also use $\mathbf{P}$ for a CTMC directly, without referring to the probabilistic matrix $\mathbf{P}$ in its embedded DTMC explicitly. A state s' is reachable from s if it is reachable from s in the embedded DTMC. If one is interested in time-abstract properties (e. g., the probability to reach a set of states) of a CTMC, it is sufficient to analyse the embedded DTMC.

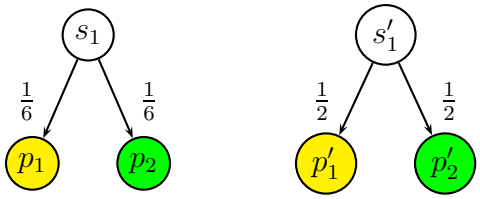

Figure 2.3: A CTMC and its induced DTMC.

Example 2.3.1. In the Figure 2.3 consider the model reachable from state s_1 as a CTMC $\mathcal{M}$. Then, the probability of reaching state p_1 within time 9 is given by $0.5 * (1 - e^{-3})$. The model reachable from s'_1 represents the embedded DTMC $emb(\mathcal{M})$. The probability of reaching p'_1 is 0.5, which corresponds to the limit $\lim_{t\to\infty} 0.5 * (1 - e^{-\frac{1}{3}t})$.

Size of Markov Chains. To measure the complexity of algorithms, we define the size of the models. For a given FPS, DTMC or CTMC, its *fanout* is defined by $\max_{s\in S} |post(s)|$. The number of states is defined by $n = |S|$, and the number of transitions is defined by $m = \sum_{s\in S} |post(s)|$.

2.4 Probabilistic Automata

Markov chains are purely probabilistic. In this subsection, we consider an extension of Markov chains with nondeterminism. The nondeterminism can be used, for example, to abstract certain behaviour of randomised distributed systems. As for the Markov chains, we consider both discrete-time and continuous-time extensions. We first recall the definition of discrete-time probabilistic automata. Just as FPSs, we allow substochastic distributions in our definition. This model can be considered as the *simple probabilistic automata* with transitions allowing deadlocks in [98].

Definition 2.4.1. *A probabilistic automaton (PA) is a tuple $\mathcal{M} = (S, Act, \mathbf{P}, L)$ where S and L are defined as before, Act is a finite set of actions, and $\mathbf{P} \subseteq S \times Act \times Dist(S)$ is a finite set, called the probabilistic transition matrix.*

For $(s, \alpha, \mu) \in \mathbf{P}$, we use $s \xrightarrow{\alpha} \mu$ as a shorthand notation, and call μ an α-successor distribution of s. Let $Act(s) = \{\alpha \mid \exists \mu : s \xrightarrow{\alpha} \mu\}$ denote the set of actions enabled at s. For $s \in S$ and $\alpha \in Act(s)$, let $Steps_\alpha(s) = \{\mu \in Dist(S) \mid s \xrightarrow{\alpha} \mu\}$ denote the set of α-successor distributions of s. Let $Steps(s)$ denote the set $\bigcup_{\alpha\in Act(s)} Steps_\alpha(s)$. A state s' is reachable from s if there exists a sequence $s_1, \alpha_1, \mu_1, \ldots, s_{n-1}, \alpha_{n-1}, \mu_{n-1}, s_n$ with $n \geq 1$, $s_1 = s$, $s_n = s'$ and $s_i \xrightarrow{\alpha_i} \mu_i$ and $\mu_i(s_{i+1}) > 0$ for $i = 0, \ldots, n-1$.

We introduce the notion of *fanout* for $\mathcal{M}$. The fanout of a state s is defined by

$$fan(s) = \sum_{\alpha \in Act(s)} \sum_{\mu \in Steps_\alpha(s)} (|\mu| + 1).$$

Intuitively, $fan(s)$ denotes the total sum of the sizes of outgoing distributions of state s and their labelling. The fanout of $\mathcal{M}$ is defined by $\max_{s \in S} fan(s)$. Summing up over all states, we define the size of the transitions of $\mathbf{P}$ by $m = \sum_{s \in S} fan(s)$.

Markov Decision Processes. A Markov decision process (MDP) [46, 93] arises from a PA $\mathcal{M}$ if for $s \in S$ and $\alpha \in Act$, there is at most one α-successor distribution μ of s which must be stochastic. Formally, an MDP is a PA $\mathcal{M} = (S, Act, \mathbf{P}, L)$ where the probabilistic transition matrix satisfies: $s \xrightarrow{\alpha} \mu$ and $s \xrightarrow{\alpha} \mu'$ implies that $\mu = \mu'$ and that μ is stochastic.

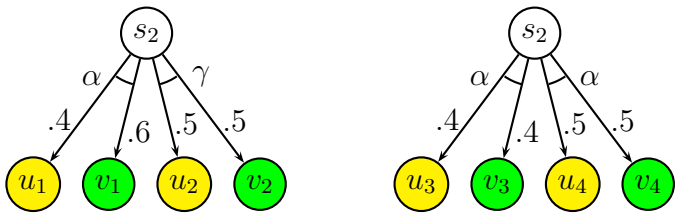

Figure 2.4: A simple PA.

Example 2.4.1. Consider the PA depicted in Figure 2.4. The sub-model reachable from state s_1 is an MDP, whereas reachable from state s_2 is not.

2.5 Continuous-time Probabilistic Automata

We consider a continuous-time counterpart of PAs where the transitions are described by rates instead of probabilities. A rate function is simply a function $r : S \to \mathbb{R}_{\geq 0}$. Let $|r| = |\{s \mid r(s) > 0\}|$ denote the size of r. Let $Rate(S)$ denote the set of all rate functions.

Definition 2.5.1. *A continuous-time PA (CPA) is a tuple $(S, Act, \mathbf{R}, L)$ where S, Act, L are as defined for PAs, and the rate matrix $\mathbf{R} \subseteq S \times Act \times Rate(S)$ a finite set.*

We write $s \xrightarrow{\alpha} r$ if $(s, \alpha, r) \in \mathbf{R}$, and call r an α-successor rate function of s. For transition $s \xrightarrow{\alpha} r$, the sum $r(S)$ is also called the exit rate of it. Let $Act(s) = \{\alpha \mid \exists r : s \xrightarrow{\alpha} r\}$ denote the set of actions enabled at s. For $s \in S$ and $\alpha \in Act(s)$, let $Steps_\alpha(s) = \{r \in Rate(S) \mid s \xrightarrow{\alpha} r\}$ denote the set of α-successor rate functions of s. Let $Steps(s)$ denote the set $\bigcup_{\alpha \in Act(s)} Steps_\alpha(s)$. The fanout of a state s is defined by

$$fan(s) = \sum_{\alpha \in Act(s)} \sum_{r \in Steps_\alpha(s)} (|r| + 1).$$

The fanout of $\mathcal{M}$ is defined by $\max_{s \in S} fan(s)$. The size of $\mathbf{P}$ is $m = \sum_{s \in S} fan(s)$.

Given that the transition $s \xrightarrow{\alpha} r$ is chosen from state s, the probability that any successor state is chosen within time t is given by $1 - e^{-r(S)t}$. A specific successor state s' is chosen within time t is given by $(1 - e^{-r(S)t}) \cdot \frac{r(s')}{r(S)}$.

Continuous-Time Markov Decision Process. Continuous-time Markov decision processes (CTMDPs) [79, 93, 20, 14] can be considered as special CPAs where for $s \in S$ and $\alpha \in Act$, there exists at most one rate function $r \in Rate(S)$ such that $s \xrightarrow{\alpha} r$. Formally, a continuous-time Markov decision process (CTMDP) is a PA $\mathcal{M} = (S, Act, \mathbf{P}, L)$ where the probabilistic transition matrix satisfies: $s \xrightarrow{\alpha} r$ and $s \xrightarrow{\alpha} r'$ implies that $r = r'$. The model CTMDPs considered in paper [120] essentially agree with our CPAs.

Embedded PAs. Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA. Similar to CTMCs, $\mathcal{M}$ induces an embedded PA, which captures the time-abstract behaviour of it. For this purpose we introduce first the induced distribution of a rate function.

Definition 2.5.2. *Let r be a rate function. The induced distribution, denoted by $\mu(r)$, is defined as follows. For the case $r(S) > 0$, $\mu(r)(s)$ equals $\frac{r(s)}{r(S)}$ for $s \in S$. If $r(S) = 0$, $\mu(r) = 0$.*

Definition 2.5.3. *Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA. The embedded PA of $\mathcal{M}$ is defined by $emb(\mathcal{M}) = (S, Act, \mathbf{P}, L)$ with $\mathbf{P} = \{(s, \alpha, \mu(r)) \mid (s, \alpha, r) \in \mathbf{R}\}$.*

We will also use $\mathbf{P}$ for a CPA directly, without referring to its embedded PA explicitly. A state s' is reachable from s if it is reachable from s in the embedded PA. If one is interested in time-abstract properties (e. g., the probability to reach a set of states) of a CPA, it is sufficient to analyse the embedded PA.

2.6 Bibliographic Notes

We briefly recall other probabilistic models considered in the literature. Reactive, generative and stratified probabilistic models are considered in [113, 112]. Reactive models coincide with MDPs, while generative models are DTMCs with actions labels over the probabilistic branchings instead of state labels (with atomic propositions). Stratified models add more information to generative models such that one can have level-wise probabilistic branching. An action- and state-labelled version of CTMCs is considered in [7, 8], where the rate matrix of a CTMC is decorated with an additional set of actions.

In the context of the probabilistic automata considered in [98] a more general notion of transition is considered where both the target states and the action chosen are determined by a probabilistic measure. Moreover, as in this thesis, it is also possible for a transition to deadlock with some probability (corresponding the auxiliary state $\perp$). The PAs in this thesis can be considered as the simple probabilistic automata [98] with transitions allowing deadlocks. In [114, 61], probabilistic systems are considered where states are partitioned into probabilistic states and nondeterministic states, thus

called alternating models. From a probabilistic state, a successor distribution is associated, while from a nondeterministic state ordinary transitions may occur, as in labelled transition systems. We refer to [100] for a nice overview of these discrete-time models.

CTMDPs [79, 93, 20] are extensions of CTMCs with nondeterminism. Recently, CTMDPs have drawn a lot of attentions, especially in model checking problem and logical characterisation of bisimulations [11, 14, 23, 66, 87]. The model CPA generalises CTMDP with internal nondeterminism, i.e., nondeterminism between equally-labelled transitions. The model CTMDPs considered in paper [120] essentially agree with our CPAs. A related continuous-time model is interactive Markov chains (IMCs) [65], which have been recently extended with input and output [27, 26]. In IMCs, each state has at most one successor rate function, but can possess more ordinary transitions.

In [105] and [52], interval-valued discrete-time Markov chains (IDTMCs) are introduced, which are DTMCs in which the transition is specified with an interval. These Markov chains are constructed through statistical experiments. Thus, IDTMCs can be used to model these statistical variations. In [52], IDTMCs are called abstract Markov chains (AMCs), since it is considered as an abstraction of traditional DTMCs. In more detail, given a DTMC and a partition of the state space for the purpose of abstraction, an AMC can be constructed. In the non-probabilistic world, abstraction of transition systems [28] to modal transition systems has been proposed in which there are may and must transitions: may transitions over-approximate and must transitions under-approximate the behavior of the concrete model. While may abstraction preserves safe CTL properties [36] (formulas with positive universal path-quantifiers only), the must abstraction preserves universal liveness CTL properties (formulas with positive existential path-quantifiers only). Similarly, for Markov chains [68, 69, 52], the lower bound corresponds to the may transitions while the upper bound corresponds to the must transitions. Thus, in AMC, both a lower and upper bound of the reachability probability can be obtained. Properties expressed in probabilistic computational tree logic [62] (PCTL) can be analysed in the AMC, in which a 3-valued PCTL semantics is given over AMCs. Then, if the property is satisfied or refuted in the AMC, it is also the case in the original system. However, if we do not know whether the property holds in the AMC, refinement steps are needed in which abstract states are split. Recently [75, 76], the approach was also successfully extended to abstract CTMCs.

Chapter 3

Computing Maximum Flows

The algorithmic workhorse of the simulation algorithms we are going to establish is based on maximum flow computations over bipartite networks. The networks are constructed out of the distributions of a pair of states (s_1, s_2). In this chapter, we first recall the maximum flow problem. Then we recall the augmenting path algorithm, which is one of the simplest and most intuitive algorithms for solving the maximum flow problem. Afterwards we recall the preflow algorithm [56]. The reason is that preflow algorithm can be extended to solve a sequence of related maximum flow problems [53], which is crucial for our improved algorithm: we can re-formulate the checks for a certain pair (s_1, s_2) over all iterations via such sequence of maximum flow problems, thus enable us to get a much better complexity. We also recall the feasible flow problem and the minimum cut which will be used in later chapters.

3.1 Maximum Flow Problems

Let $\mathcal{N} = (V, E, c)$ be a network where V is a finite set of vertices, $E \subseteq V \times V$ is a set of edges, and $c : E \to \mathbb{R}_{>0} \cup \{\infty\}$ is the capacity function. V contains a distinguished *source* vertex $\nearrow$ and a distinguished *sink* vertex $\searrow$. The capacity function c is extended to all vertex pairs by: $c(v, w) = 0$ if $(v, w) \notin E$.

We call $\mathcal{N}$ a bipartite network if V can be partitioned into two subsets V_1, V_2 with $\searrow \in V_1$ and $\nearrow \in V_2$ such that all edges have one endpoint in V_1 and another in V_2. Without loss of generality, we assume that $|V_1| \leq |V_2|$. In this section, we use $n = |V|$, $n_1 = |V_1|$, $n_2 = |V_2|$ and $m = |E|$.

A *flow* [2] f on $\mathcal{N}$ is a function $f : V \times V \to \mathbb{R}$ that satisfies:

1. $f(v, w) \leq c(v, w)$ for all $(v, w) \in V \times V$ — *capacity constraints*
2. $f(v, w) = -f(w, v)$ for all $(v, w) \in V \times V$ — *antisymmetry constraint*
3. $f(V, v) = 0$ at vertices $v \in V \setminus \{\nearrow, \searrow\}$ — *conservation rule*

The domain of the flow function f is $V \times V$. Since for $c(v, w) = 0$ for $(v, w) \notin E$, by the capacity constraint, the flow value $f(v, w)$ is bounded by 0. Precisely, for $(v, w) \notin E$, we have that $-c \leq f(v, w) \leq 0$. The capacity constraint indicates that the amount of

flow we could send along the edge (v, w) is bounded by the capacity $c(v, w)$. Negative flow along an edge (v, w) can be thought as the same amount of positive flow along the reversed edge (w, v). Thus, the antisymmetry constraint 2 states that, given $f(v, w) \geq 0$, the positive flow along the edge (v, w) equals the negative flow along the reversed edge (w, v). The conservation rule can be interpreted as follows: for vertices v which is not the source $\nearrow$ and the sink $\searrow$, the total flow into v is the same as the total flow out of v.

The value of a flow function f is given by $f(\nearrow, V)$, i. e., the total flow out of the source $\nearrow$. A trivial flow function f with $f(v, w) = 0$ for all $v, w \in V$ has value 0. A *maximum flow* is a flow of maximum value.

Example 3.1.1. As an example we consider the bipartite network depicted on the left side of Figure 3.1. The set of vertices V are represented by circles and E by edges between circles. The capacities are depicted as labelling of the edges. The capacity for (u, v) is infinity if no edge label on (u, v) is given. A possible maximum flow is given on the right side of the figure. Obviously the maximum flow function for this network is not unique: the amount of flow out of $\bot$ can also be sent along $\overline{\bot}$.

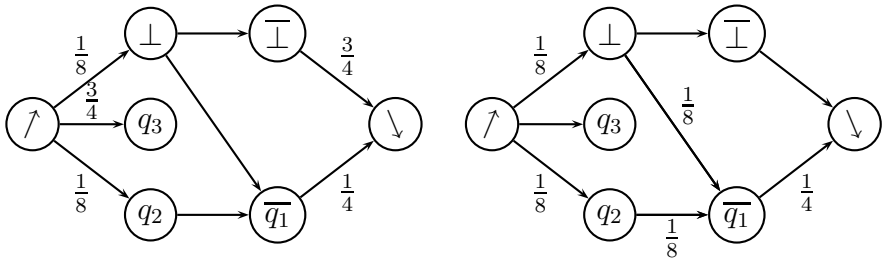

Figure 3.1: A bipartite network and a possible maximum flow for the network.

In the above definition a flow is a function on edges. As an equivalent formulation a flow can be considered as a function on paths from $\nearrow$ to $\searrow$. For example, in Example 3.1.1, the maximum flow sends $\frac{1}{8}$ amount of flow along the path $\nearrow, \bot, \overline{q_1}, \searrow$, $\frac{1}{8}$ amount of flow along the path $\nearrow, q_2, \overline{q_1}, \searrow$. By the flow decomposition theorem [2], each flow function (also called arc flow) can be decomposed into path flow which is not unique, and each path flow can be transformed to a unique arc flow. In this thesis, we use mainly arc flows. In most of the examples, however, we use path flows to describe the uniquely represented arc flow.

3.2 Augmenting Path Algorithm

We recall the augmenting path algorithm [2], which is one of the simplest and most intuitive algorithms for solving maximum flow problems.

Let $\mathcal{N} = (V, E, c)$ be a network and f a flow function over $\mathcal{N}$. A pair (v, w) is a *residual edge* of f if $f(v, w) < c(v, w)$. The set of residual edges with respect to f is denoted by E_f. The *residual capacity* $c_f(v, w)$ of the residual edge (v, w) is defined by $c(v, w) - f(v, w)$. Let $\mathcal{N}_f = (V, E_f, c_f)$ denote the residual network.

Algorithm 1 Computing Maximum Flows: the preflow algorithm.

PREFLOW($\mathcal{N} = (V, E, c), f, d$)

1.1: **repeat**
1.2: **if** exists active node $v \in V$ and $\exists (v, w) \in E_f$. $d(v) = d(w) + 1$ **then**
1.3: $\delta \leftarrow \min\{e(v), c_f(v, w)\}$
1.4: $f(v, w) \leftarrow f(v, w) + \delta; f(w, v) \leftarrow f(w, v) - \delta$
1.5: $e(v) \leftarrow e(v) - \delta; e(w) \leftarrow e(w) + \delta$
1.6: **if** exists active node $v \in V$ and $\forall (v, w) \in E_f$. $d(v) \leq d(w)$ **then**
1.7: $d(v) \leftarrow \min\{d(w) + 1 \mid (v, w) \in E_f\}$
1.8: **until** all nodes in V are not active

A directed path from the source to the sink in the residual network is called an augmenting path. In the augmenting path algorithm, we start with a zero flow function, and continue finding augmenting paths in the current residual network and sending as much flow as possible through this path. If no augmenting paths exist, we have obtained a maximum flow.

Consider again the Example 3.1.1. Assume that the first augmenting path we have found is $\nearrow, \bot, \overline{q_1}, \searrow$, and we send $\frac{1}{8}$ amount of flow along this path. In the residual network, we can send $\frac{1}{8}$ amount of flow along the augmenting path $\nearrow, q_2, \overline{q_1}, \bot, \overline{\bot}, \searrow$. Then no augmenting paths exist and we have a maximum flow with value $\frac{1}{4}$. This maximum flow function is the same as the one depicted on the right part of Figure 3.1.

3.3 The Preflow Algorithm

Let $\mathcal{N} = (V, E, c)$ be a bipartite network. A *preflow* function is a generalisation of a flow function. While a flow function satisfies the capacity constraints, antisymmetry constraint and the flow conservation rule, a preflow function is a function $f : V \times V \to \mathbb{R}$ satisfying the capacity constraints, antisymmetry constraint, and a *relaxed* flow conservation rule:

(3′) $f(V, v) \geq 0$ for all $v \in V \setminus \{\nearrow\}$.

Intuitively, for vertex $v \in V \setminus \{\nearrow\}$, the total amount of flow into v must be larger than or equal to the total amount of flow out of v.

The *excess* $e(v)$ of a vertex v is defined by $f(V, v)$. A vertex $v \notin \{\nearrow, \searrow\}$ is called *active* if $e(v) > 0$. Observe that if in a preflow function no vertex v is active for $v \in V \setminus \{\nearrow, \searrow\}$, it is then also a flow function. If (v, w) is not a residual edge, it is called *saturated*. A *valid distance function* (called *valid labelling* in [56]) d for the bipartite network $\mathcal{N}$ is a function $V \to \mathbb{N} \cup \{\infty\}$ satisfying: $d(\nearrow) = 2n_1$, $d(\searrow) = 0$ and $d(v) \leq d(w) + 1$ for every residual edge (v, w). A residual edge (v, w) is *admissible* if $d(v) = d(w) + 1$. Recall $n_1 = |V_1|$ and $|V_1| \leq |V_2|$.

The preflow algorithm [56] is presented in Algorithm 1. As parameters, the algorithm gets the network $\mathcal{N} = (V, E, c)$, a valid preflow function f, and a valid distance function

d. For bipartite networks, a possible initialisation for the preflow function f could be: $f(v,w) = c(v,w)$ if $v = \nearrow$ and 0 otherwise. And a possible initialisation for the distance function d could be: $d(v) = 2n_1$ if $v = \nearrow$ and 0 otherwise. Note, however, arbitrary valid preflow and distance functions can be used.

The preflow algorithm maintains the valid preflow f and the valid distance function d. There are two basic operations in the algorithm: the *push* operation between lines 1.2–1.5, and the *relabel* operation between lines 1.6–1.7. If there is an active vertex v such that (v,w) is admissible, one may push $\delta := \min\{e(v), c_f(v,w)\}$ amount of flow from v toward the sink along the admissible edge (v,w) by increasing $f(v,w)$ (and decreasing $f(w,v)$) by δ. The excesses of v and w are then modified accordingly by: $e(v) = e(v) - \delta$ and $e(w) = e(w) + \delta$. The push is *saturating* if $c_f(v,w) = 0$ after the push and *non-saturating* otherwise. If v is active but there are no admissible edges leaving it, one may *relabel* v by letting $d(v) := \min\{d(w) + 1 \mid (v,w) \in E_f\}$. Pushing and relabelling are repeated until there are no active vertices left. The algorithm terminates if no such operations apply. The resulting final preflow f is a maximum flow.

Lemma 3.3.1 ([56])**.** *If f is a valid preflow function and d is a valid distance function, the algorithm* Preflow *correctly computes the maximum.*

We begin by stating a few lemmas from [56, 3].

Lemma 3.3.2 ([56])**.** *For any vertex $v \in V$, the distance function $d(v)$ never decreases. An application of a relabelling operation to v strictly increases $d(v)$.*

For bipartite networks, the distance of the source $\nearrow$ is $2n_1$. For arbitrary node v, the distance is bounded by $4n_1$:

Lemma 3.3.3 ([3])**.** *For each active vertex v, $d(v) \leq 4n_1$.*

We first recall the data structures [56, 60] used to get the complexity result. Each node $v \in V$ keeps a list $I(v) \subseteq V$ in arbitrary but fixed order. The set $I(v)$ contains node w such that either $(v,w) \in E$ or $(w,v) \in E$. Intuitively, it represents edges which could be admissible leaving v. At any point of the algorithm, there is a pointer $p(v)$ into the set $I(v)$, which is initialised to the top of $I(v)$. The main loop of the algorithm consists of repeating the push and relabel operations until there are no active nodes. For an active node v, the algorithm pushes flow along the residual edge $(v, p(v))$ if a push operation is applicable, otherwise, advances $p(v)$ each time a new element $I(v)$ is considered. If the bottom element of $I(v)$ is reached, it performs the relabel operation and $p(v)$ is set to the top of $I(v)$. The overall complexity is split into saturating pushes, relabels and non-saturating pushes.

Lemma 3.3.4 ([3])**.** *The number of relabels is bounded by $\mathcal{O}(n_1 n)$. The time spent performing relabels is $\mathcal{O}(n_1 m)$. Further, the time spent in saturating pushes is also $\mathcal{O}(n_1 m)$.*

Proof. By Lemma 3.3.2, the relabel operation to v strictly increases the distance $d(v)$. By Lemma 3.3.3, for arbitrary node v we have $d(v) \leq 4n_1$, which implies that the number of relabel steps for v is bounded by $4n_1$. Summing over all nodes, the number of relabels

altogether is bounded by $\mathcal{O}(n_1 n)$. The time for relabel operations with respect to node v is $(4n_1)|I(v)|$. Altogether, this gives $\sum_{v \in V}((4n_1)|I(v)|) \in \mathcal{O}(n_1 m)$.

Between two consecutive saturating pushes on (v, w), the distances $d(v)$ and $d(w)$ must increase by 2. By Lemmas 3.3.2 and 3.3.3, the number of saturating pushes on edge (v, w) is bounded by $4n_1$. Summing over all edges, the work for saturating pushes is bounded by $\mathcal{O}(mn_1)$. ■

To get a good bound for the number of non-saturating pushes, the order of selecting the active nodes is crucial. We recall the Max-d version [60, 3] of the algorithm in which always the active node with the highest label is selected. Moreover, once an active node v is selected, all of the excess $e(v)$ is pushed until $e(v) = 0$. We recall the analysis in [60] for bipartite networks:

Lemma 3.3.5 ([60])**.** *The Max-d version of the preflow algorithm spent $\mathcal{O}(n_1 n^2)$ time in non-saturating pushes.*

Proof. Once an active node is selected, the excess of the node is pushed until it becomes 0. This implies that, between any two relabel operations, there are at most n active nodes processed (otherwise the algorithm terminates and we get the maximum flow). Also observe that at each time an active node is selected, at most one non-saturating push can occur, which implies that there are at most n non-saturating pushes between node label increases. Since the number of relabel operations is bounded by $\mathcal{O}(n_1 n)$ (Lemma 3.3.4),there can be at most $\mathcal{O}(n_1 n^2)$ non-saturating pushes. ■

Lemma 3.3.6 ([3])**.** *The preflow algorithm for bipartite networks runs in time $\mathcal{O}(n_1 m + n_1 n^2)$.*

3.4 Feasible Flow Problem

Let $A \subseteq E$ be a subset of edges of the network $\mathcal{N} = (V, E, c)$, and define the lower bound function $l : A \to \mathbb{R}_{>0}$ which satisfies $l(e) \leq c(e)$ for all $e \in A$. We address the problem of finding a flow function f satisfying the condition: $f(e) \geq l(e)$ for all $e \in A$. We briefly show that this problem can be reduced to the maximum flow problem [2, p. 169–170].

We can replace a minimum flow requirement on edge $v \to w$ by turning v into a demanding vertex (i. e., a vertex that consumes part of its inflow) and turning w into a supplying vertex (i. e., a vertex that creates some outflow *ex nihilo*). The capacity of edge $v \to w$ is then reduced accordingly.

Now, we are going to look for a flow-like function for the updated network. The function should satisfy the capacity constraints, and the difference between outflow and inflow in each vertex corresponds to its supply or demand, except for $\nearrow$ and $\searrow$. To remove that last exception, we add an edge from $\searrow$ to $\nearrow$ with capacity ∞.

We then apply another transformation to the updated network so that we can apply the maximum flow algorithm. We add new source and target vertices $\nearrow'$ and $\searrow'$. For each supplying vertex s, we add an edge $\nearrow' \to s$ with the same capacity as the supply of the vertex. For each demanding vertex d, we add an edge $d \to \searrow'$ with the same capacity

as the demand of the vertex. In [2] it is shown that the original network has a feasible flow if and only if the transformed network has a flow h that saturates all edges from $\nearrow'$ and all edges to $\searrow'$. The flow h necessarily is a maximum flow, and if there is an h, each maximum flow satisfies the requirement; therefore it can be found by the maximum flow algorithm. An example will be given in Example 6.2.3 in Chapter 6.

3.5 Minimum Cut

Related to maximum flows are minimum cuts. A *cut* of a network $\mathcal{N} = (V, E, c)$ is a partition of V into two disjoint sets (X, X') such that $\nearrow \in X$ and $\searrow \in X'$. The *capacity* of (X, X') is the sum of all capacities of edges from X to X', i. e., $\sum_{v \in X, w \in X'} c(v, w)$. A *minimum cut* is a cut with minimal capacity. The *Maximum Flow Minimum Cut Theorem* [2, Theorem 6.3] states that the capacity of a minimum cut is equal to the value of a maximum flow.

Chapter 4

Simulation Relations

This chapter recapitulates the notions of strong simulation, strong probabilistic simulation, and weak simulation relations. For comparison, we consider also the definition of bisimulations.

While bisimulation relations are equivalence relations on the state space, simulation relations ($\precsim$) are preorders. For labelled transition systems, if $s \precsim s'$ ("s' strongly simulates s") state s' can mimic all stepwise behaviour of s; the converse, i. e., $s' \precsim s$ is not guaranteed, so state s' may perform steps that cannot be matched by s. Thus, if $s \precsim s'$ then every successor of s has a corresponding related successor of s', but the reverse implication does not necessarily hold. In the context of model checking, strong simulation relations can be used to combat the well-known state explosion problem, owed to the preservation of safe CTL formulas [36].

In correspondence to the non-probabilistic setting, a simulation preorder provides the principal ingredients to perform abstraction of DTMCs [80, 72], while preserving safety fragments of the safe PCTL formulas. For DTMCs, $s \precsim s'$ requires that the distribution of s' can match the distribution of s. Correspondence of distributions is naturally defined with the concept of weight functions [72]. In [18], strong simulation relations are also introduced for CTMCs. If $s \precsim s'$, the following two conditions must hold. The first condition requires that $s \precsim s'$ holds in the embedded DTMC of the CTMC. The other condition, called the rate condition, requires that state s' is "faster" than s which manifests itself by a higher exit rate. For CTMCs, strong simulation preserves the safety fragment of CSL formulas.

Simulation relations have been further extended to compare the stepwise behaviour of states in PAs [101]: For $s \precsim s'$, it is required that every successor distribution of s via action α (called α-successor distribution) has a corresponding α-successor distribution of s'. As for DTMCs, in the context of model checking, strong simulation relations preserve safe PCTL formulas [101]. For PAs, strong probabilistic simulation [101] is also introduced which is a relaxation of strong simulation in the sense that it enables convex combinations of multiple distributions belonging to equally labelled transitions. More concretely, it may happen that a state s has an α-successor distribution, which cannot be related by any α-successor distribution of s', yet there exists a so-called α-*combined transition*, a convex combination of several α-successor distributions of s'. Strong probabilistic simulation accounts for this and is thus coarser than simulation,

but still preserves the same class of PCTL-properties as strong simulation does. Strong simulation for CPAs [122] has been introduced which combines the simulation for the embedded PAs and an additional rate condition. The notion of probabilistic simulation relations is also introduced for CPAs. Both strong simulation and strong probabilistic simulation preserve the safe CSL formulas.

As a side contribution, we propose a new equivalent definition of strong (probabilistic) simulation. Our definition is inspired by the definition of strong simulation for labelled Markov processes in [49]. In our new definition, instead of using weight function to relate distributions, we compare the distributions by considering an arbitrary subset of states. Informally, for states s, s' in an FPS, $s \precsim s'$ implies the probability of going to a set of states A from s is smaller than or equal to the probability of going to a related (by the strong simulation relation) set of states from s'. This alternative characterisation makes the simulation relations for Markov chains more understandable for those who are familiar with labelled transition systems. An alternative definition of strong simulation is also provided for CTMCs, PAs and CPAs. Using the same idea, we also provide an equivalent definition of strong probabilistic simulation for PAs and CPAs. Most interestingly, with our new definition, we are able to develop simple proofs of several lemmas involving weight functions.

We also consider weak simulation for Markov chains. As in labelled transition systems, a coarser relation called weak simulation is proposed for Markov chains in which stutter steps (or invisible steps) are allowed. In Markov chains only states are labelled. Only those successor states of s' which can also simulate s can be considered as invisible, thus a transition to that state can be considered as a stutter step. Similarly, also successor states of s which can be simulated by s' can be considered as invisible. The visible part of the successor distributions of s and s' are required to be related by a weight function for the conditional distributions. Weak simulation is strictly coarser than strong simulation for Markov chains, thus allowing further reduction of the state space.

Organisation of this Chapter. We start with the standard definition of simulation in Section 4.1. An alternative, equivalent definition of simulation is given in Section 4.2 which provides another view on simulation. In Section 4.3, we recall the definition of weak simulation for Markov chains. Section 4.4 discusses related work, and Section 4.5 concludes this chapter.

4.1 Standard Definitions

This section gives the standard notion of strong simulations and strong probabilistic simulations. We begin with the definition of weight functions in Subsection 4.1.1. In Subsection 4.1.2 we recall the notion of strong simulation. Strong probabilistic simulation is defined in Subsection 4.1.3.

4.1.1 Weight Functions

Strong simulation requires that each successor distribution of one state have a corresponding successor distribution of the other state. The correspondence of distributions is naturally defined with the concept of *weight functions* [72]. In this subsection, we recall the definition of weight function, and also recall some related lemmas which shall be used later.

Notations. For $s \in S$, let $R(s)$ denote the set $\{s' \in S \mid (s, s') \in R\}$. Similarly, for $s' \in S$, let $R^{-1}(s')$ denote the set $\{s \in S \mid (s, s') \in R\}$. If $(s, s') \in R$, we write also $s\ R\ s'$. For a relation $R \subseteq S \times S$, let $R_\perp = R \cup \{(\perp, s) \mid s \in S_\perp\}$.

Definition 4.1.1. *Let $\mu, \mu' \in Dist(S)$ and $R \subseteq S \times S$. A weight function for (μ, μ') with respect to R is a function $\Delta : S_\perp \times S_\perp \to [0, 1]$ such that*

1. *$\Delta(s, s') > 0$ implies $s\ R_\perp\ s'$,*
2. *$\mu(s) = \Delta(s, S_\perp)$ for $s \in S_\perp$ and*
3. *$\mu'(s') = \Delta(S_\perp, s')$ for $s' \in S_\perp$.*

We write $\mu \sqsubseteq_R \mu'$ if there exists a weight function for (μ, μ') with respect to R.

The first condition requires that only pairs (s, s') in the relation $R_\perp$ have a positive weight. In other words, for $s, s' \in S$ with $s' \notin R_\perp(s)$, it holds that $\Delta(s, s') = 0$. Thus, the second and the third conditions can be simplified as follows:

1. $\mu(s) = \Delta(s, R_\perp(s))$ for $s \in S_\perp$,
2. $\mu'(s') = \Delta(R_\perp^{-1}(s'), s')$ for $s' \in S_\perp$.

Consider condition 3. In case that $s' = \perp$, we have

$$\mu'(\perp) = \Delta(R_\perp^{-1}(\perp), \perp) = \Delta(\perp, \perp) \tag{4.1}$$

which implies that the weight for the pair $(\perp, \perp)$ is always $\mu'(\perp)$. Using Equation 4.1, we show that if $\mu \sqsubseteq_R \mu'$, then $\mu'(S)$ is larger than or equal to $\mu(S)$:

Lemma 4.1.1. *Let $R \subseteq S \times S$. Then, $\mu \sqsubseteq_R \mu'$ implies that $\mu(S) \leq \mu'(S)$.*

Proof. Let Δ be the associated weight function for (μ, μ') with respect to R. Then:

$$\mu(S) = \sum_{s \in S} \sum_{s' \in S_\perp} \Delta(s, s') \overset{(*)}{=} \sum_{s \in S} \sum_{s' \in S} \Delta(s, s') \leq \sum_{s \in S_\perp} \sum_{s' \in S} \Delta(s, s') = \mu'(S)$$

where the equality $(*)$ follows from that $\Delta(s, \perp) = 0$ for all $s \in S$. ■

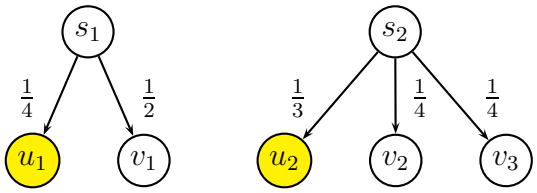

Figure 4.1: An FPS for illustrating the weight functions.

Example 4.1.1. Consider the FPS in Figure 4.1. Assume that $R = \{(u_1, u_2), (u_2, u_1), (v_1, v_2), (v_1, v_3)\}$. A weight function for $(\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot))$ with respect to R can be defined by, for example: $\Delta(u_1, u_2) = \Delta(v_1, v_2) = \Delta(v_1, v_3) = \frac{1}{4}$, $\Delta(\bot, u_2) = \frac{1}{12}$, $\Delta(\bot, \bot) = \frac{1}{6}$.

For sets $R_1, R_2 \subseteq S \times S$, define $R_1 \circ R_2 := \{(s_1, s_2) \mid \exists s \in S.\ (s_1, s) \in R_1 \wedge (s, s_2) \in R_2\}$. Below we give some properties of the weight functions.

Lemma 4.1.2 ([6])**.** *1. Let $\mu_1, \mu_2 \in Dist(S)$ and $R_1, R_2 \subseteq S \times S$ with $R_1 \subseteq R_2$. Then, $\mu_1 \sqsubseteq_{R_1} \mu_2$ implies that $\mu_1 \sqsubseteq_{R_2} \mu_2$.*

2. Let $\mu_1, \mu_2, \mu_3 \in Dist(S)$ and $R_1, R_2 \subseteq S \times S$. Assume that $\mu_1 \sqsubseteq_{R_1} \mu_2$ and $\mu_2 \sqsubseteq_{R_2} \mu_3$, then, $\mu_1 \sqsubseteq_R \mu_3$ where $R = R_1 \circ R_2$.

Proof. 1. Let Δ denote the weight function for (μ_1, μ_2) with respect to R_1, then, Δ is also a weight function for (μ_1, μ_2) with respect to R_2.

2. Let Δ_1 denote the weight function for (μ_1, μ_2) with respect to R_1, and let Δ_2 denote the weight function for (μ_2, μ_3) with respect to R_2. Define Δ by:

$$\Delta(s_1, s_2) = \sum_{s \in Supp_\bot(\mu_2)} \frac{\Delta_1(s_1, s)\Delta_2(s, s_2)}{\mu_2(s)}$$

It is easy to verify that Δ is a weight function for (μ_1, μ_3) with respect to R.

■

For an equivalence relation $R \subseteq S \times S$, the relation $\equiv_R$ on $Dist(S)$ is defined by: $\mu \equiv_R \mu'$ iff $\mu(C) = \mu'(C)\ \forall C \in S/R$. With Lemma 4.1.2, it is easy to show the following lemma:

Lemma 4.1.3 ([18, 72, 6, 47])**.** *Let $R \subseteq S \times S$:*

1. $\sqsubseteq_R$ is reflexive, transitive if R is reflexive, transitive respectively.

2. If R is symmetric and $\mu, \mu' \in Dist(S)$ with $\mu(S) = \mu'(S)$, then $\mu \sqsubseteq_R \mu'$ iff $\mu' \sqsubseteq_R \mu$.

Proof. 1. Assume R is reflexive. For $\mu \in Dist(S)$, define: $\Delta(s, s) = \mu(s)$. Thus, Δ is a weight function for (μ, μ) with respect to R. Assume that R is transitive, and let $\mu_1 \sqsubseteq_R \mu_2$ and $\mu_2 \sqsubseteq_R \mu_3$. By Lemma 4.1.2 it holds that $\mu_1 \sqsubseteq_{R \circ R} \mu_3$. Since R is transitive, we have that $R \circ R \subseteq R$. Again by Lemma 4.1.2, it holds that $\mu_1 \sqsubseteq_R \mu_3$.

2. Assume that $\mu \sqsubseteq_R \mu'$ and let Δ denote the corresponding weight function. Define Δ' by: $\Delta'(s, s') = \Delta(s', s)$. Since R is is symmetric, it is easy to show that Δ' is a weight function for (μ', μ) with respect to R.

■

The proofs of the lemmas below, especially of Lemma 4.1.5, are complicated. Most interestingly, with our new definition, we are able to develop simple proofs of several lemmas involving weight functions. The new proofs of the following two lemmas will be given in Subsection 4.2.1.

Lemma 4.1.4 ([18, 72, 6, 47])**.** *Let $R \subseteq S \times S$ be an equivalence relation. Let $\mu, \mu' \in Dist(S)$.*

1. *$\mu \equiv_R \mu'$ implies that $\mu \sqsubseteq_R \mu'$ and $\mu(S) = \mu'(S)$.*
2. *If $\mu(S) = \mu'(S)$ holds additionally, then $\mu \equiv_R \mu'$ iff $\mu \sqsubseteq_R \mu'$.*

We now recall the Lemma 5.3.5 in [6] which will be used later.

Lemma 4.1.5. *Let R be a preorder on a set S and $\mu, \mu' \in Dist(S)$. If $\mu \sqsubseteq_R \mu'$ and $\mu' \sqsubseteq_R \mu$ then $\mu(A) = \mu'(A)$ for all equivalence classes A with respect to the kernel $R \cap R^{-1}$ of R.*

4.1.2 Strong Simulation

In this subsection, we recall the definitions of strong bisimulations and strong simulations for various models we consider, i. e., FPSs, DTMCs, CTMCs, PAs and CPAs. While strong bisimulation is an equivalence relation over the set of states, strong simulation is a preorder. We will first introduce the definition of strong bisimulations, then strong simulations will be introduced. Alternatively, strong bisimulation can be obtained by restricting the corresponding simulation relation with additional conditions.

FPSs and DTMCs

Definition 4.1.2. *Let $\mathcal{M} = (S, \mathbf{P}, L)$ be an FPS. An equivalence relation $R \subseteq S \times S$ is a strong bisimulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and $\mathbf{P}(s_1, \cdot) \equiv_R \mathbf{P}(s_2, \cdot)$. We say that the states s_1 and s_2 are strongly bisimilar on $\mathcal{M}$, denoted by $s_1 \sim_{\mathcal{M}} s_2$, iff there exists a strong bisimulation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.*

A relation $R \subseteq S \times S$ is a strong simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$. We say that s_2 strongly simulates s_1 in $\mathcal{M}$, denoted by $s_1 \precsim_{\mathcal{M}} s_2$, iff there exists a strong simulation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.

By definition, it is easy to show that $\precsim_{\mathcal{M}}$ is reflexive and transitive, thus a *preorder*. Moreover, $\sim_{\mathcal{M}}$ and $\precsim_{\mathcal{M}}$ are the coarsest strong bisimulation and simulation relation for $\mathcal{M}$ respectively. If $\mathcal{M}$ is clear from the context, we write $\sim$ and $\precsim$ instead of $\sim_{\mathcal{M}}$ and $\precsim_{\mathcal{M}}$ respectively.

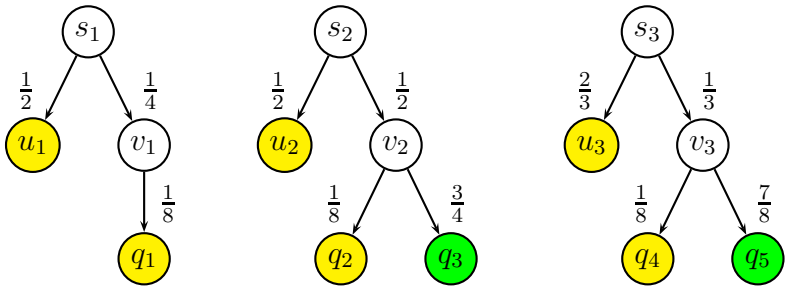

Figure 4.2: An FPS for illustrating the simulation relations.

Observe that if s is absorbing, it holds that $\mathbf{P}(s,\cdot) \sqsubseteq_R \mathbf{P}(s',\cdot)$ for all $(s,s') \in R$: it is sufficient to set $\Delta(\bot, t) = \mathbf{P}(s', t)$ for $t \in S_\bot$. Thus, absorbing state s can be strongly simulated by any state s' if they have the same labelling $L(s) = L(s')$. If R is a strong bisimulation and $(s_1, s_2) \in R$, by definition for each equivalence class $C \in S/R$, we have that $\mathbf{P}(s_1, C) = \mathbf{P}(s_2, C)$, i. e., s_1 and s_2 has the same probability of going to an equivalence class. Let R be a strong simulation relation, and let $(s_1, s_2) \in R$. By Equation 4.1 we have that $\Delta(\bot, \bot) = \mathbf{P}(s_2, \bot)$. Moreover, by Lemma 4.1.1, we have that $\mathbf{P}(s_1, S) \leq \mathbf{P}(s_2, S)$ if s_2 strongly simulates s_1. From the definition and the analysis above we observe that:

- Absorbing states with the same labelling are strongly bisimilar. Any absorbing state and a non-absorbing state are not strongly bisimilar. Moreover, absorbing state can be strongly simulated by any other states with the same labelling.
- From Lemma 4.1.4, we have that $s_1 \sim s_2$ implies both $s_1 \precsim s_2$ and $s_2 \precsim s_1$.
- If s_1 and s_2 are strongly bisimilar, it holds that $\mathbf{P}(s_1, S) = \mathbf{P}(s_2, S)$. Or equivalently, if $\mathbf{P}(s_1, S) \neq \mathbf{P}(s_2, S)$, we can conclude that s_1 and s_2 are not strongly bisimilar.

Example 4.1.2. Consider the FPS depicted in Figure 4.2. Recall that labelling is indicated by colours in the states. Since the yellow (grey) states are absorbing, they strongly simulate each other. The same holds for the green (dark grey) states. We show now that $s_1 \precsim s_2$ but $s_2 \not\precsim s_3$.

First, consider the pair (s_1, s_2). Let $R = \{(s_1, s_2), (u_1, u_2), (v_1, v_2), (q_1, q_2)\}$. We show that R is a strong simulation relation. First observe that $L(s) = L(s')$ for all $(s, s') \in R$. Since states u_1, q_1 are absorbing, the conditions for the pairs (u_1, u_2) and (q_1, q_2) hold trivially. To show the conditions for (v_1, v_2), we consider the function Δ_1 defined by: $\Delta_1(q_1, q_2) = \frac{1}{8}$, $\Delta_1(\bot, q_3) = \frac{3}{4}$, $\Delta_1(\bot, \bot) = \frac{1}{8}$ and $\Delta_1(\cdot) = 0$ otherwise. It is easy to check that Δ_1 is a weight function for $(\mathbf{P}(v_1,\cdot), \mathbf{P}(v_2,\cdot))$ with respect to R. Now consider (s_1, s_2). The weight function Δ_2 for $(\mathbf{P}(s_1,\cdot), \mathbf{P}(s_2,\cdot))$ with respect to R is given by $\Delta_2(u_1, u_2) = \frac{1}{2}$ and $\Delta_2(v_1, v_2) = \Delta_2(\bot, v_2) = \frac{1}{4}$ and $\Delta_2(\cdot) = 0$ otherwise. Thus R is a strong simulation which implies that $s_1 \precsim s_2$.

Consider the pair (s_2, s_3). Since $\mathbf{P}(s_2, v_2) = \frac{1}{2}$, to establish the Condition 2 of Definition 4.1.1, we should have $\frac{1}{2} = \Delta(v_2, S_\bot)$. Observe that v_3 is the only successor

state of s_3 which can strongly simulate v_2, thus $\Delta(v_2, S_\perp) = \Delta(v_2, v_3)$. However, for state v_3 we have $\mathbf{P}(s_3, v_3) < \Delta(v_2, v_3)$, which violates the Condition 3 of Definition 4.1.1, thus we cannot find such a weight function. Hence, $s_2 \not\precsim s_3$.

Let $\simeq_\mathcal{M}$ (or $\simeq$ if $\mathcal{M}$ is clear from the context) denote the kernel of $\precsim_\mathcal{M}$, i. e., strong simulation equivalence $\simeq = \precsim \cap \succsim$. Given the FPS $\mathcal{M}$, $\simeq_\mathcal{M}$ is also called the simulation equivalence relation on $\mathcal{M}$. For both FPSs and DTMCs, the simulation equivalence is the same as the bisimulation equivalence.

Lemma 4.1.6 ([15, 18])**.** *Let $\mathcal{M} = (S, \mathbf{P}, L)$ be an FPS. It holds that $\sim_\mathcal{M} = \simeq_\mathcal{M}$.*

Proof. Let s_1 and s_2 be two simulation equivalent states. $s_1 \precsim s_2$ implies that $\mathbf{P}(s_1, S) \leq \mathbf{P}(s_2, S)$. Symmetrically, we have $\mathbf{P}(s_2, S) \leq \mathbf{P}(s_1, S)$. Thus $\mathbf{P}(s_1, S) = \mathbf{P}(s_2, S)$. Applying Lemma 4.1.4 we have that $\mathbf{P}(s_1, \cdot) \equiv_\simeq \mathbf{P}(s_2, \cdot)$, thus $s_1 \sim s_2$. The other direction is trivial. ■

Continuous-Time Markov Chains

For CTMCs, states s_1 and s_2 are strongly bisimilar, if they move to other equivalence class with the same probability. We say that s_2 strongly simulates s_1 if, in addition to the simulation conditions for DTMCs, s_2 can move stochastically *faster* than s_1 [15, 18], which manifests itself by a higher rate. Recall for a CTMC $\mathcal{M} = (S, \mathbf{R}, L)$, we use $\mathbf{P}$ to denote the probability matrix of the embedded DTMC $emb(\mathcal{M})$.

Definition 4.1.3. *Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC. An equivalence relation $R \subseteq S \times S$ is a strong bisimulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$, $\mathbf{P}(s_1, \cdot) \equiv_R \mathbf{P}(s_2, \cdot)$ and $\mathbf{R}(s_1, S) = \mathbf{R}(s_2, S)$. We say that the states s_1 and s_2 are strongly bisimilar on $\mathcal{M}$, denoted by $s_1 \sim_\mathcal{M} s_2$, iff there exists a strong bisimulation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.*

A relation $R \subseteq S \times S$ is a strong simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$, $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$ and $\mathbf{R}(s_1, S) \leq \mathbf{R}(s_2, S)$. We say that s_2 strongly simulates s_1 in $\mathcal{M}$, denoted by $s_1 \precsim_\mathcal{M} s_2$, iff there exists a strong simulation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.

By definition, $\sim_\mathcal{M}$ and $\precsim_\mathcal{M}$ are the coarsest strong bisimulation and simulation relation for $\mathcal{M}$ respectively. Again, if the model $\mathcal{M}$ is clear from the context, the subscript $\mathcal{M}$ may be omitted. Since $\mathbf{P}(s, C) = \frac{\mathbf{R}(s,C)}{\mathbf{R}(s,S)}$ for state s and $C \subseteq S$, for strongly bisimilar states s_1, s_2, the condition can be reformulated by: an equivalence relation R is a strong bisimulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$, $\mathbf{R}(s_1, C) = \mathbf{R}(s_2, C)$ for all equivalence class $C \in S/R$. It is easy to verify that the two definitions are the same. Observe also $s \precsim_\mathcal{M} s'$ holds if $s \precsim_{emb(\mathcal{M})} s'$ and s' is faster than s. Similar to FPSs, we list several properties of (bi-)simulations on a CTMC $\mathcal{M}$:

- Absorbing states with the same labelling are strongly bisimilar. Any absorbing state and a non-absorbing state are not strongly bisimilar. Moreover, absorbing state can be strongly simulated by any other states with the same labelling.

- $s_1 \sim s_2$ implies both $s_1 \precsim s_2$ and $s_2 \precsim s_1$.

- If $\mathbf{R}(s_1, S) \neq \mathbf{R}(s_2, S)$, we can conclude that s_1 and s_2 are not strongly bisimilar.

The first and second properties concerning absorbing states hold for all bisimulation and simulation relations we shall introduce later for PAs and CPAs, also for weak simulations for Markov chains.

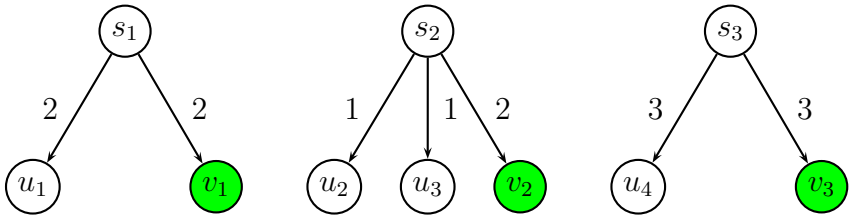

Figure 4.3: A CTMC for illustrating the simulation relations.

Example 4.1.3. In the CTMC depicted in Figure 4.3, the u_i-states are absorbing and have the same labelling, thus strongly bisimilar. Also the green (dark grey) v-states. States s_1 and s_2 are strongly bisimilar: they both go to the u-states with rates 2 and the green (dark grey) states with rates 2. States s_1 (or s_2) are s_3 are not strongly bisimilar, since $\mathbf{R}(s_1, S) \neq \mathbf{R}(s_3, S)$. However, it holds that $s_1 \precsim s_3$ and $s_2 \precsim s_3$.

For CTMC $\mathcal{M} = (S, \mathbf{R}, L)$, let $\simeq_{\mathcal{M}}$ (or $\simeq$ if $\mathcal{M}$ is clear from the context) denote the kernel of $\precsim_{\mathcal{M}}$, i. e., strong simulation equivalence $\simeq = \precsim \cap \succsim$. Similar to Lemma 4.1.6 for FPSs, the simulation equivalence is the same as the bisimulation equivalence for CTMCs.

Lemma 4.1.7 ([15, 18])**.** *Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC. It holds that $\sim_{\mathcal{M}} = \simeq_{\mathcal{M}}$.*

Proof. Let s_1 and s_2 be two simulation equivalent states, i.e., $s_1 \simeq s_2$. $s_1 \precsim s_2$ implies that $\mathbf{P}(s_1, S) \leq \mathbf{P}(s_2, S)$. Symmetrically, we have $\mathbf{P}(s_2, S) \leq \mathbf{P}(s_1, S)$. Thus $\mathbf{P}(s_1, S) = \mathbf{P}(s_2, S)$. Applying Lemma 4.1.4 we have that $\mathbf{P}(s_1, \cdot) \equiv_{\simeq} \mathbf{P}(s_2, \cdot)$, thus $s_1 \sim s_2$. The other direction is trivial. ■

PAs and CPAs

Now we consider the strong (bi-)simulations for PAs. Strong bisimulation R requires that R is an equivalence relation and for $(s_1, s_2) \in R$, every α-successor distribution μ_1 of s_1 is related to an α-successor distribution μ_2 of s_2 such that $\mu_1(S) \equiv_R \mu_2(S)$. Strong simulation R requires that if $(s_1, s_2) \in R$, every α-successor distribution μ_1 of s_1 is related to an α-successor distribution μ_2 of s_2 via a weight function [101, 72] ($\mu_1 \sqsubseteq_R \mu_2$).

Definition 4.1.4. *Let $\mathcal{M} = (S, Act, \mathbf{P}, L)$ be a PA. An equivalence relation $R \subseteq S \times S$ is a strong bisimulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and if $s_1 \xrightarrow{\alpha} \mu_1$ then there exists a transition $s_2 \xrightarrow{\alpha} \mu_2$ with $\mu_1 \equiv_R \mu_2$. We say that states s_1 and s_2 are strong bisimilar in $\mathcal{M}$, denoted $s_1 \sim_{\mathcal{M}} s_2$, iff there exists a strong bisimulation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.*

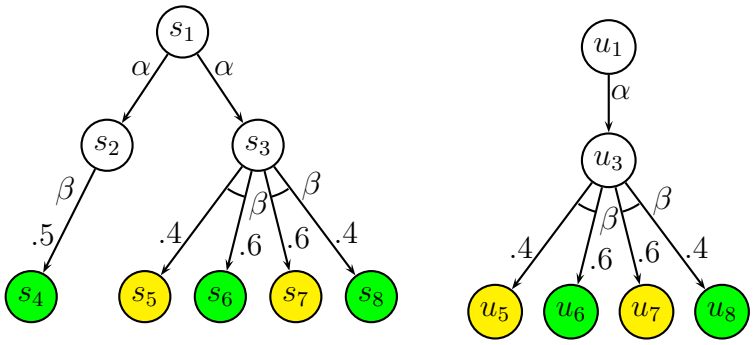

Figure 4.4: An PA for illustrating the simulation relations.

A relation $R \subseteq S \times S$ is a strong simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and if $s_1 \xrightarrow{\alpha} \mu_1$ then there exists a transition $s_2 \xrightarrow{\alpha} \mu_2$ with $\mu_1 \sqsubseteq_R \mu_2$. We say that s_2 strongly simulates s_1 in $\mathcal{M}$, denoted $s_1 \precsim_{\mathcal{M}} s_2$, iff there exists a strong simulation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.

By definition, $\sim_{\mathcal{M}}$ and $\precsim_{\mathcal{M}}$ are the coarsest strong bisimulation and simulation relation for $\mathcal{M}$ respectively. If $\mathcal{M}$ is clear from the context, we write $\sim$ and $\precsim$ instead of $\sim_{\mathcal{M}}$ and $\precsim_{\mathcal{M}}$ respectively. Observe that by Lemma 4.1.4, $\mu_1 \equiv_R \mu_2$ implies that $\mu_1 \sqsubseteq_R \mu_2$ and that $\mu_1(S) = \mu_2(S)$. For a PA $\mathcal{M}$, let $\simeq_{\mathcal{M}}$ (or $\simeq$ if $\mathcal{M}$ is clear from the context) denote the kernel of $\precsim_{\mathcal{M}}$, i. e., strong simulation equivalence $\simeq = \precsim \cap \succsim$. We say that states s, s' are strongly similar, if $s \simeq s'$. Now we give some properties of strong (bi-)simulations for PAs:

- $s_1 \sim s_2$ implies that $s_1 \simeq s_2$, which implies both $s_1 \precsim s_2$ and $s_2 \precsim s_1$. Moreover, $\simeq$ is strictly coarser than $\sim$.
- If s_1 and s_2 are strongly bisimilar, it holds that for each transition $s_1 \xrightarrow{\alpha} \mu_1$ there must exist a transition $s_2 \xrightarrow{\alpha} \mu_2$ such that $\mu_1(S) = \mu_2(S)$. Or equivalently, if no such transition from s_2 exists, we can conclude that s_1 and s_2 are not strongly bisimilar.

We consider the following example illustrating that strong simulation equivalences are strictly coarser than strong bisimulation equivalences:

Example 4.1.4. Consider the PA depicted in Figure 4.4. States s_1 and u_1 are not strongly bisimilar: the state u_3 is not bisimilar with state s_2. However, we show that it holds that $s_1 \precsim u_1$, $u_1 \precsim s_1$. Firstly, it holds that $s_3 \sim u_3$, which implies that $s_3 \precsim u_3$ and $s_3 \precsim u_3$. Thus it holds that $u_1 \precsim s_1$. Furthermore, $s_2 \precsim u_3$, which implies that $s_1 \precsim u_1$. Hence, $s_1 \simeq u_1$.

Now we consider CPAs. Based on the definition of bisimulation and simulation relation for PAs, we introduce the corresponding relation for CPAs [87, 122]:

Definition 4.1.5. *Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA. An equivalence relation $R \subseteq S \times S$ is a strong bisimulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and if $s_1 \xrightarrow{\alpha} r_1$*

then there exists a transition $s_2 \xrightarrow{\alpha} r_2$ *with* $\mu(r_1) \equiv_R \mu(r_2)$ *and* $r_1(S) = r_2(S)$. *We write* $s_1 \sim_{\mathcal{M}} s_2$ *iff there exists a strong bisimulation* R *on* $\mathcal{M}$ *such that* $s_1 \; R \; s_2$.

A relation $R \subseteq S \times S$ *is a strong simulation on* $\mathcal{M}$ *iff for all* s_1, s_2 *with* $s_1 \; R \; s_2$: $L(s_1) = L(s_2)$ *and if* $s_1 \xrightarrow{\alpha} r_1$ *then there exists a transition* $s_2 \xrightarrow{\alpha} r_2$ *with* $\mu(r_1) \sqsubseteq_R \mu(r_2)$ *and that* $r_1(S) \leq r_2(S)$. *We say that* s_2 *strongly simulates* s_1 *in* $\mathcal{M}$, *denoted* $s_1 \precsim_{\mathcal{M}} s_2$, *iff there exists a strong simulation* R *on* $\mathcal{M}$ *such that* $s_1 \; R \; s_2$.

Again, by definition, $\sim_{\mathcal{M}}$ and $\precsim_{\mathcal{M}}$ are the coarsest strong bisimulation and simulation relation for $\mathcal{M}$ respectively. Similar to CTMCs, the additional rate condition $r_1(S) \leq r_2(S)$ for strong simulation indicates that the transition $s_2 \xrightarrow{\alpha} r_2$ is faster than $s_1 \xrightarrow{\alpha} r_1$. As a shorthand notation, we use $r_1 \sqsubseteq_R r_2$ for the condition $\mu(r_1) \sqsubseteq_R \mu(r_2)$ and $r_1(S) \leq r_2(S)$.

For a CPA $\mathcal{M}$, let $\simeq_{\mathcal{M}} = \precsim_{\mathcal{M}} \cap \succsim_{\mathcal{M}}$ denote the kernel of $\precsim_{\mathcal{M}}$. We say that states s, s' are strongly similar, if $s \simeq_{\mathcal{M}} s'$. If the model $\mathcal{M}$ is clear from the context, the subscript $\mathcal{M}$ may be omitted. As for PAs, strong simulation equivalences are strictly coarser than strong bisimulations. Moreover, $s_1 \sim s_2$ implies that $s_1 \simeq s_2$, which implies both $s_1 \precsim s_2$ and $s_2 \precsim s_1$.

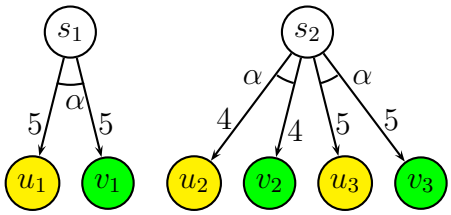

Figure 4.5: An PA for illustrating the simulation relations.

Example 4.1.5. In the CPA depicted in Figure 4.5, states s_1 and s_2 are strongly similar, but not strong bisimilar. However, observe that s_1 and s_2 are strongly bisimilar in the induced PA.

4.1.3 Strong Probabilistic Simulations

For PAs and CPAs, a coarser relation than strong bisimulation and strong simulation can be established. We recall the definition of strong probabilistic bisimulation and simulation, which is coarser than strong bisimulation and simulation respectively, but still preserves the same class of PCTL-properties. The definition is based on the concept of combined transitions [101] which are convex combinations of several equally labelled transitions:

Definition 4.1.6. *Let* $\mathcal{M} = (S, Act, \mathbf{P}, L)$ *be a PA. Let* $s \in S$, $\alpha \in Act(s)$ *and* $k = |Steps_\alpha(s)|$. *Assume that* $Steps_\alpha(s) = \{\mu_1, \ldots, \mu_k\}$. *The tuple* (s, α, μ) *is a combined transition, denoted by* $s \overset{\alpha}{\rightsquigarrow} \mu$, *iff there exist constants* $c_1, \ldots, c_k \in [0,1]$ $\sum_{i=1}^{k} c_i = 1$ *such that* $\mu = \sum_{i=1}^{k} c_i \mu_i$.

The key difference to Definition 4.1.4 is the use of $\overset{\alpha}{\rightsquigarrow}$ instead of $\xrightarrow{\alpha}$:

Definition 4.1.7. *Let $\mathcal{M} = (S, Act, \mathbf{P}, L)$ be a PA. An equivalence relation $R \subseteq S \times S$ is a strong probabilistic bisimulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and if $s_1 \xrightarrow{\alpha} \mu_1$ then there exists a combined transition $s_2 \overset{\alpha}{\rightsquigarrow} \mu_2$ with $\mu_1 \equiv_R \mu_2$. We write $s_1 \sim^p_{\mathcal{M}} s_2$ iff there exists a strong probabilistic bisimulation relation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.*

A relation $R \subseteq S \times S$ is a strong probabilistic simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and if $s_1 \xrightarrow{\alpha} \mu_1$ then there exists a combined transition $s_2 \overset{\alpha}{\rightsquigarrow} \mu_2$ with $\mu_1 \sqsubseteq_R \mu_2$. We write $s_1 \precsim^p_{\mathcal{M}} s_2$ iff there exists a strong probabilistic simulation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.

Strong probabilistic bisimulation and simulation are insensitive to combined transitions[1], thus, they are relaxations of strong simulation. Similar to strong bisimulations and strong simulations, $\sim^p_{\mathcal{M}}$ and $\precsim^p_{\mathcal{M}}$ are the coarsest strong probabilistic bisimulation and simulation for $\mathcal{M}$ respectively. As usual, the kernel of strong probabilistic simulation is denoted by $\simeq^p_{\mathcal{M}}$. Subscripts are omitted if the model is clear from the context. Since MDPs can be considered as special PAs, we obtain the notions of strong simulation and strong probabilistic simulation for MDPs. Moreover, strong simulation and strong probabilistic simulation trivially coincide for MDPs as, by definition, for each state there is at most one successor distribution per action.

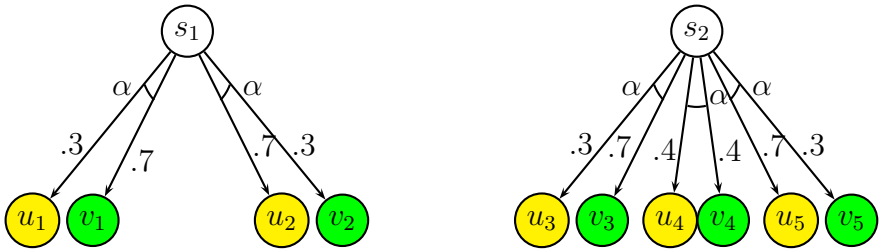

Figure 4.6: For PAs: strong probabilistic simulation is strictly coarser than strong simulation.

Example 4.1.6. We consider the PA in Figure 4.6. State s_2 strong simulates s_1, however, s_1 and s_2 are not strongly similar, as the middle α-successor distribution of s_2 (denoted by μ) cannot be related by any α-successor distribution of s_1. This distribution μ can be related by the combined transition of the two α-successor distributions of s_1 (denoted by μ_1 and μ_2): $0.5\mu_1 + 0.5\mu_2$. Hence, we have $s_2 \precsim^p s_1$. Observe that in this example it holds also: $s_1 \simeq^p s_2$ but $s_1 \not\approx s_2$, and $s_1 \not\sim^p s_2$.

The above example shows that strong probabilistic bisimulation is strictly coarser than strong bisimulation, and strong probabilistic simulation is strictly coarser than strong simulation.

[1]The combined transition defined in [98] is more general in two dimensions: First, successor distributions are allowed to combine different actions. Second, $\sum_{i=1}^{k} c_i \leq 1$ is possible. The induced strong probabilistic bisimulations equivalence and strong probabilistic simulation preorder are, however, the same.

We extend the notion of strong probabilistic simulation for PAs to CPAs. First, we introduce the notion of combined transitions for CPAs. In CPAs the probability that a transition occurs is exponentially distributed. The combined transition should also be exponentially distributed. The following example shows that a straightforward extension of Definition 4.1.6 does not work.

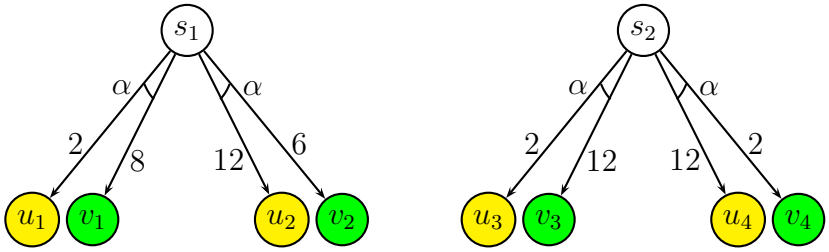

Figure 4.7: Combined transitions for CPAs.

Example 4.1.7. We consider the CPA in Figure 4.7. Let r_1 and r_2 denote left and the right α-successor rate functions out of state s_1. Obviously, they have different exit rates: $r_1(S) = 10$, $r_2(S) = 18$. Taking each with probability 0.5, we would get the combined transition $r = 0.5r_1 + 0.5r_2$: $r(\{u_1, u_2\}) = 7$ and $r(\{v_1, v_2\}) = 7$. However, r is hyper-exponentially distributed: the probability of reaching yellow (grey) states (u_1 or u_2) within time t under r is given by:

$$0.5 \cdot \frac{2}{10} \cdot (1 - e^{-10t}) + 0.5 \cdot \frac{12}{18} \cdot (1 - e^{-18t})$$

Similarly, the probability of reaching green (dark grey) states within time t is given by: $0.5 \cdot \frac{8}{10} \cdot (1 - e^{-10t}) + 0.5 \cdot \frac{6}{18} \cdot (1 - e^{-18t})$.

From state s_2, the two α-successor rate functions have the same exit rate 14. Let r'_1 and r'_2 denote left and the right α-successor rate functions out of state s_2. In this case the combined transition $r' = 0.5r'_1 + 0.5r'_2$ is also exponentially distributed with rate 14: the probability to reach yellow (grey) states (u_3 and u_4) within time t is $\frac{7}{14} \cdot (1 - e^{-14t})$, which is the same as the probability of reaching green (dark grey) states (v_3 and v_4) within time t.

Based on the above example, it is easy to see that to get a combined transition which is still exponentially distributed, we must consider rate functions with the same exit rate:

Definition 4.1.8. *Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA. Let $s \in S$, $\alpha \in Act(s)$ and let $\{r_1, \ldots, r_k\} \subseteq Steps_\alpha(s)$ where $r_i(S) = r_j(S)$ for $i, j \in \{1, \ldots, k\}$. The tuple (s, α, r) is a combined transition, denoted by $s \overset{\alpha}{\rightsquigarrow} r$, iff there exist constants $c_1, \ldots, c_k \in [0, 1]$ with $\sum_{i=1}^{k} c_i = 1$ such that $r = \sum_{i=1}^{k} c_i r_i$.*

In the above definition, unlike for the PA case, only α-successor rate functions with same exit rate are combined together. Similar to PAs, strong probabilistic bisimulation [101] and simulation [122] is insensitive to combined transitions, which is thus a relaxation of strong simulation:

Definition 4.1.9. *Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA. An equivalence relation $R \subseteq S \times S$ is a strong probabilistic bisimulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and if $s_1 \xrightarrow{\alpha} r_1$ then there exists a combined transition $s_2 \overset{\alpha}{\rightsquigarrow} r_2$ with $\mu(r_1) \equiv_R \mu(r_2)$ and $r_1(S) = r_2(S)$. We write $s_1 \sim^p_{\mathcal{M}} s_2$ iff there exists a strong bisimulation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.*

A relation $R \subseteq S \times S$ is a strong probabilistic simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and if $s_1 \xrightarrow{\alpha} r_1$ then there exists a combined transition $s_2 \overset{\alpha}{\rightsquigarrow} r_2$ with $r_1 \sqsubseteq_R r_2$. We write $s_1 \precsim^p_{\mathcal{M}} s_2$ iff there exists a strong simulation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.

Recall $r_1 \sqsubseteq_R r_2$ is a shorthand notation for $\mu(r_1) \sqsubseteq_R \mu(r_2)$ and $r_1(S) \leq r_2(S)$.

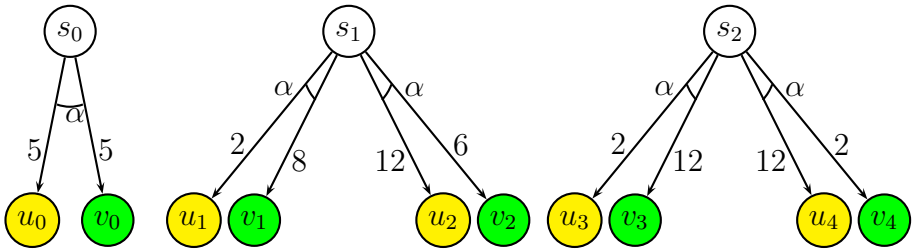

Figure 4.8: A CPA for illustrating strong simulation and strong probabilistic simulation relations.

Example 4.1.8. Consider the CPA in Figure 4.8. As discussed in Example 4.1.7, the two α-successor rate functions of s_1 can not combined together, thus the relation $s_0 \precsim^p s_1$ cannot be established. However, $s_0 \precsim^p s_2$ holds: denoting the left rate function of s_2 as r_1 and the right rate function as r_2, as the combined rate function we choose $r = 0.5r_1 + 0.5r_2$. Obviously, the conditions in Definition 4.1.9 are satisfied.

4.2 Alternative Simulation Definitions

In [49], strong simulation for labelled Markov processes (MDPs with continuous state space) is introduced without using weight functions. Their definition gives another view on strong simulation in the probabilistic setting. For DTMCs, a binary relation $R \subseteq S \times S$ is a strong simulation if R is reflexive and transitive (a preorder), and for $(s, s') \in R$ it holds that: the probability $\mu(A)$ is smaller than or equal to the probability $\mu'(A)$ where A is any closed set under the preorder R.

The requirement that R must be a preorder is, however, not necessary. In this section, we extend their definition to give an alternative definition for strong simulations. We do not impose the condition that R must be a preorder. The new definition of strong simulation is much closer to the definition of strong simulation for labelled transition systems: R is a strong simulation relation if for $(s, s') \in R$ and for each α-successor t of s, s' has also an α-successor t' such that $(t, t') \in R$. Thus the alternative characterisation of strong simulation relations for Markov chains is more understandable, especially for those who are familiar with labelled transition systems.

This section is organised as follows. In Subsection 4.2.1 we recall an equivalent condition for weight functions. Then, we give the new definition for strong and strong probabilistic simulations in Subsection 4.2.2.

Notations. For a relation $R \subseteq S \times S$ and a subset $A \subseteq S$, we let $R(A)$ denote the union $\cup_{s \in A} R(s)$, let $R^{-1}(A)$ denote the union $\cup_{s \in A} R^{-1}(s)$. We use $Dom(R) = \{u \mid \exists v.\ (u,v) \in R\}$ to denote the domain of R, and use $Ran(R) = \{v \mid \exists u.\ (u,v) \in R\}$ to denote the range of R.

4.2.1 Characterising Weight Functions

In [9], weight functions are characterised equivalently using a maximum flow problem. We recall this equivalent characterisation. Furthermore, we propose a new equivalent formulation which is the basis of our alternative definition for strong simulations.

The Network for (μ, μ') with respect to R. Let $R \subseteq S \times S$, and let $\mu, \mu' \in Dist(S)$ be distributions. The network $\mathcal{N}(\mu, \mu', R) := (V, E, c)$ is constructed out of μ_1, μ_2 and R. We consider two copies $t \in S_\perp$ and $\overline{t} \in \overline{S_\perp}$ of each state where $\overline{S_\perp} = \{\overline{t} \mid t \in S_\perp\}$. Moreover, let $\nearrow$ (the source) and $\searrow$ (the sink) be two additional vertices not contained in $S_\perp \cup \overline{S_\perp}$. The set of vertices is defined by:

$$V = \{\nearrow, \searrow\} \cup Supp_\perp(\mu) \cup \overline{Supp_\perp(\mu')}$$

and the set of edges E is defined by

$$E = \{(s, \overline{t}) \mid (s,t) \in R_\perp\} \cup \{(\nearrow, s)\} \cup \{(\overline{t}, \searrow)\}\ .$$

where $s \in Supp_\perp(\mu)$ and $t \in Supp_\perp(\mu')$. Recall the relation $R_\perp$ is defined by $R \cup \{(\perp, s) \mid s \in S_\perp\}$. The capacity function c is defined as follows: $c(\nearrow, s) = \mu(s)$ for all $s \in Supp_\perp(\mu)$, $c(\overline{t}, \searrow) = \mu'(t)$ for all $t \in Supp_\perp(\mu')$, and $c(s, \overline{t}) = \infty$ for all other $(s, \overline{t}) \in E$. Later, we will use a variant of this network: For $\gamma \in \mathbb{R}_{>0}$, we let $\mathcal{N}(\mu, \gamma\mu', R)$ denote the network obtained from $\mathcal{N}(\mu, \mu', R)$ be setting the capacities to the sink $\searrow$ by: $c(\overline{t}, \searrow) = \gamma\mu'(t)$. This network is bipartite, since the vertices can be partitioned into two subsets $V_1 := Supp_\perp(\mu) \cup \{\searrow\}$ and $V_2 := \overline{Supp_\perp(\mu')} \cup \{\nearrow\}$ such that all edges have one endpoint in V_1 and another in V_2.

Now we give another equivalent characterisation of weight functions.

Lemma 4.2.1. *Let $R \subseteq S \times S$ be a relation on S, and let $\mu_1, \mu_2 \in Dist(S)$. The following statements are equivalent:*

1. *There exists a weight function for (μ_1, μ_2) with respect to R.*
2. *The maximum flow of the network $\mathcal{N}(\mu, \mu', R)$ is 1.*
3. *$\mu_1(A) \leq \mu_2(R(A))$ for all $A \subseteq S$.*
4. *$\mu_1(A) \leq \mu_2(R(A))$ for all $A \subseteq Supp(\mu_1)$.*

Proof. The equivalence between 1 and 2 is a direct extension of Lemma 5.1 in [9] now accounting for substochasticity.

$(1 \Longrightarrow 3)$: Let Δ denote the corresponding weight function for (μ_1, μ_2) with respect to R. Now we want to prove that for every $A \subseteq S$: $\mu_1(A) \leq \mu_2(R(A))$. From the second condition of weight function (Definition 4.1.1) it holds that $\mu_1(A) = \sum_{u \in A} \Delta(u, R_\perp(u))$. Since $u \neq \perp$, $\Delta(u, \perp) = 0$ for $u \in A$, thus $\mu_1(A) = \sum_{u \in A} \Delta(u, R(u))$. We observe for $u \notin Dom(R)$, it holds that $R(u) = \emptyset$, thus $\mu_1(A) = \sum_{u \in A \cap Dom(R)} \Delta(u, R(v))$. Moreover, for $u \in A \cap Dom(R)$ and $v \in S_\perp \setminus R(A \cap Dom(R))$ we have that $(u, v) \notin R$, which implies that $\Delta(u, v) = 0$ from the first condition of weight function. We get:

$$\mu_1(A) = \sum_{u \in A \cap Dom(R)} \sum_{v \in R(A \cap Dom(R))} \Delta(u, v) \stackrel{(*)}{=} \sum_{u \in A \cap Dom(R)} \sum_{v \in R(A)} \Delta(u, v) \tag{4.2}$$

Observe that it holds always that $R(A \cap Dom(R)) = R(A)$, thus (*) holds. Similarly, from the first and the third conditions of weight function, we have that

$$\mu_2(R(A)) = \sum_{u \in R_\perp^{-1}(R(A))} \sum_{v \in R(A)} \Delta(u, v) \tag{4.3}$$

Comparing Equations 4.2 and 4.3, it is sufficient to show the following inclusion:

$$A \cap Dom(R) \subseteq R_\perp^{-1}(R(A))$$

Let $u \in A \cap Dom(R)$. Since $u \in Dom(R)$, there exists $v \in Ran(R)$ with $(u, v) \in R$. Thus $v \in R(A \cap Dom(R)) = R(A)$. Again, $(u, v) \in R$ implies that $u \in R^{-1}(R(A))$. Hence, $A \cap Dom(R) \subseteq R^{-1}(R(A)) \subseteq R_\perp^{-1}(R(A))$.

$(3 \Longrightarrow 4)$: trivial.

$(4 \Longrightarrow 2)$: Assume that the fourth clause is true. We show that[2] the maximum flow of the network $\mathcal{N}(\mu_1, \mu_2, R)$ has value 1. To construct such a maximum flow, we borrow the proof idea of Theorem 7.3.4 from Desharnais [47]. According to the *Maximum Flow Minimum Cut Theorem* [2] (cf. Chapter 3.5), the maximum flow equals the capacity of a minimal cut. Therefore, it suffices to show that there exists a minimal cut of capacity 1. Cut $\{\nearrow\}$ has capacity 1, but we still have to show that it is minimal. Let C be some minimal cut (not necessarily $\{\nearrow\}$). We let $B = C \cap S_\perp$. The capacity of C is the sum:

$$c(C) = \sum \{c(i, j) \mid i \in C, j \notin C\}.$$

Recall we use $R_\perp$ to denote the set $R \cup \{(\perp, s) \mid s \in S_\perp\}$. Cut C has to fulfill $s \in B \implies \overline{R_\perp(s)} \subseteq C$ because otherwise it would have infinite capacity. Hence the capacity of C is:

$$c(C) = \mu_1(S_\perp \setminus B) + \mu_2(R_\perp(B)).$$

First assume that $\perp \in B$. In this case we have that $R_\perp(B) \supseteq R_\perp(\perp) = S_\perp$. Thus, $c(C) \geq \mu_2(R_\perp(B)) \geq \mu_2(S_\perp) = 1$. We consider now the case $\perp \notin B$. In this case we have $R_\perp(B) = R(B)$. Thus, $\mu_2(R_\perp(B)) = \mu_2(R(B))$. By construction of the network

[2]This part is developed with Björn Wachter together.

$\mathcal{N}$, it holds that $B \subseteq \mathit{Supp}(\mu_1)$. Since $\mu_1(B) \leq \mu_2(R(B))$, we have:

$$c(C) \geq \mu_1(S_\perp \setminus B) + \mu_1(B) = \mu_1(S_\perp) = 1$$

Hence, the value of the cut C is greater than or equal than 1, implying that the minimum cut has value 1. ■

The second clause will be used later in an efficient decision procedure for checking whether a weight function as the first clause exists. Observe that in the third clause, the set A does not contain the auxiliary state $\perp$. The reason is that $\perp$ represents underspecified behaviour, which do not need to be simulated. The fourth clause allows us to restrict to subsets of the support of μ_1.

With the above lemma, we provide simple proofs for Lemma 4.1.4 and Lemma 4.1.5 from Subsection 4.1.1.

Lemma 4.2.2 (The same as **Lemma 4.1.4**). *Let $R \subseteq S \times S$ be an equivalence relation. Let $\mu, \mu' \in \mathit{Dist}(S)$.*

1. *$\mu \equiv_R \mu'$ implies that $\mu \sqsubseteq_R \mu'$ and $\mu(S) = \mu'(S)$.*

2. *If $\mu(S) = \mu'(S)$ holds additionally, then $\mu \equiv_R \mu'$ iff $\mu \sqsubseteq_R \mu'$.*

Proof. 1. By definition, $\mu \equiv_R \mu'$ implies that $\mu(S) = \mu'(S)$. Let $A \subseteq S$. Since R is an equivalence relation, it holds that $A \subseteq R(A)$, and that $R(A)$ is a union of several equivalence classes. Thus it holds that $\mu(R(A)) = \mu'(R(A))$, which implies that $\mu(A) \leq \mu(R(A)) = \mu'(R(A))$. Applying Lemma 4.2.1, we get that $\mu \sqsubseteq_R \mu'$.

2. Because of the previous part, it remains to show that $\mu \sqsubseteq_R \mu'$ implies that $\mu \equiv_R \mu'$. Let A be an equivalence relation of R. Then, it holds that $R(A) = A$. Applying Lemma 4.2.1, we have that $\mu(A) \leq \mu'(R(A)) = \mu'(A)$. Since R is symmetric, by Lemma 4.1.3, it holds that $\mu' \sqsubseteq_R \mu$, which again implies that $\mu'(A) \leq \mu(R(A)) = \mu(A)$. Thus $\mu(A) = \mu'(A)$.

■

Lemma 4.2.3 (The same as **Lemma 4.1.5**). *Let R be a preorder on a set S and $\mu, \mu' \in \mathit{Dist}(S)$. If $\mu \sqsubseteq_R \mu'$ and $\mu' \sqsubseteq_R \mu$ then $\mu(A) = \mu'(A)$ for all equivalence classes A with respect to the kernel $R \cap R^{-1}$ of R.*

Proof. Let $A_1, \ldots, A_n$ be an enumeration of the equivalence classes of the kernel $R \cap R^{-1}$. Without of loss of generality, assume that $A_1, \ldots, A_n$ are an arbitrary *reverse* topological sort with respect to R. Then, for all $(s, s') \in R$ with $s \in A_i$, $s' \in A_j$, it holds that $i \leq j$. We define a sequence of sets $Q_1, \ldots, Q_n$ as follows: $Q_i = \cup_{j=1}^{i} A_j$. By construction, it holds that $R(Q_i) = Q_i$ for all i, thus:

1. $\mu \sqsubseteq_R \mu'$ implies that $\mu(Q_i) \leq \mu'(R(Q_i)) = \mu'(Q_i)$,

2. $\mu' \sqsubseteq_R \mu$ implies that $\mu'(Q_i) \leq \mu(R(Q_i)) = \mu(Q_i)$,

implying that $\mu(Q_i) = \mu'(Q_i)$ for all $i = 1, \ldots, n$. Or equivalently, it holds that $\mu(A_1) + \ldots + \mu(A_i) = \mu'(A_1) + \ldots + \mu'(A_i)$ for all $i = 1, \ldots, n$. It is then easy to see that $\mu(A_i) = \mu'(A_i)$ for all $i = 1, \ldots, n$. ■

Since the notion of simulation relations for models we consider involves weight functions, we can give equivalent definition of the various definitions using either condition 2 or 3 instead of weight functions. In the subsequent subsections, we give a detailed discussion of using the third condition, which provides another view of the simulation relations.

4.2.2 Strong (Probabilistic) Simulations

Let $\mathcal{M} = (S, \mathbf{P}, L)$ be an FPS. Our new definition provides, a rather intuitive, view of strong simulation. Assume that state s_2 strongly simulates s_1. By definition of strong simulation, $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$ where R is a strong simulation relation. From the first and the third clause of Lemma 4.2.1, we have that $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$ is equivalent to $\mathbf{P}(s_1, A) \leq \mathbf{P}(s_2, R(A))$ for all $A \subseteq S$. This allows us to give an equivalent definition for strong simulations:

Lemma 4.2.4. *Let $\mathcal{M} = (S, \mathbf{P}, L)$ be an FPS. The relation $R \subseteq S \times S$ is a strong simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and $\mathbf{P}(s_1, A) \leq \mathbf{P}(s_2, R(A))$ for all $A \subseteq S$.*

Proof. Assume that R is a strong simulation and let $s_1\ R\ s_2$. By definition of strong simulation, it holds that $L(s_1) = L(s_2)$. Moreover, $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$. By Lemma 4.2.1 it holds that $\mathbf{P}(s_1, A) \leq \mathbf{P}(s_2, R(A))$ for all $A \subseteq S$. Now let $R \subseteq S \times S$ such that for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and $\mathbf{P}(s_1, A) \leq \mathbf{P}(s_2, R(A))$ for all $A \subseteq S$. Again Lemma 4.2.1 implies that $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$, thus R is a strong simulation. ■

By the forth clause of Lemma 4.2.1, we can restrict to those A with $A \subseteq post(s_1)$:

Lemma 4.2.5. *Let $\mathcal{M} = (S, \mathbf{P}, L)$ be an FPS. The relation $R \subseteq S \times S$ is a strong simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and $\mathbf{P}(s_1, A) \leq \mathbf{P}(s_2, R(A))$ for all $A \subseteq post(s_1)$.*

Let R be a strong simulation, and let $s_1\ R\ s_2$. Taking $A = S$, we have that $\mathbf{P}(s_1, S) \leq \mathbf{P}(s_2, R(S)) \leq \mathbf{P}(s_2, S)$. Assume that state s_1 is absorbing and $L(s_1) = L(s_2)$. It is easy to show that $s_1 \precsim s_2$: taking the relation $R = \{(s_1, s_2)\}$. The set A must be the empty set, which implies that $R(A)$ is also an empty set. Thus $\mathbf{P}(s_1, A) = 0 = \mathbf{P}(s_2, R(A))$. Hence R is a strong simulation implying that $s_1 \precsim s_2$.

Example 4.2.1. We show that in the FPS in Figure 4.9, which is the same as the one in Figure 4.2 in Example 4.1.2. As in that example, we show that it holds that $s_1 \precsim s_2$ but $s_2 \not\precsim s_3$. First, consider the pair (s_1, s_2). Let $R = \{(s_1, s_2), (u_1, u_2), (v_1, v_2), (q_1, q_2)\}$. For all pairs $(s, s') \in R$ it holds that $L(s) = L(s')$. Since states u_1, q_1 are absorbing, the condition in Definition 4.2.5 is satisfied with respect to pairs (u_1, u_2) and (q_1, q_2) immediately. For pair (v_1, v_2) it is also easy to verify: for the only meaningful

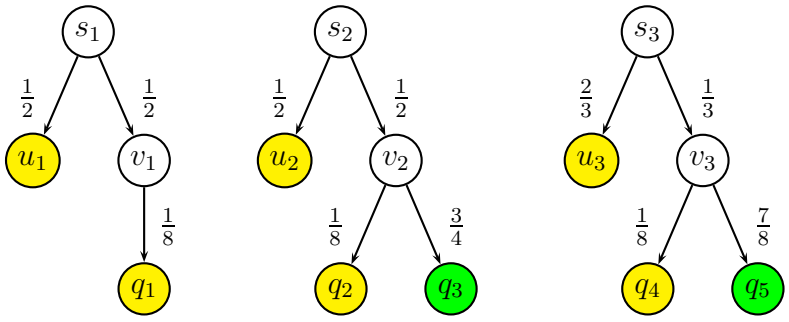

Figure 4.9: An FPS.

subset $A = \{q_1\}$ it holds that $\mathbf{P}(v_1, A) = \frac{1}{8} = \mathbf{P}(v_2, R(A)) = \mathbf{P}(v_2, \{q_2\})$. Consider the pair (s_1, s_2). The candidate sets A of interest could be: $\{u_1\}, \{v_1\}$ or $\{u_1, v_1\}$. For each of them, it is easy to verify that $\mathbf{P}(s_1, A) \leq \mathbf{P}(s_2, R(A))$. Now we consider the pair (s_2, s_3). Assume that we have a strong simulation R with $(s_2, s_3) \in R$. Taking $A = \{v_2\}$, we get that $\mathbf{P}(s_1, A) = \frac{1}{2}$. However, the only successor state of s_3 which can simulate s_2 is v_3 which implies that $R(A) \subseteq \{v_3\}$. However, we have $\mathbf{P}(s_2, A) > \mathbf{P}(s_3, v_3) \geq \mathbf{P}(s_3, R(A)) = \frac{1}{3}$. Thus, $s_2 \not\precsim s_3$.

Since DTMCs are special FPSs, Lemma 4.2.4 and 4.2.5 apply also for DTMCs. Moreover, since all states in DTMCs are either stochastic or absorbing, we show another property of strong simulation:

Lemma 4.2.6. *Let $\mathcal{M} = (S, \mathbf{P}, L)$ be a DTMC, and let $R \subseteq S \times S$ be a strong simulation on $\mathcal{M}$. Then, for all s_1, s_2 with $s_1\ R\ s_2$: $\mathbf{P}(s_1, A) = \mathbf{P}(s_2, R(A))$ for all $A \subseteq S$ with $A = R^{-1}(R(A))$.*

Proof. Let R be the strong simulation as specified, and let $s_1\ R\ s_2$. In case that s_1 is absorbing, A must be the empty set, which implies that $R(A)$ is also an empty set. Thus $\mathbf{P}(s_1, A) = 0 = \mathbf{P}(s_2, R(A))$. If s_1 is not absorbing, let $A \subseteq S$ with $A = R^{-1}(R(A))$. Since $s_1\ R\ s_2$, from Lemma 4.2.4 we have that $\mathbf{P}(s_1, A) \leq \mathbf{P}(s_2, R(A))$. Assume that $\mathbf{P}(s_1, A) < \mathbf{P}(s_2, R(A))$ for the sake of contradiction. Let $B = post(s_1) \setminus A$ and $B' = post(s_2) \setminus R(A)$. Since s_1 and s_2 are stochastic, we have that $\mathbf{P}(s_1, B) = 1 - \mathbf{P}(s_1, A)$ and $\mathbf{P}(s_2, B') = 1 - \mathbf{P}(s_2, R(A))$. Then, $\mathbf{P}(s_1, A) < \mathbf{P}(s_2, R(A))$ implies that $\mathbf{P}(s_1, B) > \mathbf{P}(s_2, B')$. On the other hand, $s_1\ R\ s_2$ implies that $\mathbf{P}(s_1, B) \leq \mathbf{P}(s_2, R(B))$, which implies that $\mathbf{P}(s_2, B') < \mathbf{P}(s_2, R(B))$. As a contradiction we show that $R(B) \subseteq B'$. Let $s \in R(B)$. By definition, we have that there exists a state $t \in B$ with $t\ R\ s$. $t \notin A$ implies that $t \notin R^{-1}(R(A))$. Thus, $s \notin R(A)$ which implies that $s \in B'$, which is the contradiction. ■

The following example shows that the converse does not hold.

Example 4.2.2. Consider the DTMC depicted in Figure 4.10. Obviously, it holds that $u_1 \precsim u_3$, $u_2 \precsim u_3$, $u_2 \precsim u_4$ and $u_1 \not\precsim u_4$. Now consider the states s_1, s_2 and the relation $R = \{(s_1, s_2), (u_1, u_3), (u_2, u_3), (u_2, u_4)\}$. We show that for all $A \subseteq S$ with $A = R^{-1}(R(A))$ it holds that $\mathbf{P}(s_1, A) = \mathbf{P}(s_2, R(A))$. If the candidate set A does not contain

either u_1 or u_2, we have $\mathbf{P}(s_1, A) = 0 = \mathbf{P}(s_2, R(A))$. The set $A = \{u_1\}$ or $A = \{u_2\}$ does not satisfy the property $A = R^{-1}(R(A))$. The set $A = \{u_1, u_2\}$ satisfies this property, but the condition holds trivially. However, $s_1 \not\precsim s_2$ since $\mathbf{P}(s_1, u_1) = 0.5$, and the only successor state which can simulate u_1 is u_3, thus $\mathbf{P}(s_2, R(u_1)) \leq \mathbf{P}(s_2, \{u_3\}) = 0.1$.

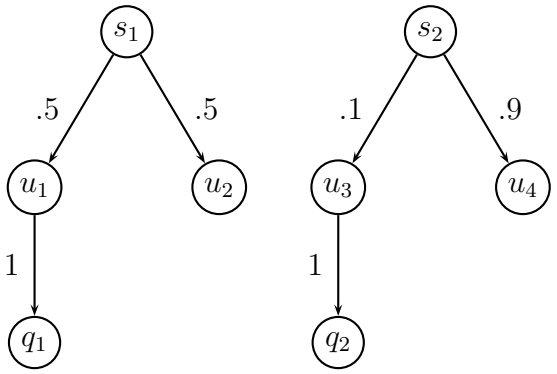

Figure 4.10: A DTMC.

Now we consider CTMCs. Similar to FPSs, by applying Lemma 4.2.1, we give an equivalent definition for strong simulation for CTMCs. The proof is similar to the proof of Lemma 4.2.4.

Lemma 4.2.7. *Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC. The relation $R \subseteq S \times S$ is a strong simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and $\mathbf{P}(s_1, A) \leq \mathbf{P}(s_2, R(A))$ for all $A \subseteq S$, and $\mathbf{R}(s_1, S) \leq \mathbf{R}(s_2, S)$.*

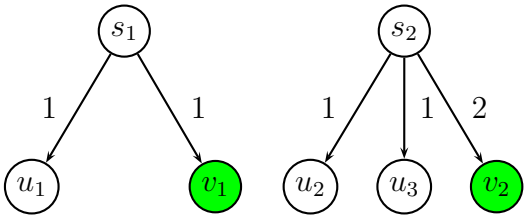

Figure 4.11: A CTMC.

Example 4.2.3. Consider the CTMC in Figure 4.11. In the embedded DTMC, for all $A \subseteq S$ it holds that $\mathbf{P}(s_1, A) \leq \mathbf{P}(s_2, R(A))$. Moreover, $\mathbf{R}(s_1, S) = 2 < \mathbf{R}(s_2, S) = 4$ which implies that $s_1 \precsim s_2$.

Lemma 4.2.8. *Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC, and let $R \subseteq S \times S$ be a strong simulation on $\mathcal{M}$. Then, for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and $\frac{\mathbf{R}(s_1, A)}{\mathbf{R}(s_2, R(A))} = \frac{\mathbf{R}(s_1, S)}{\mathbf{R}(s_2, S)}$ for all $A \subseteq S$ with $A = R^{-1}(R(A))$.*

Proof. Assume that R is a strong simulation and $s_1\ R\ s_2$. By Lemma 4.2.6, in the embedded DTMC it holds that $\mathbf{P}(s_1, A) = \mathbf{P}(s_2, R(A))$ for all $A \subseteq S$ with $A = R^{-1}(R(A))$. It holds that $\mathbf{P}(s_1, A) = \frac{\mathbf{R}(s_1,A)}{\mathbf{R}(s_1,S)}$. Similarly, $\mathbf{P}(s_2, R(A)) = \frac{\mathbf{R}(s_2,R(A))}{\mathbf{R}(s_2,S)}$. Thus $\frac{\mathbf{R}(s_1,A)}{\mathbf{R}(s_1,S)} = \frac{\mathbf{R}(s_2,R(A))}{\mathbf{R}(s_2,S)}$ which proves the lemma. ■

For PAs and CPAs, strong and strong probabilistic simulation relations also involve weight function conditions. In the following two lemmas, we give equivalent definition of strong and strong probabilistic simulation:

Lemma 4.2.9. *Let $\mathcal{M} = (S, Act, \mathbf{P}, L)$ be a PA. The relation $R \subseteq S \times S$ is a strong (strong probabilistic) simulation on $\mathcal{M}$ iff for all $s_1, s_2 \in S$ with $s_1\ R\ s_2$, $L(s_1) = L(s_2)$ and for all $s_1 \xrightarrow{\alpha} \mu_1$, there exists a transition $s_2 \xrightarrow{\alpha} \mu_2$ (a combined transition $s_2 \overset{\alpha}{\rightsquigarrow} \mu_2$) such that $\mu_1(A) \leq \mu_2(R(A))$ for all $A \subseteq post(s_1)$.*

Lemma 4.2.10. *Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA. The relation $R \subseteq S \times S$ is a strong (strong probabilistic) simulation on $\mathcal{M}$ iff for all $s_1, s_2 \in S$ with $s_1\ R\ s_2$, $L(s_1) = L(s_2)$ and for all $s_1 \xrightarrow{\alpha} r_1$, there exists a transition $s_2 \xrightarrow{\alpha} r_2$ (a combined transition $s_2 \overset{\alpha}{\rightsquigarrow} r_2$) such that $r_1(S) \leq r_2(S)$ and $\mu(r_1)(A) \leq \mu(r_2)(R(A))$ for all $A \subseteq post(s_1)$.*

4.3 Weak Simulations

This section is organised as follows. Weak simulation for DTMCs will be repeated in Subsection 4.3.1. The definition is extended to CTMCs in Subsection 4.3.2. For FPSs there are some intricacies in the definition of weak simulation, which will be discussed in Subsection 4.3.3.

4.3.1 Weak Simulation for DTMCs

A state s_2 weakly simulates another state s_1 if they have the same labelling, and if their successor states can be grouped into sets U_i and V_i for $i = 1, 2$, satisfying certain conditions. Consider Figure 4.12. We can view steps to V_i as *stutter* steps while to U_i are *visible* steps. With respect to the visible steps, it is then required that there exists a weight function for the conditional distributions: $\frac{\mathbf{P}(s_1,\cdot)}{K_1}$ and $\frac{\mathbf{P}(s_2,\cdot)}{K_2}$ where K_i intuitively is the probability to perform a visible step from s_i. The stutter steps must respect the weak simulation relations: thus states in V_2 should weakly simulate s_1, and state s_2 should weakly simulate states in V_1. This is depicted by dashed arrows in the figure. For reasons we will explain later in Example 4.3.6, the definition needs to account for states which partially belong to U_i and partially to V_i. Technically, this is achieved by functions δ_i that distribute s_i over U_i and V_i in the definition below. For a given pair (s_1, s_2) and functions $\delta_i : S \to [0, 1]$, let $U_{\delta_i}, V_{\delta_i} \subseteq S$ (for $i = 1, 2$) denote the sets

$$U_{\delta_i} = \{u \in post(s_i) \mid \delta_i(u) > 0\},\ V_{\delta_i} = \{v \in post(s_i) \mid \delta_i(v) < 1\} \tag{4.4}$$

If (s_1, s_2) and δ_i are clear from the context, we write U_i, V_i instead.

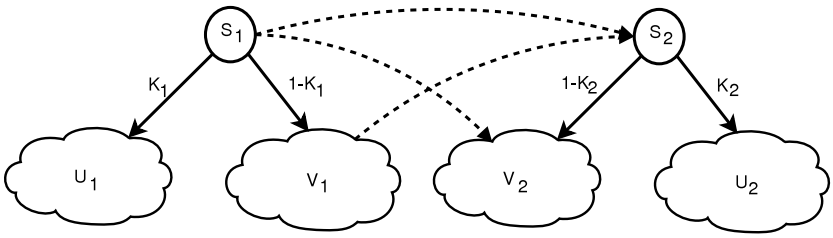

Figure 4.12: Illustrating the splitting of successor states in weak simulations for FPSs.

Definition 4.3.1 ([18]). *Let $\mathcal{M} = (S, \mathbf{P}, L)$ be a DTMC and $R \subseteq S \times S$. The relation R is a weak simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and there exist functions $\delta_i : S \to [0,1]$ such that:*

1. *(a) $v_1\ R\ s_2$ for all $v_1 \in V_1$, and (b) $s_1\ R\ v_2$ for all $v_2 \in V_2$*
2. *there exists a function $\Delta : S \times S \to [0,1]$ such that:*
   - *(a) $\Delta(u_1, u_2) > 0$ implies $u_1 \in U_1, u_2 \in U_2$ and $u_1\ R\ u_2$,*
   - *(b) if $K_1 > 0$ and $K_2 > 0$ then for all states $w \in S$:*

$$K_1 \cdot \Delta(w, U_2) = \mathbf{P}(s_1, w)\delta_1(w),\ \ K_2 \cdot \Delta(U_1, w) = \mathbf{P}(s_2, w)\delta_2(w)$$

   where $K_i = \sum_{u_i \in U_i} \delta_i(u_i) \cdot \mathbf{P}(s_i, u_i)$ for $i = 1, 2$.
3. *for $u_1 \in U_1$ there exists a path fragment $s_2, w_1, \ldots, w_n, u_2$ with positive probability such that $n \geq 0$, $s_1\ R\ w_j$ for $0 < j \leq n$, and $u_1\ R\ u_2$.*

We say that s_2 weakly simulates s_1 in $\mathcal{M}$, denoted $s_1 \precsim_{\mathcal{M}} s_2$, iff there exists a weak simulation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.

Note that the sets U_i, V_i in the above definition are defined according to Equation 4.4 with respect to the pair (s_1, s_2) and the functions δ_i. Condition 3 will in the sequel be called the *reachability condition*. If $K_1 > 0$ and $K_2 = 0$, which implies that $U_2 = \emptyset$ and $U_1 \neq \emptyset$, the reachability condition guarantees that for any visible step $s_1 \to u_1$ with $u_1 \in U_1$, s_2 can reach a state u_2 which simulates u_1 while passing only through states simulating s_1.

Example 4.3.1. Consider the DTMC in Figure 4.13. We first show that $s_i \precsim t_j$ for $i, j \in \{1, 2, 3\}$. For each of these pair we can select $U_1 = \emptyset$ (i.e. $\delta_1(s) = 0$ for all $s \in S$) and $V_2 = \emptyset$ (i.e. $\delta_2(s) = 1$ for all $s \in S$). Since $K_1 = 0$, only the Condition 1 need to checked. Since all successor states of s_i are either itself or empty, the condition holds trivially. Similarly, it holds also $v_1 \precsim v_2$.

Since v_2 has a transition to s_3 which cannot be simulated by v_1, we have that $v_2 \not\precsim v_1$. Similarly, it holds that $t_1 \precsim t_3$ but $t_3 \not\precsim t_1$.

We show that $t_1 \precsim t_2$: to establish this relation we let $U_1 = \{v_1\}$, $V_1 = \emptyset$, $U_2 = \emptyset$ and $V_2 = \{t_3\}$. Thus $K_1 = 1$ and $K_2 = 0$. Condition 1 of Definition 4.3.1 holds since $t_1 \precsim t_3$.

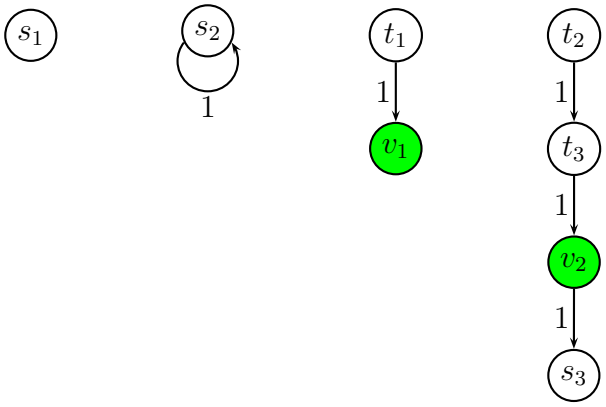

Figure 4.13: A simple DTMC for illustrating the weak simulation relations.

The second condition holds trivially with $\Delta(s, s') = 0$ for all $s, s' \in S$. To show the reachability condition, observe that t_2 can reach a state v_2 which can weakly simulate v_1.

Dropping Condition 3 would mean, however, that $t_i \precsim s_j$ for $i, j \in \{1, 2, 3\}$ by taking simply $V_1 = \emptyset$ and $U_2 = \emptyset$.

The functions δ_i can be considered as a generalisation of the characteristic function of U_i in the sense that we may *split* the membership of a state to U_i and V_i into *fragments* which sum up to 1. Assume that $s_1 \precsim s_2$. By definition, if the successor state s_1' of s_1 cannot be weakly simulated by s_2, it holds that $\delta_1(s_1') = 1$. Similarly, if the successor state s_2' cannot weakly simulates s_1, it holds that $\delta_2(s_2') = 1$. If $\delta_1(s) = \frac{1}{3}$, we say that $\frac{1}{3}$ fragment of the state s belongs to U_1, and $\frac{2}{3}$ fragment of s belongs to V_1. Hence, U_i and V_i are not necessarily disjoint. Observe that $U_i = \emptyset$ implies that $\delta_i(s) = 0$ for all $s \in S$. Similarly, $V_i = \emptyset$ implies that $\delta_i(s) = 1$ for all $s \in S$. We show that the use of fragments of states is necessary to establish weak simulation relations.

Example 4.3.2. Consider the DTMC depicted in Figure 4.14. For states u_1, u_2, v_1, v_2, obviously the following pairs $(u_1, u_2), (u_1, v_2), (v_1, v_2)$ are in the weak simulation relation. The state u_2 cannot weakly simulate v_1. Since v_2 weakly simulates v_1, it holds that $s_2 \precsim s_5$. Similarly, from $u_1 \precsim v_1$ we can easily show that $s_1 \precsim s_4$. We observe also that $s_2 \not\precsim s_3$: $K_1 > 0$ and $K_2 > 0$ since both s_2 and s_3 have yellow (grey) successor states, but the required function Δ cannot be established since u_2 cannot weakly simulate any successor state of s_2 (which is v_1). Thus $s_2 \not\precsim s_3$.

Without considering fragments of states, we show that a weak simulation between s_1 and s_3 cannot be established. Since $s_2 \not\precsim s_3$, we must have $U_1 = \{u_1, v_1, s_2\}$ and $V_1 = \emptyset$. The function δ_1 is thus defined by $\delta_1(u_1) = \delta_1(v_1) = \delta(s_2) = 1$ which implies that $K_1 = 1$. Now consider the successor states of s_3. Obviously $\delta_2(u_2) = \delta_2(v_2) = 1$, which implies that $u_2, v_2 \in U_2$. We consider the following two cases:

- The case $\delta_2(s_4) = 1$. In this case we have that $K_2 = 1$. A function Δ must be defined satisfying Condition 2b in Definition 4.3.1. Taking $w = s_4$, the following must hold: $K_2 \cdot \Delta(U_1, s_4) = \mathbf{P}(s_3, s_4)\delta_2(s_4)$. As $K_2 = 1, \mathbf{P}(s_3, s_4) = 0.75$ and

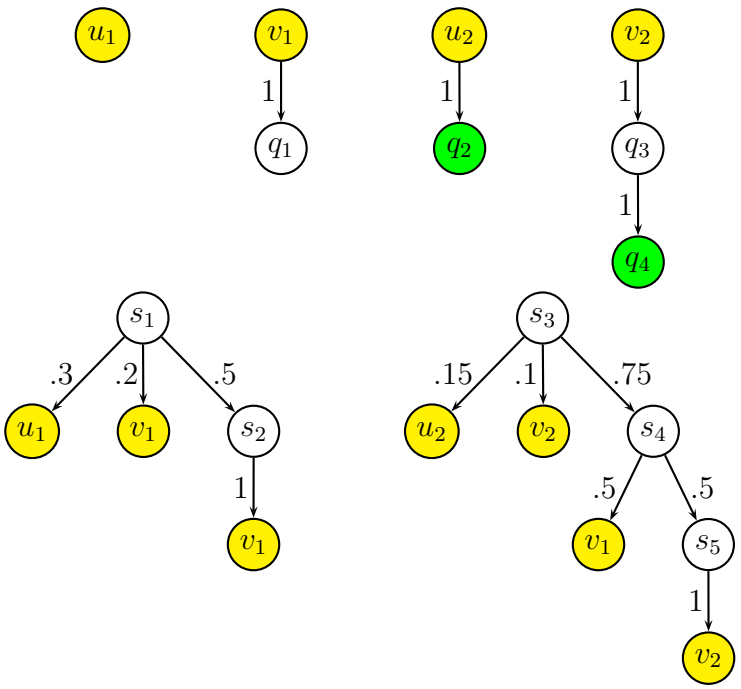

Figure 4.14: A DTMC where the splitting of states is necessary to establish the weak simulation. In the model some states are drawn more than once.

$\delta_2(s_4) = 1$, it follows that $\Delta(U_1, s_4) = 0.75$. The state s_2 is the only successor of s_1 that can be weakly simulated by s_4, so $\Delta(s_2, s_4) = 0.75$ must hold. However, the equation $K_1 \cdot \Delta(s_2, U_2) = \mathbf{P}(s_1, s_2)\delta_1(s_2)$ does not hold any more, as on the left side we have 0.75 but on the right side we have 0.5 instead.

- The case $\delta_2(s_4) = 0$. In this case we have still $K_2 > 0$. Similar to the previous case it is easy to see that the required function Δ cannot be defined: the equation $K_1 \cdot \Delta(s_2, U_2) = \mathbf{P}(s_1, s_2)\delta_1(s_2)$ does not hold since the left side is 0 (no states in U_2 can weakly simulate s_2) but the right side equals 0.5.

Thus without splitting, s_3 does not weakly simulate s_1. We show it holds that $s_1 \precsim s_3$. It is sufficient to show that the relation $R = \{(s_1, s_3), (u_1, u_2), (v_1, v_2), (q_1, q_3), (s_1, s_4), (u_1, v_1), (v_1, v_1), (q_1, q_1), (s_2, s_5), (s_2, s_4)\}$ is a weak simulation relation. By the discussions above, it is easy to verify that every pair except (s_1, s_3) satisfies the conditions in Definition 4.3.1. We show now that the conditions hold also for the pair (s_1, s_3). The function δ_1 with $\delta_1(u_1) = \delta_1(v_1) = \delta_1(s_2) = 1$ is defined as above, also the sets $U_1 = \{u_1, v_1, s_2\}$, $V_1 = \emptyset$. The function δ_2 is defined by: $\delta_2(u_2) = \delta_2(v_2) = 1$ and $\delta_2(s_4) = \frac{1}{3}$, which implies that $U_2 = \{u_2, v_2, s_4\}$ and $V_2 = \{s_4\}$. Thus, we have $K_1 = 1$ and $K_2 = 0.5$. Since $s_1 \precsim s_4$, Condition 1 holds trivially as $(s_1, s_4) \in R$. The reachability condition also holds trivially. To show that Condition 2 holds, we define the function Δ as follows: $\Delta(u_1, u_2) = 0.3$, $\Delta(v_1, v_2) = 0.2$ and $\Delta(s_2, s_4) = 0.5$. We show that $K_2 \cdot \Delta(U_1, w) = \mathbf{P}(s_3, w)\delta_2(w)$ holds for all $w \in S$. It holds that $K_2 = 0.5$. First observe that for $w \notin U_2$ both sides of the equation equal 0. Let first $w = u_2$ for which we have that $\mathbf{P}(s_3, u_2)\delta_2(u_2) = 0.15$. Since $\Delta(U_1, u_2) = 0.3$, also the left

side equals 0.15. The case $w = v_2$ can be shown in a similar way. Now consider $w = s_4$. Observe that $\Delta(U_1, s_4) = 0.5$ thus the left side equals 0.25. The right side equals $\mathbf{P}(s_3, s_4)\delta_2(s_4) = 0.75 \cdot \frac{1}{3} = 0.25$ thus the equation holds. The equation $K_1 \cdot \Delta(w, U_2) = \mathbf{P}(s_1, w)\delta_1(w)$ can be shown in a similar way. Thus Δ satisfies all the conditions which implies that $s_1 \precsim s_3$.

4.3.2 Weak Simulation for CTMCs

The idea of using fragments of states to establish weak simulation for DTMCs can be extended to CTMCs:

Definition 4.3.2 ([18, 17])**.** *Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC and $R \subseteq S \times S$. The relation R is a weak simulation on $\mathcal{M}$ iff for $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and there exist functions $\delta_i : S \to [0, 1]$ (for $i = 1, 2$) satisfying Equation 4.4 and Conditions 1 and 2 of Definition 4.3.1 and the* rate condition*:*

$$(3') \qquad K_1 \cdot \mathbf{R}(s_1, S) \leq K_2 \cdot \mathbf{R}(s_2, S)$$

We say that s_2 weakly simulates s_1 in $\mathcal{M}$, denoted $s_1 \precsim_{\mathcal{M}} s_2$, iff there exists a weak simulation R on $\mathcal{M}$ such that $s_1\ R\ s_2$.

In this definition, the rate condition (3′) strengthens the reachability condition of the preceding definition. If $U_1 \neq \emptyset$, we have that $K_1 > 0$; the rate condition then requires that $K_2 > 0$, which implies $U_2 \neq \emptyset$.

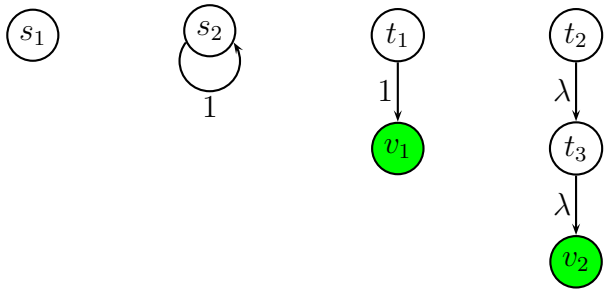

Figure 4.15: A simple CTMC for illustrating the weak simulation relations.

Example 4.3.3. Consider the CTMC in Figure 4.15 (cf. the DTMC in Figure 4.13 in Example 4.3.1). Assume that $\lambda > 1$. By similar arguments as Example 4.3.1, we have that $s_i \precsim t_j$ for $i = 1, 2$ and $j = 1, 2, 3$.

It holds obviously $t_1 \precsim t_3$: we take $U_1 = \{v_1\}$,$V_1 = \emptyset$, $U_2 = \{v_2\}$ and $V_2 = \emptyset$ to fulfill the conditions in Definition 4.3.2. Because of the rate condition, $t_3 \not\precsim t_1$: $K_1\mathbf{R}(t_3, v_2) = \lambda$ which is greater than $K_2\mathbf{R}(t_1, v_1) = 1$.

Observe also, because of the rate condition, $t_1 \not\precsim t_2$: $K_1 > 0$ implies $K_2 > 0$ which again implies that $t_3 \in U_2$. However, t_3 cannot weakly simulates v_1.

As for DTMCs, we show that the use of fragments of states is necessary to establish weak simulation relations. Firstly, if we interpret the DTMC in Figure 4.14

as a CTMC, the relation $s_1 \precsim s_3$ cannot be established (cf. Example 4.3.2): with $\mathbf{R}(s_1, S) = \mathbf{R}(s_2, S) = 1$, $K_1 = 1$ and $K_2 = 0.5$ the rate condition is violated .

Example 4.3.4. Consider the CTMC depicted in Figure 4.16. The embedded DTMC of it is exactly the DTMC in Figure 4.14. Obviously, the rate conditions holds in the CTMC for all weak simulation relations we discussed in Example 4.3.2. Thus, we can (only with splitting) establish that $s_1 \precsim s_3$.

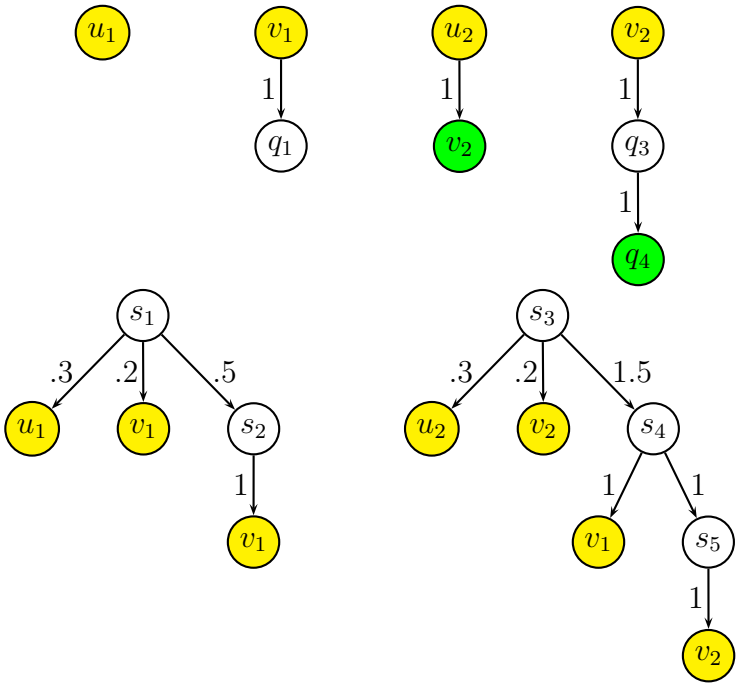

Figure 4.16: A CTMC where the splitting of states is necessary to establish the weak simulation.

4.3.3 Weak Simulation for FPSs

In [18], the notion of weak simulation is also proposed for FPSs. However, we show in this section that the definition contains a subtle flaw, and cannot be fixed in an obvious way. First we repeat the proposed definition. In this section, for a given pair (s_1, s_2) and functions functions $\delta_i : S_\perp \to [0, 1]$, let $U_{\delta_i}, V_{\delta_i} \subseteq S_\perp$ (for $i = 1, 2$) denote the sets

$$U_{\delta_i} = \{u \in post_\perp(s_i) \mid \delta_i(u) > 0\},\ V_{\delta_i} = \{v \in post_\perp(s_i) \mid \delta_i(v) < 1\} \tag{4.5}$$

If (s_1, s_2) and δ_i are clear from the context, we write U_i, V_i instead.

Definition 4.3.3. *Let $\mathcal{M} = (S, \mathbf{P}, L)$ be an FPS and $R \subseteq S \times S$. The relation R is a weak simulation on $\mathcal{M}$ iff for all s_1, s_2 with $s_1\ R\ s_2$: $L(s_1) = L(s_2)$ and there exist functions $\delta_i : S_\perp \to [0, 1]$ such that:*

1. *(a) $v_1\ R\ s_2$ for all $v_1 \in V_1 \backslash \{\perp\}$, and (b) $s_1\ R\ v_2$ for all $v_2 \in V_2 \backslash \{\perp\}$*

2. *there exists a function* $\Delta : S_\perp \times S_\perp \to [0,1]$ *such that:*

   (a) $\Delta(u_1, u_2) > 0$ *implies* $u_1 \in U_1, u_2 \in U_2$ *and* $u_1 \; R_\perp \; u_2$,

   (b) *if* $K_1 > 0$ *and* $K_2 > 0$ *then for all states* $w \in S_\perp$*:*

$$K_1 \cdot \Delta(w, U_2) = \mathbf{P}(s_1, w)\delta_1(w), \;\; K_2 \cdot \Delta(U_1, w) = \mathbf{P}(s_2, w)\delta_2(w)$$

   where $K_i = \sum_{u_i \in U_i} \delta_i(u_i) \cdot \mathbf{P}(s_i, u_i)$ *for* $i = 1, 2$.

3. *for* $u_1 \in U_1 \backslash \{\perp\}$ *there exists a path fragment* $s_2, w_1, \ldots, w_n, u_2$ *with positive probability such that* $n \geq 0$, $s_1 \; R \; w_j$ *for* $0 < j \leq n$, *and* $u_1 \; R \; u_2$.

We say that s_2 *weakly simulates* s_1 *in* $\mathcal{M}$, *denoted* $s_1 \precsim_{\mathcal{M}} s_2$, *iff there exists a weak simulation* R *on* $\mathcal{M}$ *such that* $s_1 \; R \; s_2$.

The sets U_i, V_i in the above definition are defined according to Equation 4.5 with respect to the pair (s_1, s_2) and the functions δ_i. Below we first provide an example in which we can establish $s_1 \precsim s_2$ via the weak simulation for FPSs, but which violates the logical preservation results.

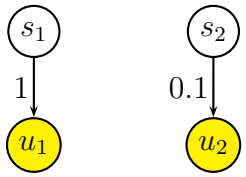

Figure 4.17: A simple FPS.

Example 4.3.5. Consider the simple FPS depicted in Figure 4.17. We first show that by definition of weak simulation it holds that $s_1 \precsim s_2$. We show that the relation $R = \{(s_1, s_2), (u_1, u_2)\}$ is a weak simulation relation. The condition for the pair (u_1, u_2) holds trivially. For (s_1, s_2) consider the functions δ_i defined by: $\delta_1(u_1) = \delta_2(u_2) = 1$. Then, the sets U_i and V_i are given by: $U_1 = \{u_1\}$, $V_1 = \emptyset$, $U_2 = \{u_2\}$, $V_2 = \{\perp\}$. The constants K_i can be easily computed: $K_1 = 1$, $K_2 = 0.1$. The required function Δ is defined as follows: $\Delta(s, s')$ equals 1 if $s = u_1$, $s' = u_2$, and equals 0 otherwise. Condition 1 and the reachability condition 3 of Definition 4.3.3 hold trivially. Consider Condition 2. Only the pair (u_1, u_2) has positive weight, thus Condition 2a holds. Consider Condition 2b: it holds that $K_2\Delta(U_1, u_2) = \mathbf{P}(s_2, u_2)\delta_2(u_2) = 0.1$, and $K_1\Delta(u_1, U_2) = \mathbf{P}(s_1, u_1)\delta_2(u_1) = 1$. For other state $s \in S_\perp$, the equation holds trivially: both sides of it equal 0.

In the above example, we have shown that according to Definition 4.3.3, we have that $s_1 \precsim s_2$. However, this violates the logical preservation results. Observe that state s_2 reaches yellow (or grey) states with probability 0.1, which is smaller than the probability of reaching yellow (or grey) states from s_1. This contradicts Theorem 63 of paper [18]. To understand this theorem, we shall recall briefly the liveness $\text{PCTL}_{\backslash\mathcal{X}}$ formulas and their semantics. Details can be found in [18]. The liveness $\text{PCTL}_{\backslash\mathcal{X}}$ formulas are defined by:

$$\Phi = a \mid \neg a \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathcal{P}_{\trianglerighteq p}(\Phi \; \mathcal{U} \; \Phi)$$

where $a \in AP$, and $\trianglerighteq \in \{>, \geq\}$. The semantics for the atomic proposition a, boolean operators negation, conjunction, disjunction is as defined for CTLs. A state s satisfies the probabilistic formula $\mathcal{P}_{>p}(\Phi_1 \,\mathcal{U}\, \Phi_2)$ (or $\mathcal{P}_{\geq p}(\Phi_1 \,\mathcal{U}\, \Phi_2)$) if starting from s the probability of the set of paths (a path is a sequence of states), which is denoted by $\mathcal{P}_s(\Phi_1 \,\mathcal{U}\, \Phi_2)$, is greater than (or greater than or equal to) p. We write *true* for the formula $a \vee \neg a$ for some $a \in AP$. If a state s satisfies a PCTL$_{\setminus \mathcal{X}}$ formula, we write $s \models \Phi$.

We define a relation $\precapprox^{live}$ by: $(s, s') \in \precapprox^{live}$ if for all PCTL$_{\setminus \mathcal{X}}$ liveness formula Φ it holds that $s \models \Phi$ implies that $s' \models \Phi$. In Theorem 63 of paper [18] it is shown that $\precapprox \subseteq \precapprox^{live}$, i.e., $s_1 \precapprox s_2$ implies that for all liveness PCTL$_{\setminus \mathcal{X}}$ formulas Φ, $s_1 \models \Phi$ implies that $s_2 \models \Phi$.

Now we go back to our example (consider the FPS in Figure 4.17). Consider the PCTL$_{\setminus \mathcal{X}}$ liveness formula: $\Phi := \mathcal{P}_{>0.5}(\mathit{true}\,\mathcal{U}\,\mathit{yellow})$ which states that the probability of reaching yellow (or grey) states is greater than 0.5. Obviously, the probability of reaching yellow (or grey) states from s_1 is 1, and from s_2 is 0.1, thus $s_1 \models \Phi$ but $s_2 \not\models \Phi$. This contradicts the afore-cited theorem.

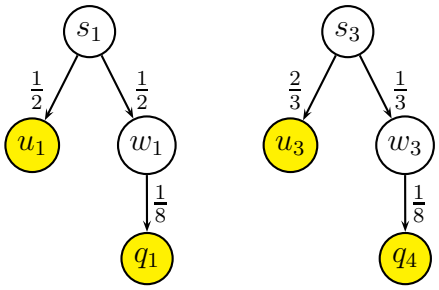

Figure 4.18: An example.

Example 4.3.6. Consider the FPS depicted in Figure 4.18. We show that $s_1 \precapprox s_3$. For this we first show that $w_1 \precapprox s_3$ and $s_1 \precapprox w_3$.

For $w_1 \precapprox s_3$, consider the relation $R_1 = \{(w_1, s_3), (q_1, u_3), (w_1, w_3), (q_1, q_4)\}$. For the pair (w_1, s_3), we define $\delta_1(q_1) = 1$ and 0 otherwise, $\delta_2(u_3) = 1$ and 0 otherwise. The function Δ is defined by $\Delta(q_1, u_3) = 1$ and 0 otherwise. Since $w_1 \precapprox w_3$, all conditions of Definition 4.3.3 are satisfied for (w_1, s_3). It is routine to verify that all conditions of Definition 4.3.3 are satisfied for other pairs. Thus R_1 is a weak simulation.

Similarly, we can show $s_1 \precapprox w_3$ by showing that the relation $R_2 = \{(s_1, w_3), (u_1, q_4), (w_1, w_3), (q_1, q_4)\}$ is a weak simulation. Consider the pair (s_1, w_3) (other pairs are simple to treat). The functions δ_i for (s_1, s_3) is defined by: $\delta_1(u_1) = 1$ and 0 otherwise, $\delta_2(q_4) = 1$ and 0 otherwise. The function Δ is defined by $\Delta(u_1, q_4) = 1$ and 0 otherwise. It is routine to verify that all conditions of Definition 4.3.3 are satisfied.

Now to show $s_1 \precapprox s_3$, we define $R = \{(s_1, s_3)\} \cup R_1 \cup R_2$. To show that R is a weak simulation, we need only to consider (s_1, s_3). We define $\delta_1(u_1) = 1$ and 0 otherwise, $\delta_2(u_3) = 1$ and 0 otherwise. The function Δ is defined by $\Delta(u_1, u_3) = 1$ and 0 otherwise. Again, it is routine to show that all conditions of Definition 4.3.3 are satisfied.

Again, $s_1 \precapprox w_3$ violates the afore-cited theorem, as the probability of reaching yellow (or grey) states from s_1 is higher than the corresponding probability from s_3. The above

example is the same as [18, Example 36]. However, in [18] their analysis flawed: they show that without splitting of states one cannot establish $s_1 \precsim s_3$. The core problem of their analysis is that they assumed that $s_1 \not\precsim w_3$ holds which is wrong with respect to Definition 4.3.3.

The analysis above shows that the definition of weak simulation for FPSs contains a subtle flaw. It seems that the problem would be that there is no condition on the probability of the under-specified behaviour of s_2: We are allowed to put $\perp$ completely into set V_2 (i.e. $\delta_2(\perp) = 0$), while the condition 1 does not require anything from $\perp$. Thus, as an ad-hoc fix we could change Condition 1b to: $s_1 \; R \; v_2$ for all $v_2 \in V_2$. Since $s \not\precsim \perp$ for all $s \in S$, this implies then $\delta_2(\perp) = 1$, i.e., $\perp$ will be completely in the set U_2. Now we consider again Example 4.3.5. With this definition, it is easy to see that $s_1 \not\precsim s_2$.

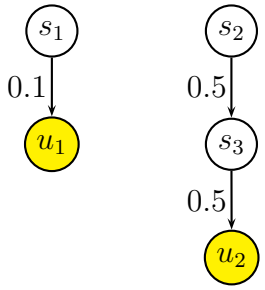

Figure 4.19: An FPS showing that our ad-hoc fix does not work.

Unfortunately, the above suggestion is also counterintuitive. Consider the FPS in Figure 4.19. Obviously, it holds that $s_1 \precsim s_3$. Now consider the state s_2. With the unmodified version of weak simulations, it is easy to see that $s_1 \precsim s_2$. In the above modified definition, however, we show that $s_1 \not\precsim s_2$. As $\mathbf{P}(s_2, \perp) > 0$, we must put $\perp$ completely into U_2, i.e. $\delta_2(\perp) = 1$. This implies that $K_2 > 0$, which means there must exists a function Δ satisfying Condition 2. However, since no successor state of s_2 can weakly simulate u_1, the condition $K_1 \Delta(u_1, U_2) = \mathbf{P}(s_1, u_1)\delta_1(u_1)$ cannot be established, implying $s_1 \not\precsim s_2$.

On the other hand, by exploiting structural induction, we can show that s_2 satisfies all PCTL$_{\setminus\mathcal{X}}$ liveness formulas satisfied by s_1. Assume that Φ is a PCTL$_{\setminus\mathcal{X}}$ liveness formula such that $s_1 \models \Phi$. We give the brief proof for $s_2 \models \Phi$ by structural induction:

- In case that $\Phi = a$, we have that $a \in L(s_1)$. Since s_1 and s_2 have the same labelling (as depicted in Figure 4.19), it holds also that $s_2 \models a$.
- The negation $\Phi = \neg a$ is shown as follows: $s_1 \models \Phi$ implies that $a \notin L(s_1) = L(s_2)$. Thus $s_2 \models \Phi$.
- Assume that $\Phi = \Phi_1 \wedge \Phi_2$. Then, $s_1 \models \Phi$ implies that $s_1 \models \Phi_1$ and $s_1 \models \Phi_2$. Since Φ_1 and Φ_2 are also liveness formulas, by structural induction it holds that $s_2 \models \Phi_1$ and $s_2 \models \Phi_2$ which implies that $s_2 \models \Phi$.
- The proof for disjunction follows similarly as for conjunction.

- Assume that $\Phi = \mathcal{P}_{>p}(\Phi_1 \ \mathcal{U} \ \Phi_2)$. State s_1 satisfies Φ implies that the formula Φ_2 must be satisfied by s_1 or u_1 (otherwise the probability of $\mathcal{P}_s(\Phi_1 \ \mathcal{U} \ \Phi_2)$ is 0). Then, we have that the probability $\mathcal{P}_s(\Phi_1 \ \mathcal{U} \ \Phi_2)$ is either 0.1 (in case that s_1 does not satisfy Φ_2) or 1 (in case that s_1 does satisfy Φ_2). In both case we have a higher probability from s_2: 0.25 in the former case and 1 in the later case. Thus s_2 also satisfies Φ. The case $\Phi = \mathcal{P}_{\geq p}(\Phi_1 \ \mathcal{U} \ \Phi_2)$ can be treated similarly.

However, this indicates that $\precsim^{live}_{\approx} \not\subseteq \precsim_{\approx}$ which is counterintuitive: it is usually desired that the relation induced by the liveness formulas coincides with the weak simulation relation.

4.4 Bibliographic Notes

Strong bisimulation has been introduced by Larsen and Skou [80] for reactive systems, i.e., MDPs. Intuitively, two states s and t are bisimilar if they have the same transition probability of going to an arbitrary equivalence class, with respect to the same action. Since PAs allow internal nondeterminism, another relaxed version of the relation, called strong probabilistic bisimulation, is introduced by Segala and Lynch [102]. The basic idea is the use of combined successor distributions, which is a relaxation of successor distributions. Strong bisimulation is then introduced for continuous-time models [65, 15, 18, 87]. In a nutshell, two states s and t are strong bisimilar if they have the same transition rate of going to an arbitrary equivalence class, with respect to the same action in case of nondeterminism. For most of these models, the notion of weak (probabilistic) bisimulation relations [103, 92, 12, 65] has also been considered which abstract from internal computations. We refer the reader to [103] for an extensive comparative study for discrete-time models [112, 92, 19, 4], especially PAs, MDPs and alternating models. For models without nondeterminism, i.e., FPSs, DTMCs and CTMCs, we refer to [18].

As for non-probabilistic systems, simulation can be considered as an uni-directional bisimulation: if t simulates s, then arbitrary successor state of s must be imitated by some successor state of t. Segala and Lynch [102] have studied strong, strong probabilistic, and weak probabilistic simulation in the context of PAs, in which the correspondence between successor distributions is established with the concept of weight functions. As we have already mentioned, weak simulation for Markov chains [18] is introduced in which states are split into visible and invisible parts, and only visible parts need to be matched.

Also very closely related is the work by Desharnais *et al.* [47, 48, 49], in which bisimulation and simulation relations are introduced for labelled Markov processes (LMPs), which can be considered as MDPs but with continuous state space. Notably, their definition for simulations, provides another (rather intuitive) view of simulations, which are much closer to the simulation for FPSs. The new definition of strong simulation we have introduced in this chapter can be considered as an extension of their definition by dropping some conditions. Please refer to Section 4.5 for an example.

Apart from these relations, most of the other preorders and equivalence relations defined for non-probabilistic models have been proposed for probabilistic systems in

the literature: such as trace distribution equivalence in [97, 119], testing preorders and equivalences [80, 34, 99, 38, 118, 81, 33].

4.5 Summary

In this chapter we have recalled strong and weak (bi-)simulation relations for Markov chains and their nondeterministic extensions. In the definition of strong simulations, the notion of weight function is used to match the distributions. Inspired by the definition of strong simulation for LMPs [49], we introduced another equivalence definition for strong simulations, which provides another (rather intuitive) view of simulations. Our definition can be considered as a relaxation of their definition: In their definition a simulation relation R must be a preorder while in our definition this condition is not imposed. As an example we consider the FPS depicted in Figure 4.9. Example 4.2.1, we have shown that $R = \{(s_1, s_2), (u_1, u_2), (v_1, v_2), (q_1, q_2)\}$ is a strong simulation, which implies that $s_1 \precsim s_2$. This relation is the minimal simulation relation (in respect to the number of pairs in it) containing (s_1, s_2). However, following the definition of simulations as in [49], a preorder is needed which contains more pairs than R, for example all of the identity relation over S.

Our alternative characterisation makes the simulation relations for Markov chains more understandable for those who are familiar with labelled transition systems: Our definition is a natural extension of the simulation relation for LTSs in which also no such restriction is imposed.

Chapter 5

Algorithms for Strong Simulations

We now come to one of the core contributions of this thesis: algorithms for deciding strong, strong probabilistic simulations and their experimental evaluation. Our algorithms reduce the theoretical complexity bounds drastically. We also report on experimental comparisons of these algorithms, together with various interesting optimisations and heuristics to accelerate the algorithm. The evaluation is carried out on both standard examples as well as randomly generated models.

The common scheme of decision algorithm for simulations is as follows. The algorithm starts with the relation $\{(s, s') \in S \times S \mid L(s) = L(s')\}$ which is guaranteed to be coarser than the simulation preorder $\precsim$. Then, the relation R will be refined. In each iteration of the refinement loop, it throws pairs (s, s') out of the relation if the corresponding simulation conditions are violated with respect to the current relation. In the context of labelled transitions systems, this means that s has a successor state t, but we cannot find a successor state t' of s' such that (t, t') is also in the current relation R. For DTMCs, this correspondence is formulated by weight functions for distributions $(\mathbf{P}(s, \cdot), \mathbf{P}(s', \cdot))$ with respect to the current relation R. In more detail, assuming that R is coarser than $\precsim$, for $(s, s') \in R$, if there does not exist a weight function for $(\mathbf{P}(s, \cdot), \mathbf{P}(s', \cdot))$ with respect to R, we can conclude that s' can not strongly simulate s. If the algorithm reached a fix-point, the strong simulation preorder is obtained.

According to Lemma 4.2.1, checking the weight function reduces to checking whether the maximum flow over the network (constructed out of $(\mathbf{P}(s, \cdot), \mathbf{P}(s', \cdot))$ and the current relation) has value 1. For a fixed pair (s, s'), we observe that the networks for it across iterations of the refinement loop are very similar: They differ from iteration to iteration only by deletion of some edges induced by the successive clean up of R. We exploit this by adapting the parametric maximum flow algorithm [53] to compute the maximum flows for the arising sequences of similar networks achieving drastic improvements in complexity. The algorithm is then extended to deal strong simulation for other models.

The maximum flow approach cannot be applied directly to strong probabilistic simulations. The reason is that the combined transition involves the quantification of the constants ranging over reals. We show that the strong probabilistic simulation for PAs and CPAs can be computed by solving LP problems which are decidable in polynomial time.

While the new parametric maximum flow based algorithm for deciding strong sim-

ulations has a tremendous drop in theoretical complexity, we are surprised to find that its practical implementation comes with an overhead that makes it considerably weaker than expected. The gap between theoretical and practical efficiency is not caused by "the constant factors" but by the fact that the corner cases that blow up the worst case complexity are rare in practice.

Organisation of this Chapter. In Section 5.1 we first introduce the notion of strong simulation up to relation R. In Section 5.2 we present algorithms for deciding the strong simulation preorder. In Section 5.3 we introduce algorithms for deciding strong probabilistic simulations. Section 5.4 provides experimental results for strong simulations, and Section 5.5 discusses related works. We conclude this chapter with a summary in Section 5.6.

5.1 Strong Simulation up to R

Strong simulation up to R is the key component of the decision algorithm: In each refinement loop of the simulation algorithm, it is checked for each pair $(s, s') \in R$ whether s' can simulates s up to the current relation R. It can be considered as a relaxation of the simulation condition:

Definition 5.1.1. *Let $\mathcal{M} = (S, \mathbf{P}, L)$ be an FPS. For an arbitrary relation $R \subseteq S \times S$ with $s_1\ R\ s_2$, we say that s_2 simulates s_1 strongly up to R, denoted $s_1 \precsim_R s_2$, if $L(s_1) = L(s_2)$ and $\mathbf{P}(s_1, \cdot) \sqsubseteq_R \mathbf{P}(s_2, \cdot)$.*

Obviously $s_1 \precsim_R s_2$ does not imply $s_1 \precsim_{\mathcal{M}} s_2$ unless R is a strong simulation, since only the first step is considered for $\precsim_R$. By definition, the following lemma is obvious:

Lemma 5.1.1. *Let $R \subseteq S \times S$. Then, R is a strong simulation if and only if for all $s_1 R s_2$ it holds that $s_1 \precsim_R s_2$.*

Example 5.1.1. Consider the FPS in Figure 5.1. Let $R = \{(s_1, s_2), (w_1, w_2)\}$. Since $L(q_1) \neq L(q_2)$ we have that $w_1 \not\precsim w_2$. Thus, R is not a strong simulation. However, $s_1 \precsim_R s_2$, as the weight function is given by $\Delta(w_1, w_2) = 1$. Let $R' = \{(s_1, s_2)\}$, then, $s_1 \not\precsim_{R'} s_2$.

Figure 5.1: A simple FPS for illustrating the simulation up to R.

These conventions extend to strong simulation up to R for DTMCs, CTMCs, PAs and CPAs in an obvious way. For PAs and CPAs, strong probabilistic simulation up to R, denoted by $\precsim_R^p$, is also defined analogously.

Algorithm 2 Basic algorithm to decide strong simulation.

$\textsc{SimRel}_s(\mathcal{M})$

2.1: $R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2)\}$ and $i \leftarrow 0$
2.2: **repeat**
2.3: $\quad i \leftarrow i + 1$
2.4: $\quad R_{i+1} \leftarrow \emptyset$
2.5: $\quad$ **for all** $(s_1, s_2) \in R_i$ **do**
2.6: $\quad\quad$ **if** $s_1 \precsim_{R_i} s_2$ **then**
2.7: $\quad\quad\quad R_{i+1} \leftarrow R_{i+1} \cup \{(s_1, s_2)\}$
2.8: **until** $R_{i+1} = R_i$
2.9: **return** R_i

5.2 Strong Simulation

We first recall the basic algorithm to decide strong simulation preorder (i. e., the relation $\precsim$) in Subsection 5.2.1. Then, we refine this algorithm to deal with strong simulations on FPSs, DTMCs and CTMCs in Subsection 5.2.2. In Subsection 5.2.3, we consider PAs and CPAs.

5.2.1 Basic Algorithm to Decide Strong Simulation

The algorithm in [9], copied as $\textsc{SimRel}_s$ in Algorithm 2, takes as a parameter a model, which, for now, is an FPS $\mathcal{M}$. The subscript 's' stands for strong simulation; a very similar algorithm, namely $\textsc{SimRel}_w$, will be used for weak simulation later. To calculate the strong simulation relation for $\mathcal{M}$, the algorithm starts with the initial relation $R_1 = \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2)\}$ which is coarser than $\precsim_{\mathcal{M}}$. In iteration i, it generates R_{i+1} from R_i by deleting each pair (s_1, s_2) from R_i if s_2 cannot strongly simulate s_1 up to R_i, i. e., $s_1 \not\precsim_{R_i} s_2$. This proceeds until there is no such pair left, i. e., $R_{i+1} = R_i$. Invariantly throughout the loop it holds that R_i is coarser than $\precsim_{\mathcal{M}}$ (i. e., $\precsim_{\mathcal{M}}$ is a subrelation of R_i). We obtain the strong simulation preorder $\precsim_{\mathcal{M}} = R_i$, once the algorithm terminates.

The decisive part of the algorithm is the check in line 2.6, i. e., whether $s_1 \precsim_{R_i} s_2$. From the first and second clauses of Lemma 4.2.1, it is equivalent to compute the maximum flow of the network $\mathcal{N}(\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot), R_i)$ and check whether it has value 1. Recall this network is constructed (see Section 4.2) out of $\mathbf{P}(s_1, \cdot)$, $\mathbf{P}(s_2, \cdot)$ and R_i. For two states s_1, s_2 of an FPS or a CTMC and a relation $R \subseteq S \times S$, we let $\mathcal{N}(s_1, s_2, R)$ denote the network $\mathcal{N}(\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot), R)$ as defined in Subsection 4.2.1. We recall the correctness and complexity of $\textsc{SimRel}_s$ which will also be used later.

Theorem 5.2.1 ([9]). *Let $\mathcal{M} = (S, \mathbf{P}, L)$ be an FPS and let n denote the number of states. If $\textsc{SimRel}_s(\mathcal{M})$ terminates, the returned relation equals $\precsim_{\mathcal{M}}$. Moreover, $\textsc{SimRel}_s(\mathcal{M})$ runs in time $\mathcal{O}(n^7 / \log n)$ and in space $\mathcal{O}(n^2)$.*

Proof. First we show that after the last iteration (say iteration k), it holds that $\precsim$ is coarser than R_k: It holds that $R_{k+1} = R_k$, thus for all $(s_1, s_2) \in R_k$, we have that

$s_1 \precsim_{R_k} s_2$. As for all $(s_1, s_2) \in R_k \subseteq R_1$, we have $L(s_1) = L(s_2)$, R_k is a strong simulation relation by Definition 4.1.2, thus $\precsim$ is coarser than R_k.

Now we show by induction that the loop of the algorithm invariantly ensures that R_i is coarser than $\precsim$. Assume $i = 1$. By definition of strong simulation, $s_1 \precsim s_2$ implies $L(s_1) = L(s_2)$. Thus, the initial relation R_1 is coarser than the simulation relation $\precsim$. Now assume that R_i is coarser than $\precsim$ for some $1 \leq i < k$; we will show that also R_{i+1} is coarser than $\precsim$. Pick a pair $(s_1, s_2) \in \precsim$ arbitrarily. By Definition 4.1.2, $\mathbf{P}(s_1, \cdot) \sqsubseteq_{\precsim} \mathbf{P}(s_2, \cdot)$, so there exists a weight function for $(\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot))$ with respect to $\precsim$. As R_i is coarser than $\precsim$ by induction hypothesis, we conclude that $\mathbf{P}(s_1, \cdot) \sqsubseteq_{R_i} \mathbf{P}(s_2, \cdot)$ according to Lemma 4.1.2. By Lemma 5.1.1, $s_1 \precsim_{R_i} s_2$. This implies that $(s_1, s_2) \in R_{i+1}$ by line 2.6 for all $s_1 \precsim s_2$. Therefore, R_{i+1} is coarser than $\precsim$ for all $i = 1 \ldots, k$.

Now we show the complexity. For one network $\mathcal{N}(s_1, s_2, R_i) = (V, E, c)$, the sizes of the vertices $|V|$ and edges $|E|$ are bounded by $2n + 4$ and $(n + 1)^2 + 2n$, respectively. The number of edges meets the worst case bound $\mathcal{O}(n^2)$. To the best of our knowledge, the best complexity of the maximum flow computation for the network G is $\mathcal{O}(|V|^3 / \log |V|) = \mathcal{O}(n^3 / \log n)$ [31, 55].

In the algorithm SimRel$_s$, only one pair, in the worst case, is removed from R_i in iteration i, which indicates that the test whether $s_1 \precsim_{R_i} s_2$ is called $|R_1|$ times, $|R_1| - 1$ times and so on. Altogether it is bounded by $\sum_{i=1}^{|R_1|} i \leq \sum_{i=1}^{n^2} i \in \mathcal{O}(n^4)$. Hence, the overall time complexity is $\mathcal{O}(n^7 / \log n)$. The space complexity is $\mathcal{O}(n^2)$ because of the representation of the transitions in $\mathcal{N}(s_1, s_2, R_i)$. ■

5.2.2 An Improved Algorithm for FPSs

We first analyse the behaviour of SimRel$_s$ in more detail. For this, we consider an arbitrary pair (s_1, s_2), and assume that (s_1, s_2) stays in relation $R_1, \ldots, R_k$ throughout the iterations $i = 1, \ldots, k$, until the pair is either found not to satisfy $s_1 \precsim_{R_k} s_2$ or the algorithm terminates with a fix-point after iteration k. Then altogether the maximum flow algorithms are run k-times for this pair. However, the networks $\mathcal{N}(s_1, s_2, R_i)$ constructed in successive iterations are very similar, and may often be identical across iterations: They differ from iteration to iteration only by deletion of some edges induced by the successive cleanup of R_i. For our particular pair (s_1, s_2) the network might not change at all in some iterations, because the deletions from R_i do not affect their direct successors. We are going to exploit this observation by an algorithm that reuses the already computed maximum flow, in a way that whatever happens is good: If no changes occur from $\mathcal{N}(s_1, s_2, R_{i-1})$ to $\mathcal{N}(s_1, s_2, R_i)$, then the maximum flow is the same as the one in the previous iteration. If changes do occur, the preflow algorithm can be applied to get the new maximum flow very fast, using the maximum flow and distance function constructed in the previous iteration as a starting point.

To understand the algorithm, we look at the network $\mathcal{N}(s_1, s_2, R_1)$. Let $D_1, \ldots, D_k$ be pairwise disjoint subsets of R_1, which correspond to the pairs deleted from R_1 in iteration i, so $R_{i+1} = R_i \setminus D_i$ for $1 \leq i \leq k$. Let $f_i^{(s_1,s_2)}$ denote the maximum flow of the network $\mathcal{N}(s_1, s_2, R_i)$ for $1 \leq i \leq k$. We sometimes omit the superscript (s_1, s_2) in the parameters if the pair (s_1, s_2) is clear from the context. We address the problem of checking $|f_i| = 1$ for all $i = 1, \ldots, k$. Our algorithm *sequence of maximum flows*

Algorithm 3 Algorithm for a sequence of maximum flows.

SMF$(i, \mathcal{N}(s_1, s_2, R_{i-1}), f_{i-1}, d_{i-1}, D_{i-1})$

3.1: $\mathcal{N}(s_1, s_2, R_i) \leftarrow \mathcal{N}(s_1, s_2, R_{i-1} \setminus D_{i-1})$ and $f_i \leftarrow f_{i-1}$ and $d_i \leftarrow d_{i-1}$
3.2: **for all** $(u_1, u_2) \in D_{i-1}$ **do**
3.3: $\quad f_i(\overline{u_2}, \searrow) \leftarrow f_i(\overline{u_2}, \searrow) - f_i(u_1, \overline{u_2})$
3.4: $\quad f_i(u_1, \overline{u_2}) \leftarrow 0$
3.5: Apply the preflow algorithm to calculate the maximum flow for the network $\mathcal{N}(s_1, s_2, R_i)$, but initialise the preflow to f_i and the distance function to d_i.
3.6: **return** $(|f_i| = 1, \mathcal{N}(s_1, s_2, R_i), f_i, d_i)$

SMF$_{init}(i, s_1, s_2, R_i)$

3.11: Initialise the network $\mathcal{N}(s_1, s_2, R_i)$.
3.12: Apply the preflow algorithm to calculate the maximum flow for the network $\mathcal{N}(s_1, s_2, R_i)$.
3.13: **return** $(|f_i| = 1, \mathcal{N}(s_1, s_2, R_i), f_i, d_i)$

SMF$(i, \mathcal{N}(s_1, s_2, R_{i-1}), f_{i-1}, d_{i-1}, D_{i-1})$ is shown as Algorithm 3. It executes iteration i of a parametric flow algorithm, where $\mathcal{N}(s_1, s_2, R_{i-1})$ is the network for (s_1, s_2) and f_{i-1} and d_{i-1} are the flow and the distance function resulting from the previous iteration $i-1$; and D_{i-1} is a set of edges that have to be deleted from $\mathcal{N}(s_1, s_2, R_{i-1})$ to get the current network. The algorithm returns a tuple, in which the first component is a boolean that tells whether $|f_i| = 1$; it also returns the new network $\mathcal{N}(s_1, s_2, R_i)$, flow f_i and distance function d_i to be reused in the next iteration. SMF is inspired by the *parametric maximum algorithm* in [53, p. 34]. A variant of SMF is used in the first iteration, shown in lines 3.11–3.13.

This algorithm for sequence of maximum flow problems is called in an improved version of SIMREL$_s$ shown as Algorithm 4. Lines 4.2–4.7 contain the first iteration, very similar to the first iteration of Algorithm 2 (lines 2.4–2.7). At line 4.4 we prepare for later iterations the set

$$\mathbf{Listener}_{(s_1,s_2)} = \{(u_1, u_2) \mid u_1 \in pre(s_1) \wedge u_2 \in pre(s_2) \wedge L(u_1) = L(u_2)\} \quad ,$$

where $pre(s) = \{t \in S \mid \mathbf{P}(t, s) > 0\}$. This set contains all pairs (u_1, u_2) such that the network $\mathcal{N}(u_1, u_2, R_1)$ contains the edge $(s_1, \overline{s_2})$. Iteration i (for $i > 1$) of the loop (lines 4.10–4.18) calculates R_{i+1} from R_i. In lines 4.11–4.14, we collect edges that should be removed from $\mathcal{N}(u_1, u_2, R_{i-1})$ in the sets $D_{i-1}^{(u_1,u_2)}$. At line 4.16, the algorithm SMF constructs the maximum flow for parameters using information from iteration $i-1$. It uses the set $D_{i-1}^{(s_1,s_2)}$ to update the network $\mathcal{N}(s_1, s_2, R_{i-1})$, flow f_{i-1}, a distance function d_{i-1}; then it constructs the maximum flow f_i for the network $\mathcal{N}(s_1, s_2, R_i)$. If SMF returns true, (s_1, s_2) is inserted into R_{i+1} and survives this iteration (line 4.18).

Consider the algorithm SMF and assume that $i > 1$. At lines 3.1–3.4, we remove the edges D_{i-1} from the network $\mathcal{N}(s_1, s_2, R_{i-1})$ and generate the preflow f_i based on the

Algorithm 4 Improved algorithm for deciding strong simulation for FPSs.

$\textsc{SimRel}_s^{\text{FPS}}(\mathcal{M})$

4.1: $R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2)\}$ and $i \leftarrow 1$
4.2: $R_2 \leftarrow \emptyset$
4.3: **for all** $(s_1, s_2) \in R_1$ **do**
4.4: $\quad$ $\textbf{Listener}_{(s_1,s_2)}$
$\quad\quad \leftarrow \{(u_1, u_2) \mid u_1 \in pre(s_1) \wedge u_2 \in pre(s_2) \wedge L(u_1) = L(u_2)\}$
4.5: $\quad$ $(match, \mathcal{N}(s_1, s_2, R_1), f_1^{(s_1,s_2)}, d_1^{(s_1,s_2)}) \leftarrow \textsc{Smf}_{init}(1, s_1, s_2, R_1)$
4.6: $\quad$ **if** *match* **then**
4.7: $\quad\quad$ $R_2 \leftarrow R_2 \cup \{(s_1, s_2)\}$
4.8: **while** $R_{i+1} \neq R_i$ **do**
4.9: $\quad$ $i \leftarrow i + 1$
4.10: $\quad$ $R_{i+1} \leftarrow \emptyset$ and $D_{i-1} \leftarrow R_{i-1} \setminus R_i$
4.11: $\quad$ **for all** $(s_1, s_2) \in R_i$ **do**
4.12: $\quad\quad$ $D_{i-1}^{(s_1,s_2)} \leftarrow \emptyset$
4.13: $\quad$ **for all** $(s_1, s_2) \in D_{i-1}, \quad (u_1, u_2) \in \textbf{Listener}_{(s_1,s_2)} \cap R_{i-1}$ **do**
4.14: $\quad\quad$ $D_{i-1}^{(u_1,u_2)} \leftarrow D_{i-1}^{(u_1,u_2)} \cup \{(s_1, s_2)\}$
4.15: $\quad$ **for all** $(s_1, s_2) \in R_i$ **do**
4.16: $\quad\quad$ $(match, \mathcal{N}(s_1, s_2, R_i), f_i^{(s_1,s_2)}, d_i^{(s_1,s_2)})$
$\quad\quad\quad \leftarrow \textsc{Smf}(i, \mathcal{N}(s_1, s_2, R_{i-1}), f_{i-1}^{(s_1,s_2)}, d_{i-1}^{(s_1,s_2)}, D_{i-1}^{(s_1,s_2)})$
4.17: $\quad\quad$ **if** *match* **then**
4.18: $\quad\quad\quad$ $R_{i+1} \leftarrow R_{i+1} \cup \{(s_1, s_2)\}$
4.19: **return** R_i

flow f_{i-1}, which is the maximum flow of the network $\mathcal{N}(s_1, s_2, R_{i-1})$, by

- setting $f_i(u_1, \overline{u_2}) = 0$ for all deleted edges $(u_1, u_2) \in D_{i-1}$, and
- reducing $f_i(\overline{u_2}, \searrow)$ such that the preflow f_i becomes consistent with the (relaxed) flow conservation rule.

The excess $e(v)$ is increased if there exists $(v, w) \in D_{i-1}$ such that $f_{i-1}(v, w) > 0$, and unchanged otherwise. Hence, f_i after line 3.4 is a preflow. The distance function $d_{i-1} = d_i$ is still valid for this preflow, since removing the set of edges D_{i-1} does not introduce new residual edges. This guarantees that, at line 3.5, the *preflow algorithm* finds a maximum flow over the network $\mathcal{N}(s_1, s_2, R_i)$. In line 3.6, SMF returns whether the flow has value 1 together with information to be reused in the next iteration. (If $|f_k| < 1$ at some iteration k, then $|f_j| < 1$ for all iterations $j \geq k$ because deleting edges does not increase the maximum flow. In that case, it would be sufficient to return **false**.) We prove the correctness and complexity of the algorithm SMF:

Lemma 5.2.2 (Correctness of SMF)**.** *Let $(s_1, s_2) \in R_1$. Then, SMF_{init} returns true iff $s_1 \precsim_{R_1} s_2$.*

For some $i > 1$, let $\mathcal{N}(s_1, s_2, R_{i-1})$, f_{i-1}, and d_{i-1} be as returned by some earlier call to SMF *or* SMF_{init}*. Let $D_{i-1} = (R_{i-1} \setminus R_i) \cap (post(s_1) \times post(s_2))$ be the set of edges that will be removed from the network $\mathcal{N}(s_1, s_2, R_{i-1})$ during the $(i-1)$th call of* SMF*. Then, the $(i-1)$th call of* SMF *returns true iff $s_1 \precsim_{R_i} s_2$.*

Proof. By Lemma 4.2.1, SMF_{init} returns true iff $|f_1| = 1$, which is equivalent to $s_1 \precsim_{R_1} s_2$. Let $i > 1$. As discussed, at the beginning of line 3.5, the function f_{i-1} is a flow (thus a preflow) with value 1, and the distance function d_{i-1} is a valid distance function. It follows directly from the correctness of the preflow algorithm [3] (cf. Lemma 3.3.1 in Chapter 3) that after line 3.5, f_i is a maximum flow for $\mathcal{N}(s_1, s_2, R_i)$. Thus, SMF returns true (i.e. $|f_i| = 1$) which is equivalent to $s_1 \precsim_{R_i} s_2$. ■

Lemma 5.2.3 (Complexity of SMF)**.** *Consider the pair of states (s_1, s_2) and assume that $|post(s_1)| \leq |post(s_2)|$. All calls to $\text{SMF}(i, \mathcal{N}(s_1, s_2, \cdot), \cdots)$ related to (s_1, s_2) together run in time $\mathcal{O}(|post(s_1)|\,|post(s_2)|^2)$.*

Proof. In the bipartite network $\mathcal{N}(s_1, s_2, R_1)$, V is partitioned into subsets V_1 with $\searrow \in V_1$ and V_2 with $\nearrow \in V_2$ as defined in Subsection 4.2.1. The number of edges of the network is at most $|E| \leq |V_1||V_2| - 1$. Assume that $|V_i| = |post(s_i)| + 1$ for $i = 1, 2$, thus $|V_1| \leq |V_2|$. In our sequence of maximum flow problems, the number of calls, denoted by k, is bounded by the number of edges, i.e., $k \leq |V_1||V_2| - 1$. We split the work being done by all calls to $\text{SMF}(i, \mathcal{N}(s_1, s_2, \cdot), \ldots)$ into time spend for edge deletions, relabels, non-saturating pushes, saturating pushes.

All edge deletions take time proportional to $\sum_{i=1}^{k} |D_i|$, which is less than the number of edges in the network. Therefore, edge deletions take time $\mathcal{O}(|V_1||V_2|)$. For all $v \in V$, it holds that $d_{i+1}(v) = d_i(v)$, i.e., the labelling function at the beginning of iteration $i+1$ is the same as the labelling function at the end of iteration i. Since $d_i(v)$ is bounded by $4|V_1|$ for all iterations i, using exact the same argument as for a single maximum flow

calculation (cf. the proof of Lemma 3.3.4), we see that the time spent for relabels and saturating pushes is bounded by $\mathcal{O}(|V_1||E|)$.

We briefly recall the analysis of the number of non-saturating pushes, which is very similar to the proof of Theorem 2.2 in [60] where Max-d version is used (cf. Section 3.3). As in [60], assume that in iteration $l \leq k$ of SMF, the last relabelling action occurs. By Lemma 3.3.4, the number of relabels is bounded by $\mathcal{O}(|V_1||V|)$. Between any two relabelling actions, at most $|V|$ non-saturating pushes can occur for the same reason as for a single preflow algorithm (cf. Lemma 3.3.5). Thus, the number of non-saturating pushes before the iteration l is bounded by $\mathcal{O}(|V_1||V|^2)$. Since the distance function does not change after iteration l any more, inside any of the iterations $l' \geq l$, there are again at most $n-1$ non-saturating pushes. Hence, the number of non-saturating pushes is bounded by $|V_1||V|^2+(k+1-l)(|V|-1) \in \mathcal{O}(|V_1||V|^2+k|V|)$. Since $k \leq |V_1||V_2|-1$, and $|V| \leq 2|V_2|$, thus, the overall time complexity is $\mathcal{O}(|V_1||V_2|^2) = \mathcal{O}(|post(s_1)|\,|post(s_2)|^2)$ as required. ■

Now we give the correctness and complexity of the algorithm SIMREL for FPSs:

Theorem 5.2.4 (Correctness for FPSs)**.** *If* $\text{SIMREL}_s^{\text{FPS}}(\mathcal{M})$ *terminates, the returned relation equals* $\precsim_{\mathcal{M}}$*.*

Proof. By Lemma 5.2.2, $\text{SMF}_{init}(i, s_1, s_2, R_1)$ returns true in iteration $i = 1$ iff $s_1 \precsim_{R_1} s_2$; and $\text{SMF}(i, \mathcal{N}(s_1, s_2, R_{i-1}), \ldots)$ returns true in iteration $i > 1$ iff $s_1 \precsim_{R_i} s_2$. The rest of the correctness proof is the same as the proof of Theorem 5.2.1. ■

Theorem 5.2.5 (Complexity for FPSs)**.** *The algorithm* $\text{SIMREL}_s^{\text{FPS}}(\mathcal{M})$ *runs in time* $\mathcal{O}(m^2 n)$ *and in space* $\mathcal{O}(m^2)$*. If the fanout is bounded by a constant, it has complexity* $\mathcal{O}(n^2)$*, both in time and space.*

Proof. We first show the space complexity. In most cases, it is enough to store information from the previous iteration until the corresponding structure for the current iteration is calculated. Obviously, the size of the set $\mathbf{Listener}_{(s_1,s_2)}$ is bounded by $|pre(s_1)|\,|pre(s_2)|$ where $pre(s) = \{t \in S \mid \mathbf{P}(t,s) > 0\}$. Summing over all (s_1, s_2), we get

$$\sum_{s_1 \in S} \sum_{s_2 \in S} |pre(s_1)|\,|pre(s_2)| = \sum_{s_1 \in S} |pre(s_1)| \sum_{s_2 \in S} |pre(s_2)| = m^2$$

Assume we run iteration i. For every pair (s_1, s_2), we generate the set $D_{i-1}^{(s_1,s_2)}$ and the network $\mathcal{N}(s_1, s_2, R_i)$ together with f_i and d_i. The size of $D_{i-1}^{(s_1,s_2)}$ is bounded by $|post(s_1)|\,|post(s_2)|$. Summing over all (s_1, s_2), we get the bound $\mathcal{O}(m^2)$. The number of edges of the initial network $\mathcal{N}(s_1, s_2, R_1)$ (together with f_i and d_i) is in $\mathcal{O}(|post(s_1)|\,|post(s_2)|)$. Summing over all (s_1, s_2) yields a memory consumption in $\mathcal{O}(m^2)$ again. Hence, the overall space complexity is $\mathcal{O}(m^2)$.

Now we show the time complexity. We observe that a pair (s_1, s_2) belongs to D_i in at most one iteration. Therefore, the time needed in lines 4.11–4.14 in all iterations together is bounded by the size of all sets $\mathbf{Listener}_{(s_1,s_2)}$, which is $\mathcal{O}(m^2)$. We analyse the time needed for all calls to the algorithm SMF. Recall that the fanout g equals

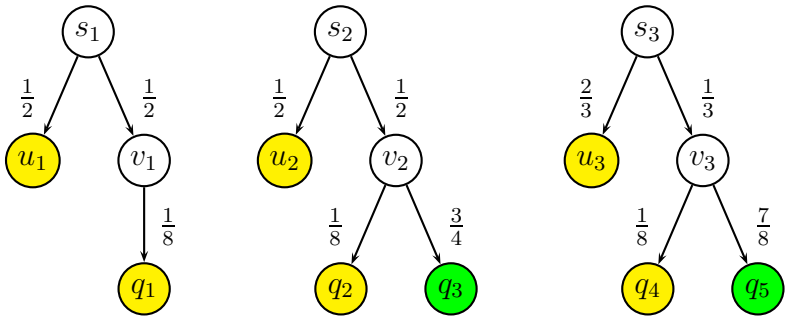

Figure 5.2: A simple FPS.

$\max_{s\in S} |post(s)|$, and therefore $|post(s_i)| \leq g$ for $i = 1, 2$. By Lemma 5.2.3, the complexity attributed to the pair (s_1, s_2) is bounded by $\mathcal{O}(g\,|post(s_1)|\,|post(s_2)|)$. Taking the sum over all possible pairs, we get

$$\sum_{s_1\in S}\sum_{s_2\in S} g\,|post(s_1)|\,|post(s_2)| = g\sum_{s_1\in S}|post(s_1)|\sum_{s_2\in S}|post(s_2)| = gm^2$$

Hence, the overall time complexity is $gm^2 \in \mathcal{O}(m^2 n)$. If g is bounded by a constant, we have $m \leq gn$, and the time complexity is $gm^2 \leq g^3 n^2 \in \mathcal{O}(n^2)$. In this case the space complexity is also $\mathcal{O}(n^2)$. ■

Example 5.2.1. Consider the FPS in Figure 5.2. Initially, R_1 contains 76 pairs. Since the absorbing states simulate each other trivially, the interesting part is the pairs consisting of s_i, v_j for $i, j \in \{1, 2, 3\}$.

Firstly, we discuss which pairs will be removed from R_1 in the first iteration. We consider three groups. The first group is the pairs (v_2, v_1) and (v_3, v_1). The network for (v_2, v_1) is depicted on the left side of Figure 5.3. Obviously, the maximum flow has value $\frac{1}{4}$, since we cannot send any flow through q_3 which cannot be simulated by any successor state of v_1. Thus, this pair will be moved in the first iteration. Very similar the pair (v_3, v_1) will also be moved in the first iteration. The second group is the pairs (s_1, s_3), (s_3, s_1), (s_2, s_3), (s_3, s_2). The networks for pairs (s_1, s_3) and (s_2, s_3) are the same except the nodes labelling, and for (s_3, s_1) and (s_3, s_2) are the same. For each of these four networks, it is easy to see that the maximum flow has value $\frac{1}{2} + \frac{1}{3} = \frac{5}{6}$, thus they can be removed from the relation in the first iteration. Consider the last group: pairs (s_i, v_j) with $i, j \in \{1, 2, 3\}$, and pairs (v_j, s_i) with $j \in \{2, 3\}$ and $i \in \{1, 2, 3\}$. The network for each of these pair can be constructed, and the maximum flow of it has always smaller value than one because of similar reason as for the first group: there exists a vertex on the left side where we cannot send any flow through it.

As a successful check we consider the pair (s_2, s_1). The network is depicted on the right side of Figure 5.3. Obviously, the maximum flow has value 1: $\frac{1}{2}$ amount of flow along the upper path and $\frac{1}{2}$ amount of flow along the lower path. Thus, $s_2 \precsim_{R_1} s_1$. The checks for the pairs (s_2, s_1), (v_1, v_2), (v_2, v_3) and (v_1, s_i) with $i \in \{1, 2, 3\}$ are also successful.

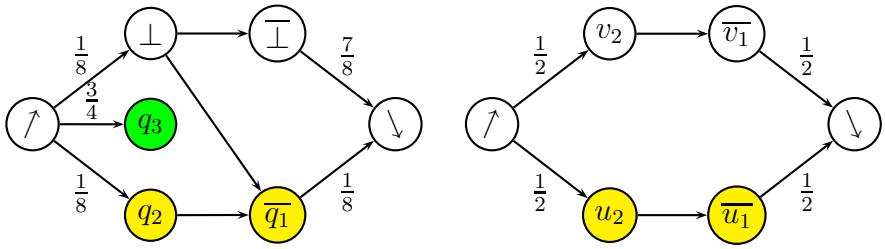

Figure 5.3: The networks for the pair (v_2, v_1) on the left, and for the pair (s_2, s_1).

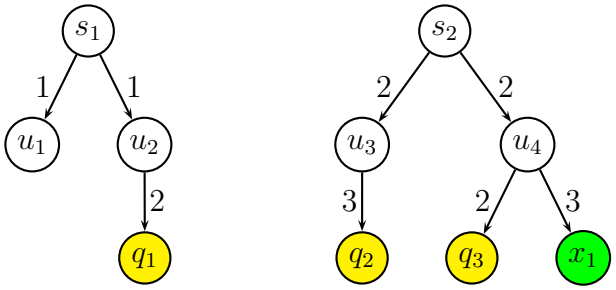

Figure 5.4: A CTMC example.

Let R_2 denote the remaining pairs, and we consider the second iteration. For all pairs in R_2, only the network for the pair (s_2, s_1) is changed: since $(s_2, s_1) \in \mathbf{Listener}_{(v_2,v_1)}$, the network $\mathcal{N}(s_2, s_1, R_2)$ is obtained by removing the edge $(v_2, \overline{v_1})$. After removing the edge, the node v_2 has excess $\frac{1}{2}$. However, there is no way to push the excess further. Thus, the maximum flow is smaller than 1, and we have that $s_2 \not\precsim_{R_2} s_1$.

In the third iteration no pairs will be removed from it, thus we get the simulation preorder for this FPS.

Algorithms for DTMCs and CTMCs

We now consider how to handle DTMCs and CTMCs. Since each DTMC is a special case of an FPS the algorithm $\textsc{SimRel}_s^{\text{FPS}}$ applies directly.

Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC. Recall that $s \precsim_{\mathcal{M}} s'$ holds if $s \precsim_{emb(\mathcal{M})} s'$ in the embedded DTMC, and s' is faster than s. We can ensure the additional rate condition by incorporating it into the initial relation R. More precisely, initially R contains only those pair (s, s') such that $L(s) = L(s')$, and that the state s' is faster than s, i. e., we replace line 4.1 of the algorithm by

$$R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \wedge \mathbf{R}(s_1, S) \leq \mathbf{R}(s_2, S)\}$$

to ensure the additional rate condition of Definition 4.1.3. In the refinement steps afterwards, only the weight function conditions need to be checked with respect to the current relation in the embedded DTMC. Thus, we arrive at an algorithm for CTMCs with the same time and space complexity as for FPSs.

Example 5.2.2. Consider the CTMC in Figure 5.4. Consider the pair $(s_1, s_2) \in R_1$.

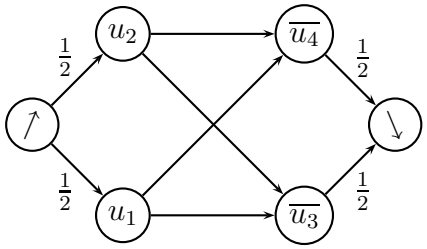

Figure 5.5: The network $\mathcal{N}(s_1, s_2, R_1)$.

The network $\mathcal{N}(s_1, s_2, R_1)$ is depicted in Figure 5.5. Assume that we get the maximum flow f_1 which sends $\frac{1}{2}$ amount of flow along the path $\nearrow, u_2, \overline{u_4}, \searrow$ and $\frac{1}{2}$ amount of flow along $\nearrow, u_1, \overline{u_3}, \searrow$. Hence, the check for (s_1, s_2) is successful in the first iteration. The checks for the pairs (u_1, u_3), (u_1, u_4) and (u_2, u_3) are also successful in the first iteration. However, the check for the pair (u_2, u_4) fails, as the probability to go from u_4 to q_3 in the embedded DTMC is $\frac{2}{5}$, while the probability to go from u_2 to q_1 in the embedded DTMC is 1.

In the second iteration, $\mathcal{N}(s_1, s_2, R_2)$ is obtained from $\mathcal{N}(s_1, s_2, R_1)$ by deleting the edge $(u_2, \overline{u_4})$. In $\mathcal{N}(s_1, s_2, R_2)$, the flows on $(u_2, \overline{u_4})$ and on $(\overline{u_4}, \searrow)$ are set to 0, and the vertex u_2 has a positive excess $\frac{1}{2}$. Applying the preflow algorithm, we push the excess from u_2, along $\overline{u_3}, u_1, \overline{u_4}$ to $\searrow$. We get a maximum flow f_2 for $\mathcal{N}(s_1, s_2, R_2)$ which sends $\frac{1}{2}$ amount of flow along the path $\nearrow, u_2, \overline{u_3}, \searrow$ and $\frac{1}{2}$ amount of flow along $\nearrow, u_1, \overline{u_4}, \searrow$. Hence, the check for (s_1, s_2) is also successful in the second iteration. Once the fix-point is reached, R still contains (s_1, s_2).

Discussions

In this section we have introduced parametric maximum flow based algorithm for deciding strong simulation for FPSs which is then extended for DTMCs and CTMCs. Lemmas 4.1.6 and 4.1.7 show that, for these Markov models, the simulation equivalence is the same as the bisimulation relation. Thus, we can first compute the bisimulation equivalence relation, build the quotient[1] automaton, then compute the simulation preorder in the quotient automaton. This has time complexity $\mathcal{O}(m \log n + m_{\sim}^2 n_{\sim})$ where $n_{\sim}$ and $m_{\sim}$ are the number of states and transitions in the bisimulation quotient automaton respectively. This can accelerate the algorithm a lot if the bisimulation quotient is much smaller than the original one. However, in the worst case the complexity remains the same $\mathcal{O}(m^2 n)$.

5.2.3 Algorithms for PAs and CPAs

In this subsection we present algorithms for deciding strong simulations for PAs and CPAs. It takes the skeleton of the algorithm for FPSs: it starts with a relation R which is coarser than $\precsim$, and then refines R until $\precsim$ is achieved. In the refinement loop, a

[1]The definition of quotient automata for PAs shall be defined in Chapter 7. Then, the notion of quotient automata for FPSs can be considered as a special case of it.

pair (s, s') is thrown out of the pair if the corresponding strong simulation conditions are violated with respect to the current relation. For PAs, this means that there exists an α-successor distribution μ of s, such that for all α-successor distribution μ' of s', we cannot find a weight function for (μ, μ') with respect to the current relation R.

Let $\mathcal{M} = (S, Act, \mathbf{P}, L)$ be a PA. We aim to extend Algorithm 4 to determine the strong simulation on PAs. For a pair (s_1, s_2), assume that $L(s_1) = L(s_2)$ and that $Act(s_1) \subseteq Act(s_2)$, which is guaranteed by the initialisation. We consider line 4.17, which checks the condition $\mathbf{P}(s_1, \cdot) \sqsubseteq_{R_i} \mathbf{P}(s_2, \cdot)$ using SMF. By Definition 4.1.4 of strong simulation for PAs, we should instead check the condition

$$\forall \alpha \in Act.\ \forall s_1 \xrightarrow{\alpha} \mu_1.\ \exists s_2 \xrightarrow{\alpha} \mu_2 \text{ with } \mu_1 \sqsubseteq_{R_i} \mu_2 \quad . \tag{5.1}$$

Recall the condition $\mu_1 \sqsubseteq_{R_i} \mu_2$ holds iff the maximum flow of $\mathcal{N}(\mu_1, \mu_2, R_i)$ has value one. Sometimes, we write $\mathcal{N}(s_1, \alpha, \mu_1, s_2, \mu_2, R_i)$ to denote the network $\mathcal{N}(\mu_1, \mu_2, R_i)$ associated with the pair (s_1, s_2) with respect to action α.

Our first goal is to extend SMF to check Condition 5.1 for a fixed action α and α-successor distribution μ_1 of s_1. To this end, we introduce a list $Sim^{(s_1,\alpha,\mu_1,s_2)}$ that contains all potential candidates of α-successor distributions of s_2 which could be used to establish the condition $\mu_1 \sqsubseteq_R \mu_2$ for the relation R considered. The set $Sim^{(s_1,\alpha,\mu_1,s_2)}$ is represented as a list. This and some subsequent notations are similar to those used by Baier *et al.* in [9]. We use the function $\mathbf{head}(\cdot)$ to read the first element of a list; $\mathbf{tail}(\cdot)$ to read all but the first element of a list; and $\mathbf{empty}(\cdot)$ to check whether a list is empty. As long as the network for a fixed candidate $\mu_2 = \mathbf{head}(Sim^{(s_1,\alpha,\mu_1,s_2)})$ allows a flow of value 1 over the iterations, we stick to it, and we can reuse the flow and distance function from previous iterations. If by deleting some edges from $\mathcal{N}(\mu_1, \mu_2, R)$, its flow value falls below 1, we delete μ_2 from $Sim^{(s_1,\alpha,\mu_1,s_2)}$ and pick the next candidate.

The algorithm ACTSMF, shown as Algorithm 5, implements this. It has to be called for each pair (s_1, s_2) and each successor distribution $s_1 \xrightarrow{\alpha} \mu_1$ of s_1. It takes as input the list of remaining candidates $Sim_{i-1}^{(s_1,\alpha,\mu_1,s_2)}$, the information from the previous iteration (the network $\mathcal{N}(\mu_1, \mu_2, R_{i-1})$, flow f_{i-1}, and distance function d_{i-1}), and the set of edges that have to be deleted from the old network D_{i-1}.

Lemma 5.2.6. *Let $(s_1, s_2) \in R_1$, $\alpha \in Act(s_1)$, and μ_1 such that $s_1 \xrightarrow{\alpha} \mu_1$. Let $Sim_1 = Steps_\alpha(s_2)$. Then* ACTSMF$_{init}$ *returns true iff $\exists \mu_2$ with $s_2 \xrightarrow{\alpha} \mu_2 \wedge \mu_1 \sqsubseteq_{R_1} \mu_2$.*

For some $i > 1$, let Sim_{i-1}, $\mathcal{N}(\mu_1, \mu_2, R_{i-1})$, f_{i-1} and d_{i-1} be as returned by some earlier call to ACTSMF *or* ACTSMF$_{init}$. *Let $D_{i-1} = (R_{i-1} \setminus R_i) \cap (Supp(\mu_1) \times Supp(\mu_2))$ be the set of edges that will be removed from the network during the $(i-1)$th call of* ACTSMF. *Then, the $(i-1)$th call of algorithm* ACTSMF *returns true iff: $\exists \mu_2$ with $s_2 \xrightarrow{\alpha} \mu_2 \wedge \mu_1 \sqsubseteq_{R_i} \mu_2$.*

Proof. Once SMF returns false because the maximum flow for the current candidate μ_2 has value < 1, it will never become a candidate again, as edge deletions cannot lead to increased flow. The correctness proof is then the same as the proof of Lemma 5.2.2. ■

The algorithm $\text{SIMREL}_s^{\text{PA}}$ for deciding strong simulation for PAs is presented as Algorithm 6. During the initialisation (lines 6.1–6.6, intermixed with iteration 1 in

Algorithm 5 Subroutine to calculate whether $s_1 \precsim_{R_i} s_2$, as far as $s_1 \xrightarrow{\alpha} \mu_1$ is concerned. The parameter Sim denotes the subsets of α-successor distributions of s_2 serving as candidates for possible μ_2.

ActSmf$(i, Sim_{i-1}, \mathcal{N}(\mu_1, \mu_2, R_{i-1}), f_{i-1}, d_{i-1}, D_{i-1})$

5.1: $Sim_i \leftarrow Sim_{i-1}$
5.2: $(match, \mathcal{N}(\mu_1, \mu_2, R_i), f_i, d_i) \leftarrow$ Smf$(i, \mathcal{N}(\mu_1, \mu_2, R_{i-1}), f_{i-1}, d_{i-1}, D_{i-1})$
5.3: **if** *match* **then**
5.4: **return** (**true**, $Sim_i, \mathcal{N}(\mu_1, \mu_2, R_i), f_i, d_i$)
5.5: $Sim_i \leftarrow$ **tail**(Sim_i)
5.6: **while** ¬**empty**(Sim_i) **do**
5.7: $\mu_2 \leftarrow$ **head**(Sim_i)
5.8: $(match, \mathcal{N}(\mu_1, \mu_2, R_i), f_i, d_i) \leftarrow$ Smf$_{init}(i, \mu_1, \mu_2, R_i)$
5.9: **if** *match* **then**
5.10: **return** (**true**, $Sim_i, \mathcal{N}(\mu_1, \mu_2, R_i), f_i, d_i$)
5.11: $Sim_i \leftarrow$ **tail**(Sim_i)
5.12: **return** (**false**, $\emptyset$, *NIL*, *NIL*, *NIL*)

ActSmf$_{init}(i, \mu_1, Sim_i, R_i)$

goto line 5.6

lines 6.7–6.9), for $(s_1, s_2) \in R_1$ and $s_1 \xrightarrow{\alpha} \mu_1$, the list $Sim_1^{(s_1,\alpha,\mu_1,s_2)}$ is initialised to $Steps_\alpha(s_2)$ (line 6.6), as no α-successor distribution can be excluded as a candidate a priori. As in SimRel$_s^{\text{FPS}}$, the set **Listener**$_{(s_1,s_2)}$ for (s_1, s_2) is introduced which contains tuples $(u_1, \alpha, \mu_1, u_2, \mu_2)$ such that the network $\mathcal{N}(u_1, \alpha, \mu_1, u_2, \mu_2, R_1)$ contains the edge $(s_1, \overline{s_2})$.

The main iteration of the algorithm starts with generating $D_{i-1}^{(u_1,\alpha,\mu_1,u_2,\mu_2)}$ in lines 6.13–6.17 in a similar way as SimRel$_s^{\text{FPS}}$. Lines 6.19–6.22 check Condition 5.1 by calling ActSmf for each action α and each α-successor distribution μ_1 of s_1. The condition is true if and only if $match_{\alpha,\mu_1}$ is true for all $\alpha \in Act(s_1)$ and $\mu_1 \in Steps_\alpha(s_1)$. In this case we insert the pair (s_1, s_2) into R_{i+1} (line 6.22). We give the correctness of the algorithm:

Theorem 5.2.7 (Correctness for PAs)**.** *When* SimRel$_s^{\text{PA}}(\mathcal{M})$ *terminates, the returned relation equals* $\precsim_{\mathcal{M}}$.

Proof. The proof follows the same lines as the proof of the correctness of the algorithm SimRel$_s^{\text{FPS}}$ in Theorem 5.2.4. The only new element is that we now have to quantify over the actions and successor distributions as prescribed by Definition 4.1.4. This translates to a conjunction in lines 6.8 and 6.21 of the algorithm. Exploiting Lemma 5.2.6 we get the correctness. ■

Now we give the complexity of the algorithm:

Algorithm 6 Algorithm for deciding strong simulation for PAs, where *arc* denotes the associated parameter $(s_1, \alpha, \mu_1, s_2, \mu_2)$.

$\textsc{SimRel}_s^{\mathrm{PA}}(\mathcal{M})$

6.1: $R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \wedge Act(s_1) \subseteq Act(s_2)\}$ and $i \leftarrow 1$
6.2: $R_2 \leftarrow \emptyset$
6.3: **for all** $(s_1, s_2) \in R_1$ **do**
6.4: $\quad$ $\mathbf{Listener}_{(s_1,s_2)} \leftarrow \{(u_1, \alpha, \mu_1, u_2, \mu_2) \mid L(u_1) = L(u_2) \wedge u_1 \xrightarrow{\alpha} \mu_1 \wedge u_2 \xrightarrow{\alpha} \mu_2 \wedge \mu_1(s_1) > 0 \wedge \mu_2(s_2) > 0\}$
6.5: $\quad$ **for all** $\alpha \in Act(s_1), \quad \mu_1 \in Steps_\alpha(s_1)$ **do**
6.6: $\quad\quad$ $Sim_1^{(s_1,\alpha,\mu_1,s_2)} \leftarrow Steps_\alpha(s_2)$
6.7: $\quad\quad$ $(match_{\alpha,\mu_1}, Sim_1^{(s_1,\alpha,\mu_1,s_2)}, \mathcal{N}(s_1, \alpha, \mu_1, s_2, \mu_2, R_1), f_1^{arc}, d_1^{arc})$ $\leftarrow \textsc{ActSmf}_{init}(1, \mu_1, Sim_1^{(s_1,\alpha,\mu_1,s_2)}, R_1)$
6.8: $\quad$ **if** $\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{\mu_1 \in Steps_\alpha(s_1)} match_{\alpha,\mu_1}$ **then**
6.9: $\quad\quad$ $R_2 \leftarrow R_2 \cup \{(s_1, s_2)\}$
6.10: **while** $R_{i+1} \neq R_i$ **do**
6.11: $\quad$ $i \leftarrow i + 1$
6.12: $\quad$ $R_{i+1} \leftarrow \emptyset$ and $D_{i-1} \leftarrow R_{i-1} \setminus R_i$
6.13: $\quad$ **for all** $(s_1, s_2) \in R, \alpha \in Act(s_1), \mu_1 \in Steps_\alpha(s_1), \mu_2 \in Steps_\alpha(s_2)$ **do**
6.14: $\quad\quad$ $D_{i-1}^{(s_1,\alpha,\mu_1,s_2,\mu_2)} \leftarrow \emptyset$
6.15: $\quad$ **for all** $(s_1, s_2) \in D_{i-1}, \quad (u_1, \alpha, \mu_1, u_2, \mu_2) \in \mathbf{Listener}_{(s_1,s_2)}$ **do**
6.16: $\quad\quad$ **if** $(u_1, u_2) \in R_{i-1}$ **then**
6.17: $\quad\quad\quad$ $D_{i-1}^{(u_1,\alpha,\mu_1,u_2,\mu_2)} \leftarrow D_{i-1}^{(u_1,\alpha,\mu_1,u_2,\mu_2)} \cup \{(s_1, s_2)\}$
6.18: $\quad$ **for all** $(s_1, s_2) \in R$ **do**
6.19: $\quad\quad$ **for all** $\alpha \in Act(s_1), \quad \mu_1 \in Steps_\alpha(s_1)$ **do**
6.20: $\quad\quad\quad$ $(match_{\alpha,\mu_1}, Sim_i^{(s_1,\alpha,\mu_1,s_2)}, \mathcal{N}(s_1, \alpha, \mu_1, s_2, \mu_2, R_i), f_i^{arc}, d_i^{arc})$ $\leftarrow \textsc{ActSmf}(i, Sim_{i-1}^{(s_1,\alpha,\mu_1,s_2)}, \mathcal{N}(s_1, \alpha, \mu_1, s_2, \mu_2, R_{i-1}), f_{i-1}^{arc}, d_{i-1}^{arc}, D_{i-1}^{arc})$
6.21: $\quad\quad\quad$ **if** $\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{\mu_1 \in Steps_\alpha(s_1)} match_{\alpha,\mu_1}$ **then**
6.22: $\quad\quad\quad\quad$ $R_{i+1} \leftarrow R_{i+1} \cup \{(s_1, s_2)\}$
6.23: **return** R_i

Theorem 5.2.8 (Complexity for PAs)**.** *The algorithm* $\textsc{SimRel}_s^{\mathrm{PA}}(\mathcal{M})$ *runs in time* $\mathcal{O}(m^2 n)$ *and in space* $\mathcal{O}(m^2)$*. If the fanout of* $\mathcal{M}$ *is bounded by a constant, it has complexity* $\mathcal{O}(n^2)$*, both in time and space.*

Proof. We first consider the space complexity. In the algorithm, we save the sets $D_i^{(s_1,\alpha,\mu_1,s_2,\mu_2)}$, the networks $\mathcal{N}(s_1,\alpha,\mu_1,s_2,\mu_2,R_i)$ which are updated in every iteration, the sets $\mathbf{Listener}_{(s_1,s_2)}$ and the sets $Sim_i^{(s_1,\alpha,\mu_1,s_2)}$. The size of the set $D_i^{(s_1,\alpha,\mu_1,s_2,\mu_2)}$ is in $\mathcal{O}(|\mu_1|\,|\mu_2|)$, which is the maximal number of edges of $\mathcal{N}(s_1,\alpha,\mu_1,s_2,\mu_2,R_i)$. Summing over all $(s_1,\alpha,\mu_1,s_2,\mu_2)$, we get

$$\sum_{s_1\in S}\sum_{\alpha\in Act(s_1)}\sum_{\mu_1\in Steps_\alpha(s_1)}\sum_{s_2\in S}\sum_{\mu_2\in Steps_\alpha(s_2)}|\mu_1|\,|\mu_2| \leq \sum_{\alpha\in Act}\left(\sum_{s_1\in S}\sum_{\mu_1\in Steps_\alpha(s_1)}|\mu_1|\right)\left(\sum_{s_2\in S}\sum_{\mu_2\in Steps_\alpha(s_2)}|\mu_2|\right)\leq m^2 \quad (5.2)$$

Similarly, the memory needed for saving the networks has the same bound $\mathcal{O}(m^2)$. Now we consider the set $\mathbf{Listener}_{(s_1,s_2)}$ for the pair $(s_1,s_2)\in R_1$. Recall that $\mathbf{Listener}_{(s_1,s_2)}$ equals:

$$\{(u_1,\alpha,\mu_1,u_2,\mu_2)\mid L(u_1)=L(u_2)\wedge u_1\xrightarrow{\alpha}\mu_1\wedge u_2\xrightarrow{\alpha}\mu_2 \wedge\mu_1(s_1)>0\wedge\mu_2(s_2)>0\}$$

Let $(u_1,\alpha,\mu_1,u_2,\mu_2)\in\mathbf{Listener}_{(s_1,s_2)}$. Then, it holds that $s_1\in Supp(\mu_1)$ and $s_2\in Supp(\mu_2)$. Hence, the tuple $(u_1,\alpha,\mu_1,u_2,\mu_2)$ can be an element of $\mathbf{Listener}_{(s_1,s_2)}$ of some arbitrary pair (s_1,s_2) at most $|\mu_1|\,|\mu_2|$ times, which corresponds to the maximal number of edges between the set of nodes $Supp(\mu_1)$ and $\overline{Supp(\mu_2)}$ in $\mathcal{N}(s_1,\alpha,\mu_1,s_2,\mu_2,R_1)$. Summing over all $(s_1,\alpha,\mu_1,s_2,\mu_2)$, we get that memory needed for the set **Listener** is also bounded by $\mathcal{O}(m^2)$. For each pair (s_1,s_2) and $s_1\xrightarrow{\alpha}\mu_1$, the set $Sim_1^{(s_1,\alpha,\mu_1,s_2)}$ has size $|Steps_\alpha(s_2)|$. Summing up, this is smaller than or equal to m^2 according to Inequality 5.2. Hence, the overall space complexity is $\mathcal{O}(m^2)$.

Now we consider the time complexity. All initialisations (lines 6.1–6.6 of $\textsc{SimRel}_s^{\mathrm{PA}}$ and the initialisations in $\textsc{ActSmf}_{init}$, which calls $\textsc{Smf}_{init}$) take $\mathcal{O}(m^2)$ time. We observe that a pair (s_1,s_2) belongs to D_i during at most one iteration. Because of the Inequality 5.2, the time needed in lines 6.13–6.17 is in $\mathcal{O}(m^2)$. The rest of the algorithm is dominated by the time needed for calling Smf in line 5.2 of ActSmf. By Lemma 5.2.3, the time complexity for successful and unsuccessful checks concerning the tuple $(s_1,\alpha,\mu_1,s_2,\mu_2)$ is bounded by $\mathcal{O}(g\,|\mu_1|\,|\mu_2|)$. Taking the sum over all possible tuples $(s_1,\alpha,\mu_1,s_2,\mu_2)$ we get the bound gm^2 according to Inequality 5.2. Hence, the complexity is $\mathcal{O}(m^2 n)$. If the fanout g is bounded by a constant, we have $m\leq gn$. Thus, the time complexity is in the order of $\mathcal{O}(n^2)$. In this case the space complexity is also $\mathcal{O}(n^2)$. ■

Remark. For a PA $\mathcal{M}=(S,Act,\mathbf{P},L)$, let $|\mathbf{P}|=\sum_{s\in S}\sum_{\alpha\in Act(s)}|Steps_\alpha(s)|$, called the number of transitions in [9], denote the number of all distributions in $\mathcal{M}$. The algorithm for deciding strong simulation introduced by Baier *et al.* has time complexity

$\mathcal{O}((|\mathbf{P}|\, n^6 + |\mathbf{P}|^2\, n^3)/\log n)$, and space complexity $\mathcal{O}(|\mathbf{P}|^2)$. The number of distributions $|\mathbf{P}|$ and the size of the transitions m are related by $|\mathbf{P}| \leq m \leq n\,|\mathbf{P}|$. The left equality is established if $|\mu| = 1$ for all distributions, and the right equality is established if $|\mu| = n$ for all distributions.

A decision algorithm for strong simulation for CPAs can be adapted from $\textsc{SimRel}_s^{\mathrm{PA}}$ in Algorithm 6 easily: Notations are extended with respect to rate functions instead of distributions in an obvious way. To guarantee the additional rate condition, we rule out successor rate functions of s_2 that violate it by replacing line 6.6 by:

$$Sim_1^{(s_1,\alpha,r_1,s_2)} \leftarrow \{r_2 \in Steps_\alpha(s_2) \mid r_1(S) \leq r_2(S)\}.$$

For each pair (s_1, s_2), and successor rate functions $r_i \in Steps_\alpha(s_i)$ $(i = 1, 2)$, the subroutine for checking whether $r_1 \sqsubseteq_{R_i} r_2$ is then performed in the network $\mathcal{N}(\mu(r_1), \mu(r_2), R_i)$. Obviously, the so obtained algorithm for CPAs has the same complexity $\mathcal{O}(m^2 n)$.

5.3 Strong Probabilistic Simulations

The problem of deciding strong probabilistic simulation for PAs has not been tackled yet. We show that it can be computed by solving LP problems which are decidable in polynomial time [74]. In Subsection 5.3.1, we first present an algorithm for PAs. Thereafter, in Subsection 5.3.2, we extend the algorithm to deal with CPAs.

5.3.1 An Algorithm for PAs

Recall that strong probabilistic simulation is a relaxation of strong simulation in the sense that it allows combined transitions, which are convex combinations of multiple distributions belonging to equally labelled transitions. Again, the most important part is to check whether $s_1 \precsim_R^p s_2$ where R is the current relation. By Definition 4.1.7, it suffices to check $L(s_1) = L(s_2)$ and the condition:

$$\forall \alpha \in Act.\ \forall s_1 \xrightarrow{\alpha} \mu_1.\ \exists s_2 \overset{\alpha}{\rightsquigarrow} \mu_2 \text{ with } \mu_1 \sqsubseteq_R \mu_2 \tag{5.3}$$

However, since the combined transition involves the quantification of the constants $c_i \in [0, 1]$, there are possibly infinitely many such μ_2. Thus, one cannot check $\mu_1 \sqsubseteq_R \mu_2$ for each possible candidate μ_2. The following lemma shows that this condition can be checked by solving LP problems which are decidable in polynomial time [74, 96]. The idea is that we introduce for each α-successor distribution of s_2 a variable, and then reformulates the requirements concerning the combined transitions by linear constraints over these variables. This allows us to construct an LP problem such that the transition $s_1 \xrightarrow{\alpha} \mu$ can be mimicked by an α-combined successor distribution μ' is equivalent to whether the LP problem has a solution.

Lemma 5.3.1. *Let $\mathcal{M} = (S, Act, \mathbf{P}, L)$ be a given PA, and let $R \subseteq S \times S$. Let $(s_1, s_2) \in R$ with $L(s_1) = L(s_2)$ and $Act(s_1) \subseteq Act(s_2)$. Then, $s_1 \precsim_R^p s_2$ iff for each transition*

$s_1 \xrightarrow{\alpha} \mu$, the following LP has a feasible solution:

$$\sum_{i=1}^{k} c_i = 1 \tag{5.4}$$

$$0 \leq c_i \leq 1 \qquad \forall\ i = 1, \ldots, k \tag{5.5}$$

$$0 \leq f_{(s,t)} \leq 1 \qquad \forall (s,t) \in R_\perp \tag{5.6}$$

$$\mu(s) = \sum_{t \in R_\perp(s)} f_{(s,t)} \qquad \forall s \in S_\perp \tag{5.7}$$

$$\sum_{s \in R_\perp^{-1}(t)} f_{(s,t)} = \sum_{i=1}^{k} c_i \mu_i(t) \qquad \forall t \in S_\perp \tag{5.8}$$

where $k = |Steps_\alpha(s_2)| > 0$ and $Steps_\alpha(s_2) = \{\mu_1, \ldots, \mu_k\}$.

Proof. First assume that $s_1 \precsim_R^p s_2$. Let $s_1 \xrightarrow{\alpha} \mu$. By the definition of simulation up to R for strong probabilistic simulation, there exists a combined transition $s_2 \overset{\alpha}{\rightsquigarrow} \mu_c$ with $\mu \sqsubseteq_R \mu_c$. Let $Steps_\alpha(s_2) = \{\mu_1, \ldots, \mu_k\}$ where $k = |Steps_\alpha(s_2)|$. It holds that $k > 0$ as $Act(s_1) \subseteq Act(s_2)$. By definition of combined transition (Definition 4.1.6), there exist constants $c_1, \ldots, c_k \in [0,1]$ with $\sum_{i=1}^k c_i = 1$ such that $\mu_c = \sum_{i=1}^k c_i \mu_i$. Thus Constraints 5.4 and 5.5 hold. Since $\mu \sqsubseteq_R \mu_c$, there exists a weight function $\Delta : S_\perp \times S_\perp \to [0,1]$ for (μ, μ_c) with respect to R. For every pair $(s,t) \in R_\perp$, let $f_{(s,t)} := \Delta(s,t)$. Thus, Constraint 5.6 holds trivially. By Definition 4.1.1 of weight functions, it holds that

(i) $\Delta(s,t) > 0$ implies that $(s,t) \in R_\perp$,

(ii) $\mu(s) = \sum_{t \in S_\perp} \Delta(s,t)$ for $s \in S_\perp$, and

(iii) $\mu_c(t) = \sum_{s \in S_\perp} \Delta(s,t)$ for all $t \in S_\perp$.

Observe that (i) implies that for all $(s,t) \notin R_\perp$, we have that $\Delta(s,t) = 0$. Thus, (ii) and (iii) imply Equations 5.7 and 5.8 respectively.

Now we show the other direction. Let $k = |Steps_\alpha(s_2)|$ and $Steps_\alpha(s_2) = \{\mu_1, \ldots, \mu_k\}$. By assumption, for each $s_1 \xrightarrow{\alpha} \mu$, we have a feasible solution $c_1, \ldots, c_k$ and $f_{(s,t)}$ for all $(s,t) \in R_\perp$ which satisfies all of the constraints. We define $\mu_c = \sum_{i=1}^k c_i \mu_i$. By Definition 4.1.6, μ_c is a combined transition, thus $s_2 \overset{\alpha}{\rightsquigarrow} \mu_c$. It remains to show that $\mu \sqsubseteq_R \mu_c$. We define a function Δ as follows: $\Delta(s,t)$ equals $f_{(s,t)}$ if $(s,t) \in R_\perp$ and 0 otherwise. With the help of Constraints 5.6, 5.7 and 5.8 we have that Δ is a weight function for (μ, μ_c) with respect to R, thus $\mu \sqsubseteq_R \mu_c$. ■

Now we are able to check Condition 5.3 by solving LP problems. For a PA $\mathcal{M} = (S, Act, \mathbf{P}, L)$, and a relation $R \subseteq S \times S$, let $(s_1, s_2) \in R$ with $L(s_1) = L(s_2)$ and $Act(s_1) \subseteq Act(s_2)$. For $s_1 \xrightarrow{\alpha} \mu_1$, we introduce a predicate $LP(s_1, \alpha, \mu, s_2)$ which is true iff the LP problem described as in Lemma 5.3.1 has a solution. Then, $s_1 \precsim_R^p s_2$ iff the conjunction

$$\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{\mu_1 \in Steps_\alpha(s_1)} LP(s_1, \alpha, \mu_1, s_2)$$

Algorithm 7 Algorithm for deciding strong probabilistic simulation for PAs.

$\textsc{SimRel}_s^{\mathrm{PA},p}(\mathcal{M})$

7.1: $R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \wedge Act(s_1) \subseteq Act(s_2)\}$ and $i \leftarrow 0$
7.2: **repeat**
7.3: $\quad i \leftarrow i + 1$
7.4: $\quad R_{i+1} \leftarrow \emptyset$
7.5: $\quad$ **for all** $(s_1, s_2) \in R_i$ **do**
7.6: $\quad\quad$ **for all** $\alpha \in Act(s_1), \quad \mu_1 \in Steps_\alpha(s_1)$ **do**
7.7: $\quad\quad\quad match_{\alpha,\mu_1} \leftarrow LP(s_1, \alpha, \mu_1, s_2)$
7.8: $\quad\quad$ **if** $\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{\mu_1 \in Steps_\alpha(s_1)} match_{\alpha,\mu_1}$ **then**
7.9: $\quad\quad\quad R_{i+1} \leftarrow R_{i+1} \cup \{(s_1, s_2)\}$
7.10: **until** $R_{i+1} = R_i$
7.11: **return** R_i

is true. The algorithm, denoted by $\textsc{SimRel}_s^{\mathrm{PA},p}(\mathcal{M})$, is depicted in Algorithm 7. It takes the skeleton of $\textsc{SimRel}_s(\mathcal{M})$. The key difference is that we incorporate the predicate $LP(s_1, \alpha, \mu_1, s_2)$ in line 7.7. The correctness of the algorithm $\textsc{SimRel}_s^{\mathrm{PA},p}(\mathcal{M})$ can be obtained from the one of $\textsc{SimRel}_s(\mathcal{M})$ together with Lemma 5.3.1. We discuss briefly the complexity. The number of variables in the LP problem in Lemma 5.3.1 is $k + |R|$, and the number of constraints is $1 + k + |R| + 2|S| \in \mathcal{O}(|R|)$. In iteration i of $\textsc{SimRel}_s^{\mathrm{PA},p}(\mathcal{M})$, for $(s_1, s_2) \in R_i$ and $s_1 \xrightarrow{\alpha} \mu_1$, the corresponding LP problem is queried once. The number of iterations is in $\mathcal{O}(n^2)$. Therefore, in the worst case, one has to solve $n^2 \sum_{s \in S} \sum_{\alpha \in Act(s)} \sum_{\mu \in Steps_\alpha(s)} 1 \in \mathcal{O}(n^2 m)$ many such LP problems and each of them has at most $\mathcal{O}(n^2)$ constraints.

5.3.2 An Algorithm for CPAs

Now we discuss how to extend the algorithm to handle CPAs. Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA. Similar to PAs, the most important part is to check the condition $s_1 \precsim_R^p s_2$ for some relation $R \subseteq S \times S$. By Definition 4.1.9, it suffices to check $L(s_1) = L(s_2)$ and the condition:

$$\forall \alpha \in Act.\ \forall s_1 \xrightarrow{\alpha} r_1.\ \exists s_2 \overset{\alpha}{\rightsquigarrow} r_2 \text{ with } \mu(r_1) \sqsubseteq_R \mu(r_2) \wedge r_1(S) \leq r_2(S) \tag{5.9}$$

Recall that for CPAs only successor rate functions with the same exit rate can be combined together. For state $s \in S$, we let $E(s) := \{r(S) \mid s \xrightarrow{\alpha} r\}$ denote the set of all possible exit rates of α-successor rate functions of s. For $E \in E(s)$ and $\alpha \in Act(s)$, we let $Steps_\alpha^E(s) = \{r \in Steps_\alpha(s) \mid r(S) = E\}$ denote the set of α-successor rate functions of s with the same exit rate E. As for PAs, to check the condition $s_1 \precsim_R^p s_2$ we resort to a reduction to LP problems.

Lemma 5.3.2. *Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a given CPA, and let $R \subseteq S \times S$. Let $(s_1, s_2) \in R$ with $L(s_1) = L(s_2)$ and that $Act(s_1) \subseteq Act(s_2)$. Then, $s_1 \precsim_R^p s_2$ iff for each transition $s_1 \xrightarrow{\alpha} r$ either $r(S) = 0$, or there exists $E \in E(s_2)$ with $E \geq r(S)$ such that*

Algorithm 8 Algorithm for deciding strong probabilistic simulation for CPAs.

$\textsc{SimRel}_s^{\text{CPA},p}(\mathcal{M})$

8.1: $R_1 \leftarrow \{(s_1, s_2) \in S \times S \mid L(s_1) = L(s_2) \wedge Act(s_1) \subseteq Act(s_2)\}$ and $i \leftarrow 0$
8.2: **repeat**
8.3: $\quad i \leftarrow i + 1$
8.4: $\quad R_{i+1} \leftarrow \emptyset$
8.5: $\quad$ **for all** $(s_1, s_2) \in R_i$ **do**
8.6: $\quad\quad$ **for all** $\alpha \in Act(s_1), \quad r_1 \in Steps_\alpha(s_1), \quad E \in E(s_2)$ **do**
8.7: $\quad\quad\quad match_{\alpha,r_1,E} \leftarrow LP'(s_1, \alpha, r_1, s_2, E)$
8.8: $\quad\quad$ **if** $\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{r_1 \in Steps_\alpha(s_1)} \bigwedge_{E \in E(s_2)} match_{\alpha,r_1,E}$ **then**
8.9: $\quad\quad\quad R_{i+1} \leftarrow R_{i+1} \cup \{(s_1, s_2)\}$
8.10: **until** $R_{i+1} = R_i$
8.11: **return** R_i

the following LP has a feasible solution, which consists of Constraints 5.4, 5.5, 5.6 of Lemma 5.3.1, and additionally:

$$r(s) = r(S) \sum_{t \in R_\perp(s)} f_{(s,t)} \qquad \forall s \in S_\perp \tag{5.10}$$

$$E \sum_{s \in R_\perp^{-1}(t)} f_{(s,t)} = \sum_{i=1}^{k} c_i r_i(t) \qquad \forall t \in S_\perp \tag{5.11}$$

where $k = \left|Steps_\alpha^E(s)\right|$ *with* $Steps_\alpha^E(s) = \{r_1, \ldots, r_k\}$.

Proof. The proof follows the same strategy as the proof of Lemma 5.3.1, in which the induced distribution of the corresponding rate function should be used. ■

Now we are able to check Condition 5.9 by solving LP problems. For a CPA $\mathcal{M} = (S, Act, \mathbf{R}, L)$, and a relation $R \subseteq S \times S$, let $(s_1, s_2) \in R$ with $L(s_1) = L(s_2)$ and $Act(s_1) \subseteq Act(s_2)$. For $s_1 \xrightarrow{\alpha} r_1$, and $E \in E(s_2)$, we introduce the predicate $LP'(s_1, \alpha, r_1, s_2, E)$ which is true iff $E \geq r_1(S)$ and the corresponding LP problem has a solution. Then, $s_1 \precsim_R^p s_2$ iff the conjunction

$$\bigwedge_{\alpha \in Act(s_1)} \bigwedge_{r_1 \in Steps_\alpha(s_2)} \bigwedge_{E \in E(s_2)} LP'(s_1, \alpha, r_1, s_2, E)$$

is true. The decision algorithm is given in Algorithm 8. As complexity we have to solve $\mathcal{O}(n^2 m)$ LP problems and each of them has at most $\mathcal{O}(n^2)$ constraints.

5.4 Experimental Results

In this section, we provide systematic experimental results of the space and time requirements of the algorithms for Markov chains[2], also comparing several optimisations and heuristics to accelerate the algorithm. As a base algorithm we consider the algorithm without any optimisations (cf. Section 5.2.1). The parametric maximum flow based algorithm (PMF) is treated as one particular optimisation. We also consider the effect of several other optimisations which can be applied selectively. We apply our approach to several case studies taken from PRISM [67]. In order to avoid a bias in the selection of models, we also evaluate the algorithms on randomly generated Markov models. This is inspired by [108] where the authors experimentally evaluated algorithms for classical automata constructions on randomly generated automata. Our experimental approach follows the same strategy. On randomly generated Markov chains, we have two interesting parameters to adjust in our studies: the density of transitions and the density of labels. We study the performance curve for various density combinations.

We first discuss various optimisation strategies in Subsection 5.4.1. In Subsection 5.4.2 different combinations of the optimisations are compared on regular models, uniform random models and non-uniform random models.

5.4.1 Optimisation Options

Beside the PMF optimisation, our implementation of the principal algorithm uses the following optimisations and heuristics to eliminate redundant or trivial computations. All of the optimisations and heuristics presented apply to FPSs and CTMCs directly. Throughout this section, we fix an FPS $\mathcal{M}$ and a pair of states s_1, s_2. Let n denote the number of states and m denote the number of transitions of $\mathcal{M}$. Let $\mathcal{N}(s_1, s_2, R)$ denote the network as defined earlier. Furthermore, let V denote the set of the vertices, and E denote the set of the edges of $\mathcal{N}(s_1, s_2, R)$.

Compact Maximum Flow. The algorithm used to compute the maximum flow is based on the existing push-relabel preflow algorithm [56] and tailored specially to the needs of the decision algorithm in order to save memory and to omit computations for cases that never arise in the scenario considered. In a complete maximum flow implementation, the value of the flow is computed. However, for the purpose at hand, it is sufficient to determine whether or not the maximum flow has value 1. To decide the simulation preorder, we consider bipartite networks in which source and sink and all arcs connected with them are not relevant to the computation and can be omitted. Furthermore, the fact that all remaining (not connected to source or sink) arcs have infinite capacity allows us to ignore the concept of arc capacity altogether.

The use of this tailored algorithm greatly reduces the memory usage (by a factor of approximately 4 to 6) in comparison to a more generic implementation while its runtime stays almost unchanged in most cases. It should be noted that this implementation does

[2]Experimental results for PAs will be discussed in Chapter 7, in which a memory-efficient algorithm is presented for PAs.

not use certain known optimisations for the push-relabel based method and is inferior in speed to implementations which use these optimisations.

State Partitioning. In many large models, many states are structurally identical. This can be exploited by grouping states with identical probabilistic structure together into an equivalence class. This forms a partition of the state space. The equivalence classes are also referred to as blocks. Given two blocks B_1 and B_2 of the partition, simulation algorithm will yield the same result for any pair (s_1, s_2) with $s_1 \in B_1$ and $s_2 \in B_2$. Thus, it suffices to decide simulation once for an arbitrary pair of states picked from B_1 and B_2.

Two states s_1 and s_2 have an identical probabilistic structure if their successors have pairwise the same labelling and the same respective transition probabilities. It is important to note that state partitioning is only correct in the first iteration of the simulation algorithm when the initial relation is defined solely on the basis the labelling, thus is an equivalence relation. As soon as the relation is not an equivalence relation any more, state partitioning can no longer be applied.

State partitioning adds an overhead of $\mathcal{O}(n \log n)$ for sorting states and successor sets. This is necessary for being able to compute the partition and to be able to test whether two states should belong to the same block in linear time with respect to the number of transitions in the model. State partitioning uses an extra $\mathcal{O}(n + h^2)$ space, where h is the number of blocks in the partition. In order to store which block a state belongs to we need $\mathcal{O}(n)$, and in order to store the result of whether one block simulates another block we need $\mathcal{O}(h^2)$. Consider Figure 5.6. The white states can then be partitioned into two blocks: $B = \{s_1, s_2\}$ and $B' = \{t_1, t_2\}$. Obviously, the networks $\mathcal{N}(s_i, t_j, R)$ are initially the same for $i, j \in \{1, 2\}$.

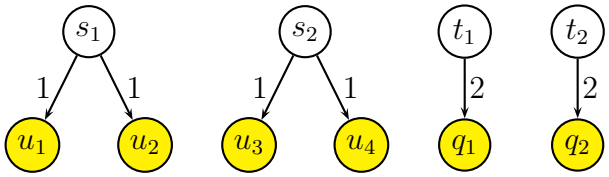

Figure 5.6: A network showing the case in which state partitioning applies.

P-Invariant Checking. The maximum flow of a network $\mathcal{N}(s_1, s_2, R)$ can only have value 1 if the following two constraints are met:

1. $\mu(s) \leq \mu'(R_\perp(s))$ for all $s \in S_\perp$,
2. $\mu'(s') \leq \mu(R_\perp^{-1}(s'))$ for all $s' \in S_\perp$.

The complexity of verifying these constraints is in the order of $\mathcal{O}(|E|)$ per network and $\mathcal{O}(m^2)$ overall. This operation needs an additional $\mathcal{O}(|V|)$ space while performing the checks. As an example we consider the network depicted in Figure 5.7. Obviously the P-Invariant condition is violated: for state q_3 $\mu(q_3) = \frac{3}{4}$ whereas $\mu'(R_\perp(q_3)) = 0$. Observe

further that the state q_1 violates the second condition: $\mu'(q_1) = \frac{7}{8}$ which is greater than $\mu(R_{\perp}^{-1}(q_1)) = \mu(\{\perp, q_2\}) = \frac{1}{4}$. Thus without running the maximum flow algorithm, we can report no and delete the network.

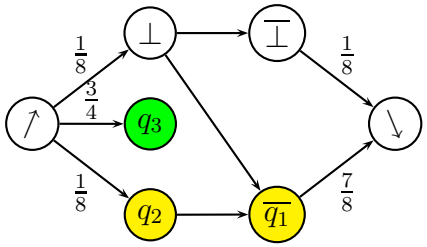

Figure 5.7: A network showing the case in which P-Invariant checking applies.

Significant Arc Detection. The P-Invariant constraints are only checked when a network is created. However, it would be desirable to check whether or not the constraints are still fulfilled after a certain arc has been deleted as a result of its corresponding pair having been removed from the relation. This can be done as follows: For a network which satisfies the P-Invariant constraints, an arc is called *significant* iff its removal would cause the network to violate the constraints. The detection of these arcs takes $\mathcal{O}(|V|^2)$ time in addition to that of P-Invariant checking and $\mathcal{O}(|E|)$ space per network for storing the flag for every arc. Removing an arc takes constant time if the arc is significant, otherwise $\mathcal{O}(|E|)$ time to recompute the significance of the remaining arcs.

Significant arc detection is an extension of PMF. It requires that networks be stored rather than recomputed from scratch, otherwise it is equivalent to P-Invariant checking. In the network depicted in Figure 5.8, the arcs $(q_3, \overline{q_1})$ and $(q_2, \overline{q_1})$ are significant. The arc $(\perp, \overline{\perp})$ is also significant, however, it will be never removed by definition.

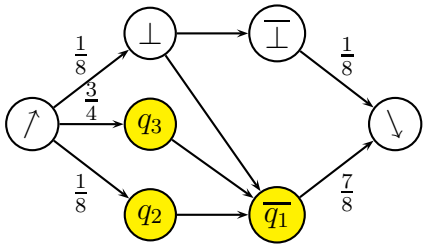

Figure 5.8: A network showing the case in which significant arc detection applies.

5.4.2 Case studies

In the case studies, we refer to the different configurations of optimisations considered in this chapter by binary numbers constituting combinations of the following strategies: State Partitioning (0001), P-Invariant Checking (0010), Significant Arcs (0100), PMF (1000). Reported run-times measure the amount of CPU time (user mode only) spent

Table 5.1: Time used for Leader Election models under various optimisations.

States	439	1031	2007	3463
Trans.	654	1542	3006	5190
Unit	Time (sec)		Time (min)	
0000	6.62001	196.25106	47.409	421.233
0001	0.22081	2.07773	0.234	1.181
0010	0.14801	0.69684	0.049	0.209
0011	0.09101	0.39202	0.026	0.113
1000	6.59761	196.70669	47.632	422.430
1001	0.19201	2.04513	0.235	1.180
1110	0.10681	0.59084	0.043	0.170
1111	*0.06600*	*0.32102*	*0.022*	*0.084*

computing the simulation. Time used on parsing the model prior to simulation and cleaning up memory after simulation is not accounted for. By omitting time spent in system mode, the result is not affected by virtual memory operations. The code was compiled with compiler optimisations turned off to demonstrate the advantage achieved by the heuristics alone. With compiler optimisations turned on, an additional speed-up of up to three times is achieved in some cases. The lowest amount of time/memory is marked in italic print in the tables. All experiments were run on a Linux machine with an AMD Athlon(tm) XP 2600+ processor at 2 GHz equipped with 2GB of RAM.

Leader Election Models. The leader election family of models have a very simple structure, namely that of one state in each model with a large number (denoted by k) of successors while the remaining states have only one successor. As such, these models are a prime example for a successful application of partitioning. Due to the structural similarity of the models, the number of blocks of the state partition is 4 for all leader election models and the number of times that the maximum flow algorithm is actually invoked is drastically decreased. For the simulation of three leaders and $k = 8$ (1031 states, 1542 transitions) with uniform distribution of three different labels, the maximum flow algorithm is invoked 369859 times without any optimisation, and 228109 times with state partitioning.

The time advantage achieved by this becomes apparent in Table 5.1 (0000 vs. 0001). Due to the simplistic structure of the models, PMF yields only a small advantage on the leader election models as recomputing from scratch is not very complex. In general, using the PMF by itself is not desirable for sparse models because the advantage is negligible in comparison to the time and memory overhead. Table 5.1 and Table 5.2 illustrates the additional amount of time and memory required for PMF (1000) versus the approach without any optimisations (0000).

Additionally, maximum flow usage statistic shows that the maximum flow algorithm is invoked more often (although by a relatively small margin) with PMF enabled than not. This is due to the fact that certain trivial networks are discarded during construction without ever computing their maximum flow. However if a network was not initially trivial but becomes trivial after an arc is deleted, this is only detected upon reconstruc-

Table 5.2: Memory (in kB) used for Leader Election models under various optimisations. They represent peak values throughout the process of deciding simulation preorder, excluding memory used by the relation map which is present in all configurations (Map size).

States	439	1031	2007	3463
Trans.	654	1542	3006	5190
Map size	47.158	259.763	983.900	2928.669
0000	*754.500*	*4156.195*	*15742.382*	*46858.687*
0001	754.515	4156.210	15742.398	46858.703
0010	754.500	4156.195	15742.382	46858.687
0011	754.516	4156.211	15742.398	46858.703
1000	3910.007	20711.601	81310.734	266355.210
1001	2589.883	13113.180	53140.984	182841.039
1110	4015.472	21263.984	83497.390	273674.011
1111	2651.290	13412.281	54388.648	187375.586

tion of the same network, but not upon updating and recomputing the network if it was saved. Significant arc detection works against this by effectively performing P-Invariant checking every time an arc is removed from a network.

P-Invariant checking and significant arc detection have little effect in reducing the number of times that the maximum flow algorithm is used on models similar to leader election when used alone. This is due to the fact that almost all states (all except for the first) have exactly one successor and consequently almost all networks have either one arc or none at all. Those with no edges at all are filtered out in advance and those with one edge have $\sum_{s' \in S} P(s, s') = 1$ for both s_1 and s_2 so that P-Invariant checking cannot achieve any additional filtering. The small reduction in maximum flow usage is due to the first state which has more than one successor but is unfortunately negligible.

Time advantage achieved by P-Invariant checking and significant arc detection is exceptionally large compared to the reduction in maximum flow usage. This is because a small number of networks which appear in the leader election models and are filtered out by these optimisations, are inefficient to compute under the maximum flow implementation used in this study. Therefore, the time spent computing maximum flow decreases significantly even though the algorithm is still used almost as much.

Overall, it is notable that the minimum time for leader election is consistently achieved by the configuration 1111. It can be said that in general, the combination of all presented optimisations is beneficial for extremely sparse models such as leader election. If memory usage is a concern, 0011 should be preferred over 1111 as it works without ever storing more than one network in memory at a time (cf. Table 5.2) while only slightly inferior to 1111 in speed.

Molecular Reactions. For CTMCs we consider the Molecular Reactions as a case study. In particular, we focus on the reaction

$$Mg \ + \ 2Cl \ \longleftrightarrow \ Mg^{+2} \ + \ 2Cl^{-}.$$

Table 5.3: Time (in seconds) used for Molecular Reaction models under various optimisations.

States	676	1482	2601	4032	5776
Trans.	2550	5700	10100	15750	22650
0000	0.226	1.159	3.622	9.261	19.840
0001	0.235	1.169	3.751	9.487	20.650
0010	0.204	0.976	3.059	*7.660*	16.960
0011	0.212	1.039	3.375	8.321	18.552
1000	0.227	1.139	3.610	9.039	19.458
1001	0.233	1.182	3.788	9.571	20.386
1110	*0.195*	*0.954*	*3.027*	7.761	*16.754*
1111	0.215	1.078	3.349	8.744	19.107

Table 5.4: Memory (in MB) used for Molecular Reaction models under various optimisations. They represent peak values throughout the process of deciding simulation preorder, excluding memory used by the relation map which is present in all configurations (Map size).

States	676	1482	2601	4032	5776
Trans.	2550	5700	10100	15750	22650
Map size	0.11	0.52	1.61	3.88	7.95
0000	*0.88*	*4.28*	*13.19*	*31.72*	*65.12*
0001	1.33	6.42	19.79	47.60	97.70
0010	0.88	4.28	13.19	31.72	65.12
0011	1.33	6.42	19.79	47.60	97.70
1000	1.09	5.50	17.16	40.99	85.49
1001	1.52	7.44	23.08	55.73	114.53
1110	0.90	4.37	13.53	32.54	66.88
1111	1.46	7.21	22.29	53.92	110.80

Models for other reactions found on the PRISM web-site are very similar in structure and do not offer any additional insight.

While the structure of this family of models is relatively simple, few optimisations show any notable effect. All states have between 1 and 4 successors with the average being around 3.8 for all models, but the transition rates are different between almost all states. As a consequence, state partitioning fails entirely. With a few minor exceptions, all blocks of the partition contain exactly one state, which means that no speed-up can be achieved at all. In particular, the reduction in maximum flow usage is always below 1%.

Although the optimisations are not very effective, in comparison to the leader election models, the algorithm terminates very quickly on this family of models (See also Table 5.1 and Table 5.3): 7 hours for Leader Election with 3463 States and 5190 Transitions (cf. 0000), 9 seconds for Molecular Reaction with 4032 States and 15750 Transitions (cf. 0000). This is because the simulation relation is empty except for the identity relation for all these models which is known after just two iterations of the algorithm.

Table 5.5: Comparison of all optimisations on uniform random models $400, 1, B$ with varying numbers of B. Values are in milliseconds.

B	10	20	30	40	50	60	70	80
0000	7.93	36.60	83.81	140.34	224.68	372.66	650.67	718.48
0001	3.13	28.04	66.64	117.97	185.61	303.72	521.30	573.94
0010	6.90	34.37	81.47	151.68	229.28	395.62	649.67	671.28
0011	*2.77*	*26.43*	*60.64*	97.14	196.08	276.15	473.63	520.97
1000	8.00	34.80	78.97	126.47	195.01	319.29	543.03	612.57
1001	3.17	27.37	64.54	109.37	166.64	272.08	*449.16*	510.20
1010	7.10	34.54	80.57	138.21	211.98	349.39	573.07	637.74
1011	2.77	26.50	61.24	*96.44*	183.28	*268.75*	455.56	*493.56*
1100	9.47	40.30	89.01	137.24	214.05	356.22	601.07	685.98
1101	3.90	31.04	72.24	117.64	181.38	296.05	490.80	555.77
1110	7.37	36.87	84.64	132.37	207.65	344.99	583.04	660.61
1111	2.90	27.47	63.24	99.57	*174.71*	278.78	469.56	509.00

The leader election family on the other hand needs four iterations and does not have a trivial simulation relation, which makes the process of deciding simulation preorder more complex. (Additionally, the leader election family also has some networks for which the maximum flow is hard to compute.) This is also why the memory values are all relatively close to each other (see Table 5.4), specifically the configurations which use PMF (1***). Intuitively this is true because almost every pair is immediately discarded and does not have to be saved for later iterations. This implies that PMF does not hold any benefit for this type of model.

The only optimisation which shows some promise for this type of model is P-Invariant checking (0010). Only surpassed slightly by configuration 1110 in some cases, it has the greatest performance boost of all, although it is relatively small when compared to the approach without any optimisations (0000). While P-Invariant checking consistently reduces maximum flow computation by about 99.2%, the largest part of the run-time is taken up by the remaining set of pairs which are not discarded until the second iteration. Significant arc detection, which builds upon P-Invariant checking and PMF, does not hold any benefit for this model due to the failure of PMF. While faster than pure P-Invariant checking in some cases as a result of the left-over pairs not discarded in the first iteration, the speed-up is not consistent and only in the range of about 1.5% to 5.25%.

Uniform Random Models. In addition to regular case studies, we consider randomly generated DTMCs with uniform distributions, that is, all transitions from a state s have equal probabilities. If not stated explicitly, we also use three different labels which are uniform distributed. Furthermore, these random models can be described by three parameters n, a and b such that $|S| = n$ and $a \leq |post(s)| \leq b \ \forall s \in S$. We will reference random uniform model by the parameters n, a, b. Table 5.5 illustrates required time, memory and number of invocations of the maximum flow algorithm with respect to different model sizes for random uniform models.

This study is particularly remarkable because it demonstrates the strength of PMF. In comparison to other cases studied above, leading configurations in the study at hand use PMF. This is due to the density of the model, i.e. the larger number of successors per state in comparison to the other case studies in this chapter. It is also remarkable that, in contrast to other case studies above, all optimisations hold some (even though limited) benefit.

State partitioning performs well on the lower end of the range, yielding a speed-up of about 80% at best and about 20% at worst. While a larger speed-up may be desirable, this is a very good result since it means that state partitioning will never slow down the process on this kind of model.

P-Invariant checking is beneficial in most cases, particularly towards the upper end of the range, but in a few cases ($40 \leq B \leq 65$) it is actually slower than approach 0000 and it is also slower than state partitioning in general. Consequentially, P-Invariant checking should not be applied on its own. Coupled with state partitioning however (see configuration 0011), P-Invariant checking performs better and is in fact one of the best configurations in the study at hand.

While faster in a few cases, significant arc detection does not yield a consistent performance boost in any configuration. Significant arc detection is most powerful in gradual simulation decision processes where few arcs are deleted in one iteration. The simulation relations in this study however are decided in only three to four iterations, indicating that most pairs of states are deleted from the relation in the first iteration already, but significant arc detection can only speed up the decision on pairs which are not deleted immediately. It stands to reason that significant arc detection would perform better in models with a larger minimum number of successors per state.

PMF shows good results in this study. Clocking in at speeds faster than P-Invariant checking in many cases, this is the kind of model for which PMF is beneficial. At its worst, PMF is about 4% slower than approach 0000. At its best, it is faster by 18%.

The best configuration for this model is a tie between 1001 and 1011. While 1111 sometimes achieves times better than 1001 or 1011, it also requires more memory and has about the same average performance as either 1001 or 1011.

Consider also Figure 5.9 which compares the performances of all configurations[3] on uniform random models with different numbers of labels. All optimisations except state partitioning (0001) and configurations making use of it have monotone falling curves because more labels means that the initial relation will be smaller. Configurations using state partitioning however are affected in a different manner, displaying a very low value at one label, a maximum at two labels and a monotone curve after that. The reason for this behavior is that having only one label works in favor of the partitioning algorithm, enabling it to partition the state space into fewer blocks.

Non-Uniform Random Models. In addition to random uniform models, we also briefly consider randomly generated DTMCs with varying degrees of structure. For this purpose, we define several structural features called biases which loosely represent the probability that a certain feature is present or not. We define the following biases:

[3]To get a readable picture, we plot only the representative configurations, i.e., configurations showing extreme performances. This holds also for Figure 5.10.

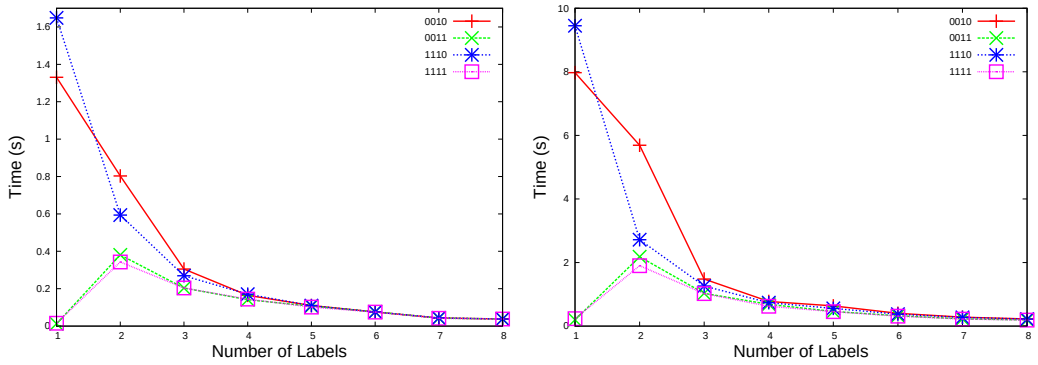

Figure 5.9: Comparison of configurations on random uniform models $200, 1, 25$ (left)and $200, 1, 50$ (right) with respect to varying numbers of labels. Values are averaged over 4 independently generated models of the same class.

- Probability Bias, $pb \in [0; 1]$, defines whether or not the transition probabilities are distributed uniformly ($pb = 0$) or randomly ($pb = 1$)
- Fanout Bias, $fb \in [-1; 1]$, defines if a state is more likely to have the minimum ($fb < 0$) or maximum ($fb > 0$) number of successors

It must be noted that, in case of $pb > 0$, the generated probabilities are not random values. Rather, the partition of the successor set into subsets of successors, each of which have different transition probabilities, is random. This means that the distribution for state s is equal to the distribution for state s' w.r.t. transition probabilities iff $|post(s)| = |post(s')|$ and the successor sets are partitioned into subsets of equal sizes. As a consequence, the state partitioning optimisation is still likely to find useful partitions, even though the same optimisation would be useless for models with truly randomized transition probabilities.

Consider Figure 5.10 (first row) which plots the time needed for $200, 10, 20$ models with different values of probability bias. On the left, we have all configurations which use state partitioning (***1). On the right, we have all remaining configurations. We observe that state partitioning (left) performs best with uniform distributions and gets progressively slower for higher values of the bias. Intuitively this is because the partitioning algorithm is able to create fewer blocks when more distributions are uniform. All other configurations are only insignificantly affected by the bias (right). In these cases, only the complexity of computing the maximum flow depends on the distributions, which accounts for a comparatively small portion of the run-time in models with a low number of successors per state. In both subsets, the configurations using P-Invariant checking (**1*) perform better compared to the remaining configurations for higher values of the bias, because nonuniform distributions are more likely to violate the P-Invariant constraints.

In Figure 5.10 we also compare the impact of different fanout biases on the set of representative configurations. We observe, as one might expect, that a higher fanout bias increases the run-time of the algorithm. An exception to this are configurations which use state partitioning (***1), which are only insignificantly affected by the bias, except for the special case of $fb = 1$. For this value, all states are in the same block and thus

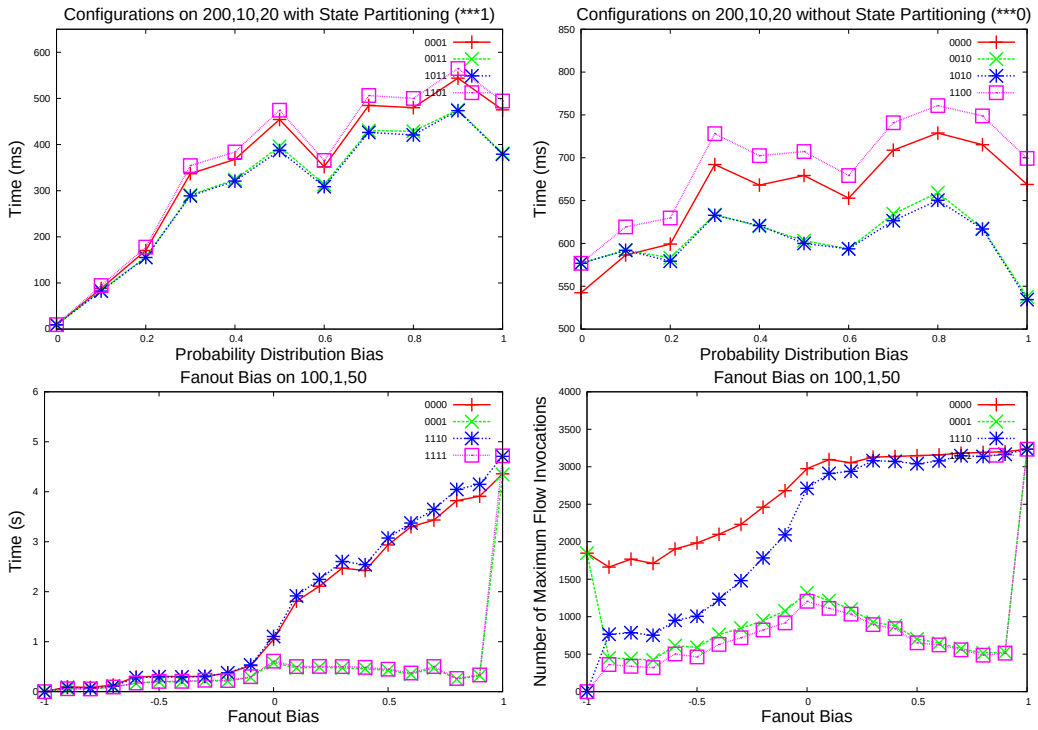

Figure 5.10: Comparison on random nonuniform models with probability bias and fanout bias.

state partitioning cannot improve the run-time. The right plot shows that the increase in run-time is not directly linked to the number of times the maximum flow algorithm is invoked. In particular, the maximum (disregarding corners) for configurations which use state partitioning (***1) is at $fb = 0$, the value which represents the highest entropy and the highest number of blocks. For other configurations (***0), the maximum is reached by $fb > 0$, in which case only a statistically insignificant number of maximum flow computations is trivial. However, the run-time of the algorithm still rises because the complexity of the individual maximum flow computations increases. We conclude that this result depends to a high degree on the complexity of maximum flow computation more than the number of such computations, which means that it will vary greatly for different ranges of numbers of successors.

An Extreme Example. For the examples considered until now, the number of refinement iterations is always very small. The PMF based algorithm reduces the theoretical complexity drastically as the maximum flow computed from last iteration will be reused in the current iteration. Due to the low number of iterations, however, the optimisation PMF does not perform well in practice. We construct now an example in which the number of iterations is in the order of the number of states.

Consider the DTMC depicted in Figure 5.11. The set of states is $S = \{0, 1, \ldots, 2k\}$ where $k \geq 2$. The transition probabilities are as depicted in the figure. We compute the strong simulation $\precsim$ for the case $p = q = 0.5$. Let $R = I \cup \{(2i-1, 2i) \mid i = 1, \ldots, k\}$ where $I = \mathcal{I}(S) \cup \{(0, i) \mid i = 3, 4, \ldots, 2k\}$. First for $(s, t) \in R$ it holds $L(s) = l(t)$. It is easy to verify that R is a simulation relation: the defined R is reflexive implying

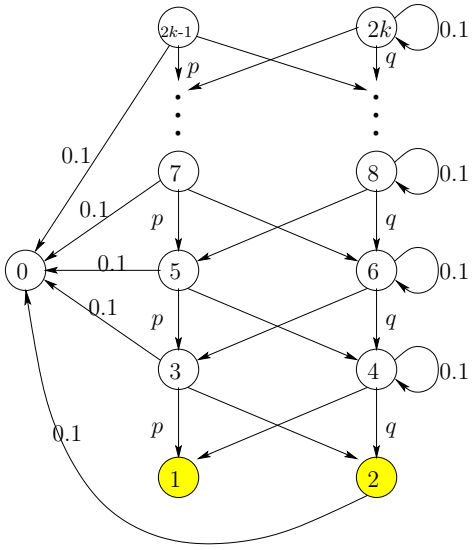

Figure 5.11: A DTMC where the transition probability between $2k-1$ and $2k-2$ is $0.9-p$ for all $k=2,3,\ldots$. The transition probability between $2k$ and $2k-3$ is $0.9-q$ for all $k=2,3,\ldots$.

that $A \subseteq R(A)$. It is routine to verify that for all $(s,s') \in R$ it holds that $\mathbf{P}(s,A) \leq \mathbf{P}(s',R(A))$ for all $A \subseteq \mathit{post}(s)$. Then, by exploiting Lemma 4.2.5 we conclude that R is a strong simulation relation which implies that $R \subseteq \precsim$.

The relation R is indeed the simulation relation, which is returned by our algorithm. In this example the number of iterations of the algorithm is in the order of $\mathcal{O}(k)$. More precisely, the number of iteration needed is $k+1$. Table 5.12 shows the running time for a few configurations. We have following observations:

- In case studies considered until now, the state partitioning approach produces promising results. However in this example, it is almost the same as the configuration 0000. The reason is that state partitioning is only applied in the first iteration, and in this example the number of iterations is $k+1$, thus the time saved comprises only a small fraction.

- The optimisation P-Invariant is even slower in this example. This is due to the fact initially no pairs can be thrown out using this optimisation.

- PMF performs the best in this example. This matches the theoretical analysis. We consider for example the pair $(2k, 2k-1)$. This pair will be thrown out of the relation R at the iteration k. Between iterations 1 and $k-1$, the networks for this pair are always the same, thus the PMF saves time from re-computing maximum flows.

- Similar as the P-Invariant optimisation, the significant arc detection combined with PMF is slower than simply using PMF.

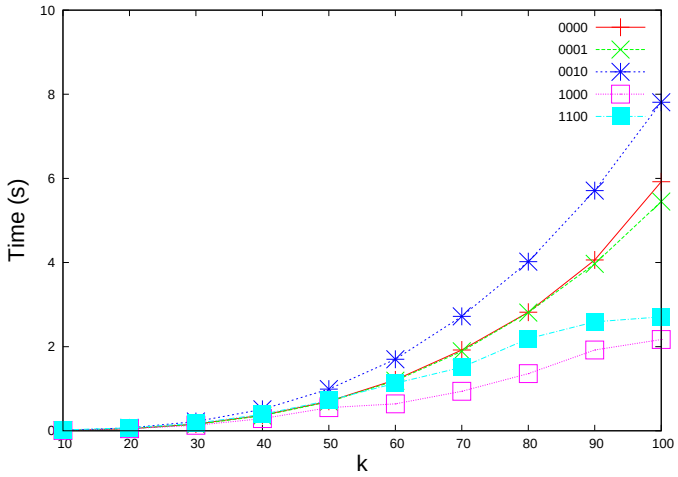

Figure 5.12: The running time for the DTMC in Figure 5.11. For all configurations the number of iterations needed is $k + 1$.

5.5 Bibliographic Notes

The algorithmic problem to minimise a labelled transition system with respect to strong bisimulation is well studied. The most efficient decision algorithm for bisimulation for non-probabilistic models is based on a partition refinement technique, proposed by Kanellakis and Smolka [73], and further improved by Paige and Tarjan [88]. Based on this work, Huynh and Tian [70] presented an $\mathcal{O}(m \log n)$ algorithm for computing strong bisimulation for reactive systems. Derisavi *et. al.* [45] presented an $\mathcal{O}(m \log n)$ algorithm for strong bisimulations for CTMCs, which can be extended easily to handle FPSs or DTMCs. The partition refinement strategy has also been applied to decide bisimulations for MDPs and PAs. In [9], an algorithm with time complexity[4] $\mathcal{O}(mn \log m)$ and space complexity $\mathcal{O}(mn)$ has been proposed for MDPs. Cattani and Segala have presented decision algorithms [30] for strong (probabilistic) bisimulation for PAs. They have shown that the strong probabilistic bisimulation can be obtained by solving $\mathcal{O}(n)$ LP problems.

Orzan and Blom have devised an efficient *distributed* algorithm for bisimulation minimisation, based on the notion of *signatures* [21]. Wimmer et. al. [116] have taken up this idea to arrive at a fully *symbolic* implementation of the signature-refinement approach, to effectively bridge to BDD-based representations of state spaces. Efficient symbolic implementation is further studied in [44, 43, 117]. In [51], an algorithm with $\mathcal{O}(m)$ complexity has been proposed for deciding strong bisimulation on directed acyclic graphs. This is then extended to compute strong bisimulation on acyclic IMCs in [42], with the same complexity $\mathcal{O}(m)$.

Cleaveland, Parrow and Steffen [39] proposed the initial algorithm for computing simulation preorder for non-probabilistic systems. The best algorithm for deciding strong simulation over non-probabilistic systems has been proposed in [64] with time complexity $\mathcal{O}(mn)$. To compute the strong simulation for PAs, Baier *et al.* [9] have presented an

[4] Again, the m used in this paper is slightly different from the m as we use it. A detailed comparison is provided in Remark 5.2.3 of Section 5.2.3.

algorithm which reduces the query whether a state strongly simulates another to a maximum flow problem. Their algorithm has complexity $\mathcal{O}((mn^6 + m^2n^3)/\log n)$ for PAs. For Markov chains, their algorithm has time complexity $\mathcal{O}(n^7/\log n)$ and space complexity $\mathcal{O}(n^2)$. The heavy complexity overhead is due to the complexity of the underlying maximum flow algorithm.

Another related work is the bisimulation algorithm proposed by Cleaveland [37]: In their algorithm, if two state s, t are reported not bisimilar, a Hennessy-Milner [63] styled formula is generated which differentiates between s and t. This idea is extended to finite MDPs in [48] for bisimulations and in [47] for simulations: These algorithms find a formula that witnesses non-bisimilarity or non-similarity of states respectively. Their algorithm has, however, exponential complexity, for both bisimulation and simulation relations.

5.6 Summary

In this chapter, we presented drastically improved algorithms for strong simulation preorder. The core observation is that the networks on which the maximum flows are calculated are very similar across iterations of the refinement loop. We exploit this by adapting the parametric maximum flow algorithm [53] to solve the maximum flows for the arising sequences of similar networks, arriving at an overall time complexity $\mathcal{O}(m^2n)$.

We have investigated experimental approaches to our strong simulation algorithms for Markov models. We experimented with different models to determine ways of further improving upon this algorithm. At the end of this empirical process we have several promising concepts, implemented as optimisations to the fundamental algorithm. Using a collection of well-chosen case studies as well as randomly generated models we studied the practical performance of the concepts.

One of the most interesting observations of our experimental studies is the not uncommon imbalance between theoretical complexity and runtime in practice. While the parametric maximum flow based method offered a tremendous drop in theoretical complexity, its practical implementation comes with an overhead that makes it considerably weaker in many practical applications than more straightforward approaches. Its strength are large, dense models which require more number of iterations. These cases seem seldom in models commonly used for case studies. The gap between theoretical and practical efficiency is not caused by "the constant factors" but by the fact that the corner cases that blow up the worst case complexity are rare in practice.

Surprisingly, more intuitive approaches like state partitioning and P-Invariant checking actually produced promising results in general in our practical studies, in comparison to our theoretically proven algorithm. In particular, state partitioning works very well on models with low to medium transition densities and near-uniform or uniform probability distributions. On the other hand, P-Invariant checking performs well on models with non-uniform probability distributions.

The implementation of the algorithms for deciding strong probabilistic simulations remains as future work.

Chapter 6

Algorithms for Weak Simulations

In this chapter we turn our attention to algorithms to decide weak simulation preorders for Markov chains. As for strong simulations, the algorithm starts with the relation $\{(s, s') \in S \times S' \mid L(s) = L(s')\}$ which is coarser than the weak simulation preorder $\precsim$. Then, the relation R will be refined such that in each iteration pairs (s, s') are removed out of the relation if s' cannot weakly simulates s up to the current relation R. Similar to strong simulation up to a particular relation R, s' weakly simulates s up to R is a relaxation of the weak simulation: it imposes the conditions of weak simulations only on the pair (s, s').

From the definition of weak simulations, s' weakly simulates s up to R implies that the successor states of s and s' are partitioned into U_i and V_i parts for $i = 1, 2$. Recall the intersections $U_i \cap V_i$ are in general not empty for $i = 1, 2$, and these sets are defined via the functions δ_i. For state u, the $\delta_i(u)$ fragment of u belongs to U_i while the $1 - \delta_i(u)$ fragment of u belongs to V_i. Steps to V_i can be viewed as stutter steps which must respect the weak simulation relation: all states in V_2 should weakly simulate s_1 with respect to R, and, analogously, state s_2 should weakly simulate all states in V_1 with respect to R. Steps to U_i are visible steps. For these visible steps, it is required that there exists a weight function for the conditional distributions $(\frac{\mathbf{P}(s_1,\cdot)}{K_1}, \frac{\mathbf{P}(s_2,\cdot)}{K_2})$ with respect to R, where K_i intuitively correspond to the probability of performing a visible step from s_i. If the functions δ_i is known, K_i can be computed directly.

As the above analysis, for fixed characteristic functions δ_i $(i = 1, 2)$, we could again apply maximum flow algorithm to check whether the acquired weight function exists for the conditional distributions with respect to the current relation. Unfortunately, δ_i-functions are not known a priori. We consider a parametric network constructed out of $\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot)$ and R, and then computed a sequence of finite key values, called *breakpoints*, using the parametric maximum flow algorithm. Only these breakpoints need to be considered: For each of the breakpoints, the corresponding weight function for the conditional distributions can be checked, as for strong simulations, via maximum flow algorithms.

Organisation of this Chapter. We first introduce the notion of weak simulation up to R in Section 6.1. We present dedicated algorithms for DTMCs in Section 6.2 and CTMCs in Section 6.3 respectively. In Section 6.4 we provide experimental results, and

Section 6.5 discusses related works. We conclude this chapter in Section 6.6.

6.1 Weak Simulation up to R

The basic algorithm $\textsc{SimRel}_s(\mathcal{M})$ (in Algorithm 2) for strong simulation can be reused for weak simulations: we replace line 2.6 by:

$$\textbf{if } \; s_1 \precsim_R s_2$$

which shall be defined below. We say that s_2 simulates s_1 weakly up to R, denoted by $s_1 \precsim_R s_2$, if, only for this pair, there are functions δ_i, Δ as required by Definition 4.3.1:

Definition 6.1.1. *Let $\mathcal{M} = (S, \mathbf{P}, L)$ be a DTMC. Let $R \subseteq S \times S$ be a relation over S and let $(s_1, s_2) \in R$. We say that s_2 weakly simulates s_1 up to R, written as $s_1 \precsim_R s_2$ if: $L(s_1) = L(s_2)$ and there exist functions $\delta_i : S \to [0,1]$ such that*

1. *(a) $v_1 \; R \; s_2$ for all $v_1 \in V_1$, and (b) $s_1 \; R \; v_2$ for all $v_2 \in V_2$*
2. *there exists a function $\Delta : S \times S \to [0,1]$ such that:*
   - *(a) $\Delta(u_1, u_2) > 0$ implies $u_1 \in U_1, u_2 \in U_2$ and either $u_1 \; R \; u_2$,*
   - *(b) if $K_1 > 0$ and $K_2 > 0$ then for all states $w \in S$:*

     $$K_1 \cdot \Delta(w, U_2) = \mathbf{P}(s_1, w)\delta_1(w), \;\; K_2 \cdot \Delta(U_1, w) = \mathbf{P}(s_2, w)\delta_2(w)$$

     where $K_i = \sum_{u_i \in U_i} \delta_i(u_i) \cdot \mathbf{P}(s_i, u_i)$ for $i = 1, 2$.
3. *for $u_1 \in U_1$ there exists a path fragment $s_2, w_1, \ldots, w_n, u_2$ with positive probability such that $n \geq 0$, $s_1 \; R \; w_j$ for $0 < j \leq n$, and $u_1 \; R \; u_2$.*

Recall that the sets U_i, V_i in the above definition are defined according to Equation 4.4 with respect to the pair (s_1, s_2) and the functions δ_i. If for the pair $(s_1, s_2) \in R$ the conditions in the previous definition do not hold, we write $s_1 \not\precsim_R s_2$. Similar to strong simulation up to R, $s_1 \precsim_R s_2$ does not imply $s_1 \precsim s_2$, since no conditions are imposed on pairs in R different from (s_1, s_2).

Weak simulation up to R for CTMCs, also denoted by $\precsim_R$, is defined similarly.

Example 6.1.1. Consider the DTMC in Figure 6.1, and assume that the relation R is given by $R = \{(s_1, s_3), (u_1, u_2), (v_1, v_2), (s_2, s_4), (s_1, s_4)\}$ (cf. Example 4.14). Now we want to check whether $s_1 \precsim_R s_3$. This relation can be established with:

- $\delta_1(u_1) = \delta_1(v_1) = \delta_1(s_2) = 1$ which implies that $U_1 = \{u_1, v_1, s_2\}$, $K_1 = 1$, and $V_1 = \emptyset$,
- $\delta_2(u_2) = \delta_2(v_2) = 1$, $\delta_2(s_4) = \frac{1}{3}$ which implies that $U_2 = \{u_2, v_2, s_4\}$, $K_2 = 0.5$, and $V_2 = \{s_4\}$,
- $s_1 R s_4$ holds,

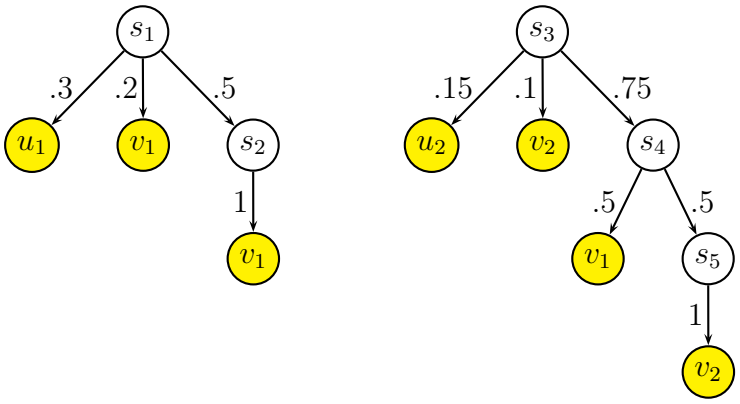

Figure 6.1: A DTMC illustrating the weak simulation up to R.

- the function Δ is defined by: $\Delta(u_1, u_2) = 0.3$, $\Delta(v_1, v_2) = 0.2$ and $\Delta(s_2, s_4) = 0.5$. Similar as Example 4.14, we can show that Δ satisfies the condition 2b of Definition 6.1.1.

Observe that for this relation R only one possible such δ_i exists. The reason is that s_4 is the only state which can be split into visible and stutter parts. Observe that state s_1 goes to $\{u_1, v_1\}$ with probability 0.5, and s_3 goes to $\{u_2, v_2\}$ with probability 0.25. Thus we have to choose $\delta_2(s_4)$ such that the conditional probability of reaching $\{u_2, v_2\}$ must match 0.5 which implies that $\delta_2(s_4) = \frac{1}{3}$. Thus, $s_1 \precsim_R s_3$. Observe also $s_1 \not\precsim_R s_4$ and $s_2 \not\precsim_R s_4$. Another observation is that for any set $Q \subset R$ we have that $s_1 \not\precsim_Q s_3$.

In the above example the δ_i functions are unique. We show that this is in general not the case:

Example 6.1.2. Consider again the DTMC in Figure 6.1, and assume that $R' = \{(s_1, s_3), (u_1, u_2), (v_1, v_2), (s_2, s_4), (s_1, s_4), (s_2, s_3)\}$. Let R denote the relation as in the previous example. Since $R \subseteq R'$, $s_1 \precsim_{R'} s_3$ can be established with the same defined functions δ_i and Δ. For this relation, however, the δ_i functions are not unique any more:

- $\delta_1(u_1) = \delta_1(v_1) = 1$ and $\delta(s_2) = 0$ which implies that $U_1 = \{u_1, v_1\}$ and $V_1 = \{s_2\}$,
- $\delta_2(u_2) = \delta(v_2) = 1$ and $\delta(s_4) = 0$ which implies that $U_2 = \{u_2, v_2\}$ and $V_2 = \{s_4\}$,
- it holds that $(s_1, s_4) \in R'$ and $(s_2, s_3) \in R'$,
- the function Δ is defined by: $\Delta(u_1, u_2) = 0.6$ and $\Delta(v_1, v_2) = 0.4$ and 0 otherwise. It is a routine to show that Δ satisfies the condition the condition 2b of Definition 6.1.1.

An interesting observation with respect to R' is that we could split state s_2 arbitrarily. Assume that $\delta_1(s_2) = x$ where $x \in (0, 1)$. In this case the conditional probability $\mu := \frac{\mathbf{P}(s_1, \cdot)}{K_1}$ is: $\mu(u_1) = \frac{3}{5+10x}$, $\mu(v_1) = \frac{2}{5+10x}$ and $\mu(s_2) = \frac{10x}{5+10x}$. We let $\delta_2(s_4) = \frac{x}{2}$, which implies that the condition probability $\mu' := \frac{\mathbf{P}(s_3, \cdot)}{K_2}$ is the same as μ. Thus the weight function Δ can be defined by: $\Delta(u_1, u_2) = \frac{3}{5+10x}$, $\Delta(v_1, v_2) = \frac{2}{5+10x}$, $\Delta(s_2, s_4) = \frac{10x}{5+10x}$, and 0 otherwise.

As indicated, $s_1 \precsim_R s_2$ does not imply $s_1 \precsim_{\mathcal{M}} s_2$, since no conditions are imposed on pairs in R different from (s_1, s_2). However, by definition, the following lemma holds, for both DTMCs and CTMCs:

Lemma 6.1.1. *Let $R \subseteq S \times S$. Then, R is a weak simulation if and only if for all $s_1 R s_2$ it holds that $s_1 \precsim_R s_2$.*

6.2 An Algorithm for DTMCs

Let $\mathcal{M} = (S, \mathbf{P}, L)$ be a DTMC. Let $R \subseteq S \times S$ be a relation and $s_1\ R\ s_2$. Whether s_2 weakly simulates s_1 up to R is equivalent to whether there exist functions $\delta_i : S \to [0, 1]$ such that the conditions in Definition 4.3.1 are satisfied. Assume that we are given the U_i-characterising functions δ_i. In this case, $s_1 \precsim_R s_2$ can be checked as follows:

- Concerning Condition 1a we check whether for all $v \in S$ with $\delta_1(v) < 1$ it holds that $v\ R\ s_2$. Similarly, for Condition 1b, we check whether for all $v \in S$ with $\delta_2(v) < 1$ it holds that $s_1\ R\ v$.
- The reachability condition can be checked by using standard graph algorithms. In more detail, for each u with $\delta_1(u) > 0$, the condition holds if a state in $R(u)$ is reachable from s_2 via $R(s_1)$ states.
- Finally consider Condition 2. From the given δ_i functions we can compute K_i. In case of that $K_1 > 0$ and $K_2 > 0$, we need to check whether there exists a weight function for the conditional distributions: $\frac{\mathbf{P}(s_1,\cdot)}{K_1}$ and $\frac{\mathbf{P}(s_2,\cdot)}{K_2}$ with respect to the current relation R. From Lemma 4.2.1, this is equivalent to check whether the maximum flow for the network constructed from $(\frac{\mathbf{P}(s_1,\cdot)}{K_1}, \frac{\mathbf{P}(s_2,\cdot)}{K_2})$ and R has value 1.

To check $s_1 \precsim_R s_2$, we want to check whether such δ_i functions exist. The difficulty is that there exist uncountably many possible δ_i functions. In this section, we first show that whether such δ_i exists can be characterised by analysing a parametric network in Subsection 6.2.1. Then, in Subsection 6.2.2, we recall the notion of breakpoints, and show that the breakpoints play a central role in the parametric networks considered: only these points need to be considered. Based on this, we present the algorithm for DTMCs in Subsection 6.2.3. An improvement of the algorithm for certain cases is reported in Subsection 6.2.4.

6.2.1 The Parametric Network $\mathcal{N}(\gamma)$

For $\gamma \in \mathbb{R}_{\geq 0}$, consider the network $\mathcal{N}(\mathbf{P}(s_1, \cdot), \gamma\mathbf{P}(s_2, \cdot), R)$, which is obtained from $\mathcal{N}(\mathbf{P}(s_1, \cdot), \mathbf{P}(s_2, \cdot), R)$ be setting the capacities to the sink $\searrow$ by: $c(\overline{t}, \searrow) = \gamma\mathbf{P}(s_2, t)$. If the states s_1, s_2 and the relation R are clear from the context, we use $\mathcal{N}(\gamma)$ to denote the network $\mathcal{N}(\mathbf{P}(s_1, \cdot), \gamma\mathbf{P}(s_2, \cdot), R)$ for arbitrary $\gamma \in \mathbb{R}_{\geq 0}$.

We introduce some notations. We focus on a particular pair $(s_1, s_2) \in R$, where R is the current relation. We partition the set $post(s_i)$ into MU_i (for: must be in U_i) and PV_i (for: potentially in V_i). The set PV_1 consists of those successors of s_1 which can be either

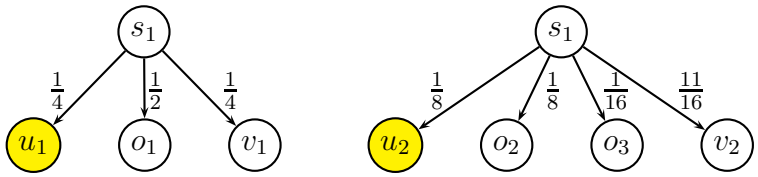

Figure 6.2: A simple DTMC.

put into U_1 or V_1 or both: $PV_1 = post(s_1) \cap R^{-1}(s_2)$. The set MU_1 equals $post(s_1) \backslash PV_1$, which consists of the successor states which can only be placed in U_1. The sets PV_2 and MU_2 are defined similarly by: $PV_2 = post(s_2) \cap R(s_1)$ and $MU_2 = post(s_2) \backslash PV_2$. Obviously, $\delta_i(u) = 1$ for $u \in MU_i$ for $i = 1, 2$.

Example 6.2.1. Consider the DTMC depicted in Figure 6.2, together with the relation $R = \{(s_1, s_2), (s_1, v_2), (v_1, s_2), (u_1, u_2), (o_1, o_2), (o_1, v_2), (v_1, o_3), (v_1, v_2), (o_2, o_1)\}$. By definition, we have $PV_1 = \{v_1\}$ and $PV_2 = \{v_2\}$. Thus, $MU_1 = \{u_1, o_1\}$, $MU_2 = \{u_2, o_2, o_3\}$.

We say a flow function f of $\mathcal{N}(\gamma)$ is valid for $\mathcal{N}(\gamma)$ iff f saturates all edges $(\nearrow, u_1)$ with $u_1 \in MU_1$ and all edges $(\overline{u_2}, \searrow)$ with $u_2 \in MU_2$. If there exists a valid flow f for $\mathcal{N}(\gamma)$, we say that γ is valid for $\mathcal{N}(\gamma)$. The following lemma considers the case in which both s_1 and s_2 have visible steps:

Lemma 6.2.1. *Let s_1 R s_2. Assume that there exists a state $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$, and $s_2' \in post(s_2)$ such that $s_2' \notin R(s_1)$. Then, $s_1 \precsim_R s_2$ iff there exists a valid γ for $\mathcal{N}(\gamma)$.*

Proof. By assumption, we have that $s_i' \in MU_i$ for $i = 1, 2$, thus $MU_i \neq \emptyset$, and it holds that $\delta_i(s_i') = 1$ for $i = 1, 2$.

We first show the *only if* direction. Assume $s_1 \precsim_R s_2$, and let $\delta_i, U_i, V_i, K_i, \Delta$ (for $i = 1, 2$) as described in Definition 4.3.1. Since $MU_i \neq \emptyset$ for $i = 1, 2$, both K_1 and K_2 are greater than 0. We let $\gamma = \frac{K_1}{K_2}$. For $s, s' \in S$, we define the function f for the network $\mathcal{N}(\gamma) = \mathcal{N}(\mathbf{P}(s_1, \cdot), \gamma \mathbf{P}(s_2, \cdot), R)$:

$$f(\nearrow, s) = \mathbf{P}(s_1, s)\delta_1(s), \qquad f(s, \overline{t}) = K_1 \Delta(s, t), \qquad f(\overline{s}, \searrow) = \gamma \mathbf{P}(s_2, s)\delta_2(s)$$

Since $\delta_i(s) \leq 1$ $(i = 1, 2)$ for $s \in S$, $f(\nearrow, s) \leq \mathbf{P}(s_1, s)$ and $f(\overline{s}, \searrow) \leq \gamma \mathbf{P}(s_2, s)$. Therefore, f satisfies the capacity constraints. f also satisfies the conservation rule:

$$f(s, \overline{S}) = K_1 \Delta(s, S) = \mathbf{P}(s_1, s)\delta_1(s) = f(\nearrow, s)$$
$$f(S, \overline{s}) = K_1 \Delta(S, s) = \gamma K_2 \Delta(S, s) = \gamma \mathbf{P}(s_2, s)\delta_2(s) = f(\overline{s}, \searrow)$$

Hence, f is a flow function for $\mathcal{N}(\gamma)$. For $u_1 \in MU_1$, we have $\delta_1(u_1) = 1$, therefore, $f(\nearrow, u_1) = \mathbf{P}(s_1, u_1)$. Analogously, $f(\overline{u_2}, \searrow) = \gamma \mathbf{P}(s_2, u_2)$ for $u_2 \in MU_2$. Hence, f is valid for $\mathcal{N}(\gamma)$, implying that γ is valid for $\mathcal{N}(\gamma)$.

Now we show the *if* direction. Assume that there exists $\gamma > 0$ and a valid flow f for $\mathcal{N}(\gamma)$. The function δ_1 is defined by: $\delta_1(s)$ equals $\frac{f(\nearrow, s)}{\mathbf{P}(s_1, s)}$ if $s \in post(s_1)$ and 0 otherwise.

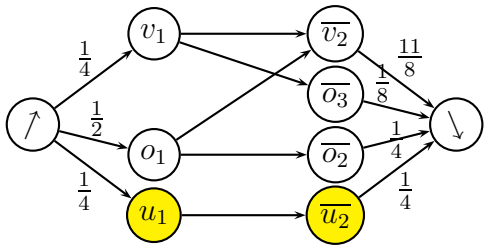

Figure 6.3: The network $\mathcal{N}(2)$ of the DTMC in Figure 6.2.

The function δ_2 is defined similarly: $\delta_2(s)$ equals $\frac{f(\overline{s},\searrow)}{\gamma \mathbf{P}(s_2,s)}$ if $s \in post(s_2)$ and 0 otherwise. Let the sets U_i and V_i be defined as required by Definition 4.3.1. It follows that

$$K_1 = \sum_{s \in U_1} \delta_1(s)\mathbf{P}(s_1,s) = \sum_{s \in U_1} f(\nearrow, s) = f(\nearrow, U_1)$$

$$K_2 = \sum_{s \in U_2} \delta_2(s)\mathbf{P}(s_2,s) = \sum_{s \in U_2} \frac{f(\overline{s},\searrow)}{\gamma} = \frac{f(\overline{U_2},\searrow)}{\gamma}$$

Since the amount of flow out of $\nearrow$ is the same as the amount of flow into $\searrow$, we have $\frac{K_1}{K_2} = \gamma$. Since $\emptyset \neq MU_i \subseteq U_i$ for $i = 1, 2$, both of K_1 and K_2 are greater than 0. We show that the Conditions 1a and 1b of Definition 4.3.1 are satisfied. For $v_1 \in V_1$, we have that $\delta_1(v_1) < 1$ which implies that $f(\nearrow, v_1) < \mathbf{P}(s_1, v_1)$. Since f is valid for $\mathcal{N}(\gamma)$, and since the edge $(\nearrow, v_1)$ is not saturated by f, it must hold that $v_1 \in PV_1$. Therefore, $v_1 \; R \; s_2$. Similarly, we can prove that $s_1 \; R \; v_2$ for $v_2 \in V_2$.

We define $\Delta(w, w') = \frac{f(w,\overline{w'})}{K_1}$ for $w, w' \in S$. Assume that $\Delta(w, w') > 0$. Then, $f(w, \overline{w'}) > 0$, which implies that $(w, \overline{w'})$ is an edge of $\mathcal{N}(\gamma)$, therefore, $(w, w') \in R$. By the flow conservation rule, $f(\nearrow, w) \geq f(w, \overline{w'}) > 0$, implying that $\delta_1(w) > 0$. By the definition of U_1, we obtain that $w \in U_1$. Similarly, we can show that $w' \in U_2$. Hence, the Condition 2a is satisfied. To prove Condition 2b:

$$\Delta(w, U_2) = \sum_{u_2 \in U_2} \frac{f(w,\overline{u_2})}{K_1} = \frac{f(w,\overline{U_2})}{K_1} \overset{(*)}{=} \frac{f(\nearrow, w)}{K_1} = \frac{\delta_1(w)\mathbf{P}(s_1,w)}{K_1}$$

where equality $(*)$ follows from the flow conservation rule. Therefore, for $w \in S$ we have that $K_1\Delta(w, U_2) = \mathbf{P}(s_1, w)\delta_1(w)$. Similarly, we can show that $K_2\Delta(U_1, w) = \mathbf{P}(s_2, w)\delta_2(w)$ holds. Condition 2b is also satisfied. As $K_1 > 0$ and $K_2 > 0$, the reachability condition holds trivially, hence, $s_1 \precsim_R s_2$. ■

Example 6.2.2. Consider the DTMC in Example 6.2.1, together with the relation $R = \{(s_1, s_2), (s_1, v_2), (v_1, s_2), (u_1, u_2), (o_1, o_2), (o_1, v_2), (v_1, o_3), (v_1, v_2), (o_2, o_1)\}$. The network $\mathcal{N}(2)$ is depicted in Figure 6.3. Edges without numbers have capacity ∞. It is easy to see that 2 is valid for $\mathcal{N}(2)$: the corresponding flow sends $\frac{1}{4}$ amount of flow along the path $\nearrow, u_1, \overline{u_2}, \searrow$, $\frac{1}{4}$ amount of flow along the path $\nearrow, o_1, \overline{o_2}, \searrow$, $\frac{1}{4}$ amount of flow along the path $\nearrow, o_1, \overline{v_2}, \searrow$, and $\frac{1}{8}$ amount of flow along the path $\nearrow, v_1, \overline{o_3}, \searrow$.

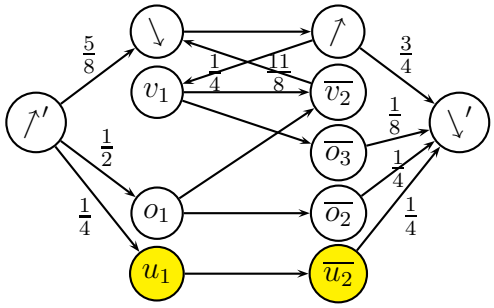

Figure 6.4: The transformed network $\mathcal{N}_t(2)$ for $\mathcal{N}(2)$ in Figure 6.3.

For a fixed $\gamma \in \mathbb{R}_{>0}$, we now address the problem of checking whether there exists a valid flow f for $\mathcal{N}(\gamma)$. This is a feasible flow problem (f has to saturate edges to MU_1 and from MU_2). As we have discussed in Section 3.4, it can be solved by applying a simple transformation to the graph (in time $\mathcal{O}(|MU_1| + |MU_2|)$), solving the maximum flow problem for the transformed graph, and checking whether the flow saturates all edges from the new source.

Example 6.2.3. Consider the network $\mathcal{N}(2)$ in Figure 6.3. Applying the transformation for the feasible flow problem described in Section 3.4, we get the transformed network $\mathcal{N}_t(2)$ depicted in Figure 6.4. It is easy to see that the maximum flow h for $\mathcal{N}_t(2)$ has value $\frac{11}{8}$. Namely: It sends $\frac{1}{4}$ amount of flow along the path $\nearrow', u_1, \overline{u_2}, \searrow'$, $\frac{1}{4}$ amount of flow along the path $\nearrow', o_1, \overline{o_2}, \searrow'$, $\frac{1}{4}$ amount of flow along $\nearrow', o_1, \overline{v_2}, \searrow, \nearrow, \searrow'$, $\frac{1}{8}$ amount of flow along $\nearrow', \searrow, \nearrow, v_1, \overline{o_3}, \searrow'$, and $\frac{1}{2}$ amount of flow along $\nearrow', \searrow, \nearrow, \searrow'$. Thus, it uses all capacities of edges from $\nearrow'$. This implies that 2 is valid for the network $\mathcal{N}(2)$.

6.2.2 Breakpoints

Consider the pair $(s_1, s_2) \in R$. Assume that the conditions of Lemma 6.2.1 are satisfied, thus, to check whether $s_1 \precsim_R s_2$ it is equivalent to check whether a valid γ for $\mathcal{N}(\gamma)$ exits. We show that only a finite possible γ, called breakpoints, need to be considered. The breakpoints can be computed using a variant of the parametric maximum flow algorithm. Then, $s_1 \precsim_R s_2$ if and only if for some breakpoint it holds that the maximum flow for the corresponding transformed network $\mathcal{N}_t(\gamma)$ has a large enough value.

Let $|V|$ denote the number of vertices of $\mathcal{N}(\gamma)$. Let $\kappa(\gamma)$ denote the *minimum cut capacity function* of the parameter γ, which is the capacity of a minimum cut of $\mathcal{N}(\gamma)$ as a function of γ. The capacity of a minimum cut equals the value of a maximum flow. If the edge capacities in the network are linear functions of γ, $\kappa(\gamma)$ is a piecewise-linear concave function with at most $|V| - 2$ breakpoints [53], i. e., points where the slope $\frac{d\kappa}{d\gamma}$ changes. The $|V| - 1$ or fewer line segments forming the graph of $\kappa(\gamma)$ correspond to $|V| - 1$ or fewer distinct minimal cuts. The minimum cut can be chosen as the same on a single linear piece of $\kappa(\gamma)$, and at breakpoints certain edges become saturated or unsaturated. The capacity of a minimum cut for some γ^* gives an equation that contributes a line segment to the function $\kappa(\gamma)$ at $\gamma = \gamma^*$. Moreover, this line segment connects the two

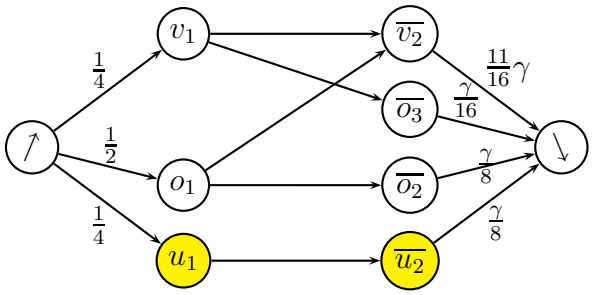

Figure 6.5: The network $\mathcal{N}(\gamma)$ of the DTMC in Figure 6.2 in Example 6.2.1.

points $(\gamma_1, \kappa(\gamma_1))$ and $(\gamma_2, \kappa(\gamma_2))$, where γ_1, γ_2 are the nearest breakpoints to the left and right, respectively.

Example 6.2.4. Consider the DTMC in Figure 6.2, together with the relation $R = \{(s_1, s_2), (s_1, v_2), (v_1, s_2), (u_1, u_2), (o_1, o_2), (o_1, v_2), (v_1, o_3), (v_1, v_2), (o_2, o_1)\}$. The network $\mathcal{N}(\gamma)$ for the pair (s_1, s_2) is depicted in Figure 6.5.

There are two breakpoints, namely $\frac{6}{7}$ and 2. For $\gamma \leq \frac{6}{7}$, all edges leading to the sink can be saturated. This can be established by the following flow function f: sending $\frac{\gamma}{8}$ amount of flow along the path $\nearrow, u_1, \overline{u_2}, \searrow$, $\frac{\gamma}{8}$ amount of flow along the path $\nearrow, o_1, \overline{o_2}, \searrow$, $\frac{11\gamma}{24}$ amount of flow along the path $\nearrow, o_1, \overline{v_2}, \searrow$, $\frac{\gamma}{16}$ amount of flow over $\nearrow, v_1, \overline{o_3}, \searrow$, $\frac{11\gamma}{48}$ amount of flow along the path $\nearrow, v_1, \overline{v_2}, \searrow$. The amount of flow out of node o_1, denoted by $f(o_1, \overline{S})$, is $\frac{7\gamma}{12}$. Given that $\gamma \leq \frac{6}{7}$, we have that $f(o_1, \overline{S}) \leq \frac{1}{2}$. Similarly, consider the amount of flow out of node v_1, which is denoted by $f(v_1, \overline{S})$, is $\frac{7\gamma}{24}$ which implies that $f(v_1, \overline{S}) \leq \frac{1}{4}$. The maximum flow thus has value $|f| = \frac{\gamma}{8} + \frac{\gamma}{8} + \frac{11\gamma}{24} + \frac{\gamma}{16} + \frac{11\gamma}{48} = \gamma$. Thus the value of the maximum flow, or equivalently the value of the minimum cut, for $\gamma \leq \frac{6}{7}$ is $\kappa(\gamma) = \gamma$.

Observe that for $\gamma = \frac{6}{7}$, the edges to v_1 and o_1 are saturated, i.e., we have used full capacities of the edge $(\nearrow, v_1)$ and $(\nearrow, o_1)$. Thus, by a greater value of γ, although the capacities $c(\{\overline{v_2}, \overline{o_2}, \overline{o_3}\}, \searrow)$ increase (become greater than $\frac{3}{4}$), no additional flow can be sent through $\{v_1, o_1\}$. For the other breakpoint 2, we observe that for a value of $\gamma \leq 2$, we can still send $\frac{\gamma}{8}$ through the path $\nearrow, u_1, u_2, \searrow$, but for $\gamma > 2$, the edge to u_1 keeps saturated, thus the amount of flow sent through this path does not increase any more. Thus, for $\gamma \in [\frac{6}{7}, 2]$, the maximum value, or the value of the minimum cut, is $\frac{3}{4} + \frac{\gamma}{8}$. The first term $\frac{3}{4}$ corresponds to the amount of flow through v_1 and o_1. The breakpoint $\frac{6}{7}$ is not valid since the edge to u_1 can not be saturated. As discussed in Example 6.2.3, the breakpoint 2 is valid. The curve for $\kappa(\gamma)$ is depicted in Figure 6.6.

In the following lemma we show that if there is any valid γ, then at least one breakpoint is valid.

Lemma 6.2.2. *Assume $\gamma^* \in (\gamma_1, \gamma_2)$ where γ_1, γ_2 are two subsequent breakpoints of $\kappa(\gamma)$, or $\gamma_1 = 0$ and γ_2 is the first breakpoint, or γ_1 is the last breakpoint and $\gamma_2 = \infty$. Assume γ^* is valid for $\mathcal{N}(\gamma^*)$, then, γ is valid for $\mathcal{N}(\gamma)$ for all $\gamma \in [\gamma_1, \gamma_2]$.*

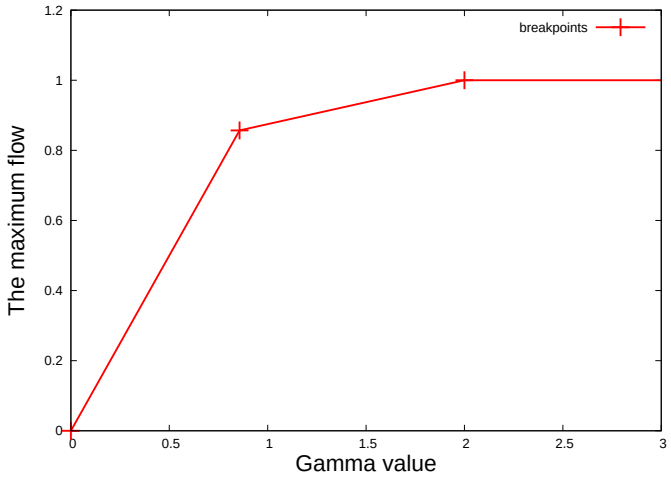

Figure 6.6: The value of the maximum flow, or equivalently the value of the minimum cut, as a function of γ for the network in Figure 6.5.

Proof. Consider the network $\mathcal{N}(\gamma^*)$. Assume that the maximum flow f_{γ^*} is a valid maximum flow for $\mathcal{N}(\gamma^*)$.

Assume first $\gamma' \in (\gamma^*, \gamma_2]$. We use the augmenting path algorithm [2] to obtain a maximum flow f^* in the residual network $\mathcal{N}_{f_{\gamma^*}}(\gamma')$, requiring that the augmenting path contains no cycles, which is a harmless restriction. Then, $f_{\gamma'} := f_{\gamma^*} + f^*$ is a maximum flow in $\mathcal{N}(\gamma')$. Since f_{γ^*} saturates edges from $\nearrow$ to MU_1, $f_{\gamma'}$ saturates edges from $\nearrow$ to MU_1 as well , as flow along an augmenting path without cycles does not un-saturate edges to MU_1. We choose the minimum cut (X, X') for $\mathcal{N}(\gamma^*)$ with respect to f_{γ^*} such that $\overline{MU_2} \cap X' = \emptyset$, or equivalently $\overline{MU_2} \subseteq X$. This is possible since f_{γ^*} saturates all edges $(\overline{u}_2, \searrow)$ with $u_2 \in MU_2$. The minimum cut for $f_{\gamma'}$, then, can also be chosen as (X, X'), as $(\gamma', \kappa(\gamma'))$ lies on the same line segment as $(\gamma^*, \kappa(\gamma^*))$. Hence, $f_{\gamma'}$ saturates the edges from $\overline{MU_2}$ to $\searrow$, which indicates that $f_{\gamma'}$ is valid for $\mathcal{N}(\gamma')$. Therefore, γ' is valid for $\mathcal{N}(\gamma')$ for $\gamma' \in (\gamma^*, \gamma_2]$.

Now let $\gamma' \in [\gamma_1, \gamma^*)$. For the valid maximum flow f_{γ^*} we select the minimal cut (X, X') for $\mathcal{N}(\gamma^*)$ such that $MU_1 \cap X = \emptyset$. Let d denote a valid distance function corresponding to f_{γ^*}. We replace $f_{\gamma^*}(\overline{v}, \searrow)$ by $\min\{f_{\gamma^*}(\overline{v}, \searrow), c_{\gamma'}(\overline{v}, \searrow)\}$ where $c_{\gamma'}$ is the capacity function of $\mathcal{N}(\gamma')$. The modified flow is a preflow for the network $\mathcal{N}(\gamma')$. Moreover, d stays a valid distance function as no new residual edges are introduced. Then, we apply the preflow algorithm to get a maximum flow $f_{\gamma'}$ for the network $\mathcal{N}(\gamma')$. Since no flow is pushed back from the sink, edges from $\overline{MU_2}$ to $\searrow$ are kept saturated. Since $(\gamma^*, \kappa(\gamma^*))$ and $(\gamma', \kappa(\gamma'))$ are on the same line segment, the minimal cut for $f_{\gamma'}$ can also be chosen as (X, X'), which indicates that $f_{\gamma'}$ saturates all edges to MU_1. This implies that γ' is valid for $\mathcal{N}(\gamma')$ for $\gamma' \in [\gamma_1, \gamma^*)$. ■

In Example 6.2.4, only one breakpoint is valid. In the following example we show that it is in general possible that more than one breakpoint is valid.

Example 6.2.5. Consider the network depicted in Figure 6.7. By a similar analysis as Example 6.2.4, we can compute that there are three breakpoints $\frac{1}{2}$, 1 and 2. Assuming

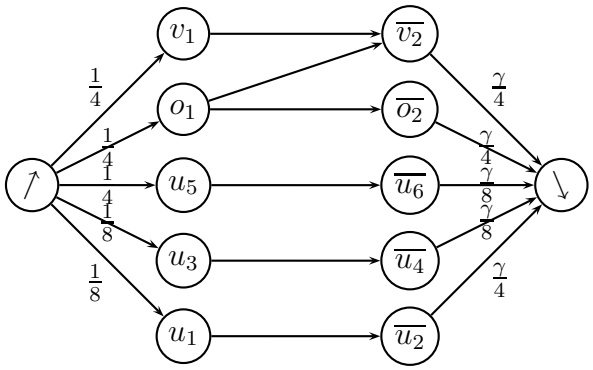

Figure 6.7: A network in which more than one breakpoint is valid.

that $MU_1 = \{o_1\}$ and $MU_2 = \{o_2\}$, we show that all $\gamma \in [\frac{1}{2}, 1]$ are valid. We send $\frac{\gamma}{4}$ amount of flow along the path $\nearrow, o_1, \overline{o_2}, \searrow$, and $\frac{1}{4} - \frac{\gamma}{4}$ amount of flow along the path $\nearrow, o_1, \overline{v_2}, \searrow$. If $\gamma \in [\frac{1}{2}, 1]$, then $0 \leq \frac{1}{4} - \frac{\gamma}{4} \leq \frac{\gamma}{4}$ implying that the flow on edge $(\overline{v_2}, \searrow)$ satisfies the capacity constraints. Obviously this flow is feasible, and all $\gamma \in [\frac{1}{2}, 1]$ are valid for $\mathcal{N}(\gamma)$.

As we would expect now, it is sufficient to consider only the breakpoints of $\mathcal{N}(\gamma)$:

Lemma 6.2.3. *There exists a valid γ for $\mathcal{N}(\gamma)$ iff one of the breakpoints of $\mathcal{N}(\gamma)$ is valid.*

Proof. If there exists a valid γ for $\mathcal{N}(\gamma)$, Lemma 6.2.2 guarantees that one of the breakpoints of $\kappa(\gamma)$ is valid. The other direction is trivial. ■

For a given breakpoint, we need to solve one feasible flow problem to check whether it is valid. In the network $\mathcal{N}(\gamma)$ the capacities of the edges leading to the sink are increasing functions of a real-valued parameter γ. If we reverse $\mathcal{N}(\gamma)$, we get a parametric network that satisfies the conditions in [53]: The capacities emanating from $\nearrow$ are non-decreasing functions of γ. So we can apply the *breakpoint algorithm* [53, p. 40] to obtain all of the breakpoints of $\mathcal{N}(\gamma)$.

6.2.3 The Algorithm

Let $\mathcal{M}$ be a DTMC and let $\textsc{SimRel}_w(\mathcal{M})$ denote the weak simulation algorithm, which is obtained by replacing line 2.6 of $\textsc{SimRel}_s(\mathcal{M})$ in Algorithm 2 by: **if** $s_1 \precsim_R s_2$. The condition $s_1 \precsim_R s_2$ is checked in $\textsc{Ws}(\mathcal{M}, s_1, s_2, R)$, shown as Algorithm 9.

The first part of the algorithm is the preprocessing part. Line 9.1 tests for the case that s_1 could perform only *stutter* steps with respect to the current relation R. If line 9.5 is reached, s_1 has at least one *visible* step, and all successors of s_2 can simulate s_1 up to the current relation R. In this case we need to check the reachability Condition 3 of Definition 4.3.1, which is performed in line 9.5. Recall that $reach(s)$ denotes the set of states that are reachable from s with positive probability. If the algorithm does not

Algorithm 9 Algorithm to check whether $s_1 \precsim_R s_2$.

WS$(\mathcal{M}, s_1, s_2, R)$

9.1: **if** $post(s_1) \subseteq R^{-1}(s_2)$ **then**
9.2: **return true**
9.3: **if** $post(s_2) \subseteq R(s_1)$ **then**
9.4: $U_1 \leftarrow \{s_1' \in post(s_1) \mid s_1' \notin R^{-1}(s_2)\}$
9.5: **return** $(\forall u_1 \in U_1.\ \exists s \in post(reach(s_2) \cap R(s_1)).\ s \in R(u_1))$
9.6: Compute all of the breakpoints $b_1 < b_2 < \ldots < b_j$ of $\mathcal{N}(\gamma)$
9.7: **return** $(\exists i \in \{1, \ldots, j\}.\ b_i$ is valid for $\mathcal{N}(b_i))$

terminate in the preprocessing part, the breakpoints of the network $\mathcal{N}(\gamma)$ are computed. Then, corresponding to Lemma 6.2.3, we check whether one of the breakpoints is valid. We show the correctness of the algorithm WS:

Lemma 6.2.4 (Correctness of WS)**.** *The algorithm* WS$(\mathcal{M}, s_1, s_2, R)$ *returns true iff* $s_1 \precsim_R s_2$.

Proof. We first show the *only if* direction. Assume that WS$(\mathcal{M}, s_1, s_2, R)$ returns true. We consider three possible cases:

- The algorithm returns true at line 9.2. It holds that $post(s_1) \subseteq R^{-1}(s_2)$. We choose $U_1 = \emptyset, V_1 = post(s_1)$, $U_2 = post(s_2)$ and $V_2 = \emptyset$ to fulfill the conditions in Definition 4.3.1. Hence, $s_1 \precsim_R s_2$.

- The algorithm returns true at line 9.5. If the algorithm reaches line 9.5, the following conditions hold: there exists a state $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$ (line 9.1), and $post(s_2) \subseteq R(s_1)$ (line 9.3). Let $U_1 = \{s_1' \in post(s_1) \mid s_1' \notin R^{-1}(s_2)\}$, and define δ_i by: $\delta_1(s) = 1$ if $s \in U_1$, and 0 otherwise, $\delta_2(s) = 0$ for all $s \in S$. By construction, to show $s_1 \precsim_R s_2$ we only need to show the reachability condition. Since the algorithm returns true at line 9.5, it holds that for each $u_1 \in U_1$, there exists $s \in post(reach(s_2) \cap R(s_1))$ such that $s \in R(u_1)$. This is exactly the reachability condition required by weak simulation up to R, thus $s_1 \precsim_R s_2$.

- The algorithm returns true at line 9.7. Thus, there exists breakpoint b_i which is valid for $\mathcal{N}(b_i)$. Then, there exists a state $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$, and $s_2' \in post(s_2)$ such that $s_2' \notin R(s_1)$. By Lemma 6.2.1, we have that $s_1 \precsim_R s_2$.

Now we show the *if* direction. Assume that WS returns false. It is sufficient to show that $s_1 \not\precsim_R s_2$. We consider two cases:

- The algorithm returns false at line 9.5. This implies that there exists a state $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$ (line 9.1), and $post(s_2) \subseteq R(s_1)$ (line 9.3). All states $s_1' \in post(s_1)$ with $s_1' \notin R^{-1}(s_2)$ must be put into U_1. However, since the algorithm returns false at line 9.5, it holds that there exists a state $u_1 \in U_1$, such that there does not exist $s \in post(reach(s_2) \cap R(s_1))$ with $s \in R(u_1)$. Thus the reachability condition of Definition 4.3.1 is violated which implies that $s_1 \not\precsim_R s_2$.

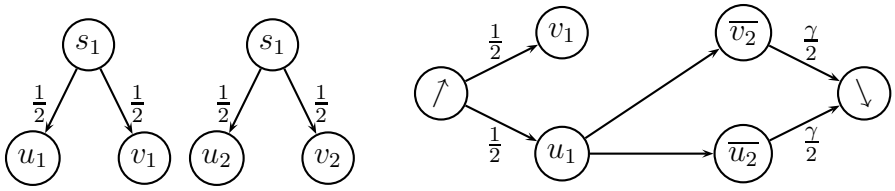

Figure 6.8: A simple DTMC for illustrating that not all maximum flows are valid.

- The algorithm returns false at line 9.7. Then, there exists a state $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$, and $s_2' \in post(s_2)$ such that $s_2' \notin R(s_1)$. Moreover, for all breakpoints b of $\mathcal{N}(\gamma)$, b is not valid for $\mathcal{N}(b)$. By Lemma 6.2.3, there does not exist a valid γ for $\mathcal{N}(\gamma)$. By Lemma 6.2.1, we have that $s_1 \not\precsim_R s_2$. ■

Now we state the correctness of the algorithm SimRel$_w$ for DTMCs:

Theorem 6.2.5 (Correctness of Simrel$_w$)**.** *If* SimRel$_w(\mathcal{M})$ *terminates, the returned relation R equals $\precsim_{\mathcal{M}}$.*

Proof. The proof follows exactly the lines of the proof of Theorem 5.2.1. Let iteration k be the last iteration of Ws. Then, by Lemma 6.2.4, for each pair $(s_1, s_2) \in R_k$, it holds that s_2 weakly simulates s_1 up to R_k, so R_k is a weak simulation. On the other hand, one can prove by induction that each R_i is coarser than $\precsim$. ■

Complexity For $(s_1, s_2) \in R$, we have shown that to check whether $s_1 \precsim_R s_2$ we could first compute the breakpoints of $\mathcal{N}(\gamma)$, then solve $\mathcal{O}(|V|)$ many maximum flow problems. To achieve a better bound, we first prove that applying a binary search method over the breakpoints, we only need to consider $\mathcal{O}(\log |V|)$ breakpoints, and thus solve $\mathcal{O}(\log |V|)$ maximum flow problems.

Assume that the sets MU_i and PV_i for $i = 1, 2$ are constructed as before for $\mathcal{N}(\gamma)$. Recall that a flow function f of $\mathcal{N}(\gamma)$ is valid for $\mathcal{N}(\gamma)$ iff f saturates all edges $(\nearrow, u_1)$ with $u_1 \in MU_1$ and all edges $(\overline{u_2}, \searrow)$ with $u_2 \in MU_2$. If f is also a maximum flow, we say that f is a valid maximum flow of $\mathcal{N}(\gamma)$. We first reformulate Lemma 6.2.1 using maximum flow.

Lemma 6.2.6. *There exists a valid flow f for $\mathcal{N}(\gamma)$ iff there exists a valid maximum flow f_m for $\mathcal{N}(\gamma)$.*

Proof. Assume there exists a valid flow f for $\mathcal{N}(\gamma)$. We let $\mathcal{N}_f(\gamma)$ denote the residual network. We use the augmenting path algorithm to get a maximum flow f' in the residual network $\mathcal{N}_f(\gamma)$. Assume that the augmenting path contains no cycles, which is a harmless restriction. Let $f_m = f + f'$. Obviously, f_m is a maximum flow for $\mathcal{N}(\gamma)$, and it saturates all of the edges saturated by f. Hence, f_m is valid for $\mathcal{N}(\gamma)$. The other direction is simple, since a valid maximum flow is also a valid flow for $\mathcal{N}(\gamma)$. ■

We first discuss how to get a valid maximum flow provided that γ is valid. Observe that even if γ is valid for $\mathcal{N}(\gamma)$, not all maximum flows for $\mathcal{N}(\gamma)$ are necessarily valid. Consider the DTMC on the left part of Figure 6.8. Assume that the relation R is given

by $\{(s_1, s_2), (s_1, v_2), (v_1, s_2), (u_1, u_2), (u_1, v_2)\}$ and consider the pair (s_1, s_2). Thus, we have that $PV_1 = \{v_1\}, MU_1 = \{u_1\}, PV_2 = \{v_2\}, MU_2 = \{u_2\}$. The network $\mathcal{N}(\gamma)$ is depicted on the right part of Figure 6.8. The maximum flow f for $\mathcal{N}(1)$ has value 0.5. If f contains positive sub-flow along the path $\nearrow, u_1, \overline{v_2}, \searrow$, it does not saturate the edge $(\overline{u_2}, \searrow)$. On the contrary, the flow along the single path $\nearrow, u_1, \overline{u_2}, \searrow$ with value 0.5 would be a valid maximum flow.

This example gives us the intuition to use the augmenting path through edges between MU_1 and $\overline{MU_2}$ as much as possible. For this purpose we define a cost function *cost* from edges in $\mathcal{N}(\gamma)$ to real numbers as follows: $cost(u_1, \overline{u_2}) = 2$ for $u_1 \in MU_1$ and $u_2 \in MU_2$, $cost(u_1, \overline{v_2}) = 1$ for $u_1 \in MU_1$ and $v_2 \in PV_2$, $cost(v_1, \overline{u_2}) = 1$ for $v_1 \in PV_1$ and $u_2 \in MU_2$, $cost(s, s') = 0$ otherwise. The costs of edges starting from source, or ending at sink, or in $PV_1 \times \overline{PV_2}$ are 0. The cost of a flow f is defined by $cost(f) = \sum_{e \in E} f(e) cost(e)$. By definition of the cost function, we have the property $cost(f) = f(\nearrow, MU_1) + f(\overline{MU_2}, \searrow)$, i.e., the cost equals the sum of the amount of flow from $\nearrow$ into MU_1 and from $\overline{MU_2}$ into $\searrow$.

Lemma 6.2.7. *Assume that $\gamma > 0$ is valid for $\mathcal{N}(\gamma)$. Let f_γ denote a maximum flow over $\mathcal{N}(\gamma)$ with maximum cost. Then, f_γ is valid for $\mathcal{N}(\gamma)$.*

Proof. By Lemma 6.2.6, provided γ is valid for $\mathcal{N}(\gamma)$, there exists a valid maximum flow function g for $\mathcal{N}(\gamma)$. Since g saturates edges to MU_1 and from MU_2, obviously, $cost(g) = \mathbf{P}(s_1, MU_1) + \gamma \mathbf{P}(s_2, MU_2)$. Assume that f_γ is not valid, which indicates that f_γ does not saturate an edge $(\nearrow, u_1)$ with $u_1 \in MU_1$ or an edge $(\overline{u_2}, \searrow)$ with $u_2 \in \overline{MU_2}$. Then, $cost(f_\gamma) = f(\nearrow, MU_1) + f(\overline{MU_2}, \searrow) < \mathbf{P}(s_1, MU_1) + \gamma \mathbf{P}(s_2, MU_2) = cost(g)$. This contradicts the assumption that f_γ has maximum cost. ■

Let $\mathcal{N}_U(\gamma)$ denote the subnetwork of $\mathcal{N}(\gamma)$ where the set of vertices is restricted to $MU_1, \overline{MU_2}$ and $\{\nearrow, \searrow\}$. The following lemma discusses how to construct a maximum flow with maximum cost.

Lemma 6.2.8. *Assume that f^* is an arbitrary maximum flow of $\mathcal{N}_U(\gamma)$ and $\tilde{f}$ is an arbitrary maximum flow in the residual network $\mathcal{N}_{f^*}(\gamma)$ with the residual edges from MU_1 back to $\nearrow$ removed, as well as the residual edges from $\searrow$ back to $\overline{MU_2}$. Then $f_\gamma = f^* + \tilde{f}$ is a maximum flow over $\mathcal{N}(\gamma)$ with maximum cost.*

Proof. Recall that the cost of f_γ is equal to $cost(f_\gamma) = f_\gamma(\nearrow, MU_1) + f_\gamma(\overline{MU_2}, \searrow)$. Assume that $cost(f_\gamma)$ is not maximal for the sake of contradiction. Let f be a maximum flow such that $cost(f_\gamma) < cost(f)$. Without loss of generality, we assume that $f_\gamma(\nearrow, MU_1) < f(\nearrow, MU_1)$. It holds that $f_\gamma = f^* + \tilde{f}$ where f^* is a maximum flow of $\mathcal{N}_U(\gamma)$, and $\tilde{f}$ is a maximum flow in the residual network $\mathcal{N}_{f^*}(\gamma)$ with the residual edges from MU_1 back to $\nearrow$ removed, as well as the residual edges from $\searrow$ back to $\overline{MU_2}$. On the one hand, f^* sends as much flow as possible along MU_1 in $\mathcal{N}_U(\gamma)$. Since in the residual network $\mathcal{N}_{f^*}(\gamma)$ edges from MU_1 back to $\nearrow$ are removed, this guarantees that no flow can be sent back to $\nearrow$ from MU_1. On the other hand, $\tilde{f}$ sends as much flow as possible from MU_1 to $\overline{PV_2}$ (and also from PV_1 to $\overline{MU_2}$) in $\mathcal{N}_{f^*}(\gamma)$. Thus, $f_\gamma(\nearrow, MU_1)$ must be maximal which contradicts the assumption $f_\gamma(\nearrow, MU_1) < f(\nearrow, MU_1)$. ■

Assume that γ^* is not valid. The following lemma determines, provided a valid γ exists, whether it is greater or smaller than γ^*:

Lemma 6.2.9. *Let $\gamma^* \in [0, \infty)$, and let f be a maximum flow function with maximum cost for $\mathcal{N}(\gamma^*)$, as described in Lemma 6.2.8. Then,*

1. *If f saturates all edges $(\nearrow, u_1)$ with $u_1 \in MU_1$ and $(\overline{u_2}, \searrow)$ with $u_2 \in MU_2$, γ^* is valid for $\mathcal{N}(\gamma^*)$.*
2. *Assume that $\exists u_1 \in MU_1$ such that $(\nearrow, u_1)$ is not saturated by f, and all edges $(\overline{u_2}, \searrow)$ with $u_2 \in MU_2$ are saturated by f. Then, γ^* is not valid. If there exists a valid γ, $\gamma > \gamma^*$.*
3. *Assume that all edges $(\nearrow, u_1)$ with $u_1 \in MU_1$ are saturated by f, and $\exists u_2 \in MU_2$ such that $(\overline{u_2}, \searrow)$ is not saturated by f. Then, γ^* is not valid. If there exists a valid γ, $\gamma < \gamma^*$.*
4. *Assume that $\exists u_1 \in MU_1$ and $\exists u_2 \in MU_2$ such that $(\nearrow, u_1)$ and $(\overline{u_2}, \searrow)$ are not saturated by f. Then, there does not exist a valid γ.*

Proof. 1 : Follows directly from the definition. 2 : In this case, $f(\nearrow, u_1) < \mathbf{P}(s_1, u_1)$ for some $u_1 \in MU_1$. To saturate $(\nearrow, u_1)$, without un-saturating other edges from $\nearrow$ to MU_1, we have to increase the capacities of edges leading to $\searrow$, thus increase γ^*. 3 : Similar to the previous case. 4 : Combining 2 and 3. ■

According to the above lemma, we can use the binary search method over the breakpoints to check whether there exists a valid breakpoint for $\mathcal{N}(\gamma)$. Since there are at most $\mathcal{O}(|V|)$ breakpoints, we invoke the maximum flow algorithm at most $\mathcal{O}(\log |V|)$ times where $|V|$ is the number of vertices of $\mathcal{N}(\gamma)$.

Theorem 6.2.10 (Complexity of $\textsc{SimRel}_w$)**.** *The algorithm $\textsc{SimRel}_w(\mathcal{M})$ runs in time $\mathcal{O}(m^2 n^3)$ and in space $\mathcal{O}(n^2)$. If the fanout g is bounded by a constant, the time complexity is $\mathcal{O}(n^5)$.*

Proof. First, we consider a pair (s_1, s_2) out of the current relation R_i. Look at a single call of $\textsc{Ws}(\mathcal{M}, s_1, s_2, R_i)$. By saving the current relation sets R and R^{-1} in a two dimensional array, the conditions $s \in R(s')$ or $s \in R^{-1}(s')$ can be checked in constant time. Hence, line 9.1 takes time $|post(s_1)|$. To construct the set $reach(s)$ for a state s, BFS can be used, which has complexity $\mathcal{O}(m)$. The size of the set $reach(s)$ is bounded by n. Therefore, we need $\mathcal{O}(|post(s_1)|\, n)$ time at lines 9.3–9.5.

The algorithm computes all breakpoints of $\mathcal{N}(\gamma)$ (with respect to s_1, s_2 and R) using the breakpoint algorithm [53, p. 37–42]. Assume the set of vertices of $\mathcal{N}(\gamma)$ is partitioned into subsets V_1 and V_2 similar to the network described in Section 5.2.1. The number of edges of the network is at most $|E| := |V_1|\,|V_2| - 1$. Let $|V| := |V_1| + |V_2|$, and, without loss of generality, we assume that $|V_1| \leq |V_2|$. For our bipartite networks, the time complexity [53, p. 42] for computing the breakpoints is $\mathcal{O}(|V_1|\,|E| \log(\frac{|V_1|^2}{|E|} + 2))$ which can be simplified to $\mathcal{O}(|V_1|^2\,|V_2|)$. Then, the binary search can be applied over

all of the breakpoints to check whether at least one breakpoint is valid, for which we need to solve at most $\mathcal{O}(\log|V|)$ many maximum flow problems. For our network $\mathcal{N}(\gamma)$, the best known complexity[1] of the maximum flow problem is $\mathcal{O}(|V|^3/\log|V|)$ [31]. As indicated by Lemma 3.3.3, the distance function is bounded by $4|V_1|$ for our bipartite network. Applying this fact in the complexity analysis in [31], we get the corresponding complexity for computing maximum flow for bipartite networks $\mathcal{O}(|V_1||V|^2/\log|V|)$. Hence, the complexity for the $\mathcal{O}(\log|V|)$-invocations of the maximum flow algorithm is bounded by $\mathcal{O}(|V_1||V|^2)$. As $|V| \leq 2|V_2|$, the complexity is equal to $\mathcal{O}(|V_1||V_2|^2)$. Recall $|V_1|$ equals $post(s_1)$, up to a constant of 2. Summing over all (s_1, s_2) over all R_i, we get the overall complexity of $\textsc{SimRel}_w(\mathcal{M})$:

$$\sum_{i=1}^{k} \sum_{(s_1,s_2)\in R_i} (|post(s_1)| + m + |post(s_1)|\, n + |V_1|\,|V_2|^2) \leq 4knm^2 \tag{6.1}$$

Recall that in the algorithm $\textsc{SimRel}_w(\mathcal{M})$, the number of iterations k is at most n^2. Hence, the time complexity is $\mathcal{O}(m^2n^3)$. The space complexity is $\mathcal{O}(n^2)$ because of the representation of R. If the fanout is bounded by a constant g, we have $m \leq gn$, and get the complexity $\mathcal{O}(n^5)$. ■

6.2.4 An Improvement

The algorithm $\textsc{Ws}(\mathcal{M}, s_1, s_2, R)$ is dominated by the part in which all breakpoints ($\mathcal{O}(n)$ many) must be computed, and a binary search is applied to the breakpoints, with $\mathcal{O}(\log n)$ many feasible flow problems. In this section we discuss how to achieve an improved algorithm if the network $\mathcal{N}(\gamma)$ can be partitioned into sub-networks.

Let H denote the sub-relation $R \cap [(post(s_1) \cup \{s_1\}) \times (post(s_2) \cup \{s_2\})]$, which is the local fragment of the relation R. Now let $A_1, A_2, \ldots A_h$ enumerate the classes of the equivalence relation $(H \cup H^{-1})^*$ generated by H, where h denotes the number of classes. W. l. o. g., we assume in the following that A_h is the equivalence class containing s_1 and s_2, i. e., $s_1, s_2 \in A_h$. The following lemma gives some properties of the sets A_i provided that $s_1 \precsim_R s_2$:

Lemma 6.2.11. *For $(s_1, s_2) \in R$, assume that there exists a state $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$, and $s_2' \in post(s_2)$ such that $s_2' \notin R(s_1)$. Let $A_1, \ldots, A_h$ be the sets constructed for (s_1, s_2) as above. If $s_1 \precsim_R s_2$, the following hold:*

1. $\mathbf{P}(s_1, A_i) > 0$ *and* $\mathbf{P}(s_2, A_i) > 0$ *for all* $i < h$,
2. $\gamma_i = \frac{K_1}{K_2}$ *for all* $i < h$ *where* $\gamma_i = \frac{\mathbf{P}(s_1,A_i)}{\mathbf{P}(s_2,A_i)}$

Proof. Since $s_1 \precsim_R s_2$, we let $\delta_i, U_i, V_i, \Delta$ (for $I = 1, 2$) as described in Definition 4.3.1. Because of states s_1' and s_2', we have $K_1 > 0, K_2 > 0$. Let $post_i(s_j) = A_i \cap post(s_j)$ for $i = 1, \ldots, h$ and $j = 1, 2$. We prove the first part. For $i < h$, the set A_i does not contain

[1] For a network $G = (V, E)$ with small $|E|$, there are more efficient algorithms in [32] with complexity $\mathcal{O}(|V|^2\sqrt{|E|})$, and in [78] with complexity $\mathcal{O}(|E|\,|V| + |V|^{2+\epsilon})$ where ϵ is an arbitrary constant. In our bipartite networks, however, $|E|$ is in the order of $|V|^2$. Hence, these complexities become $\mathcal{O}(|V|^3)$.

Algorithm 10 Algorithm to check whether $s_1 \precsim_R s_2$ tailored to DTMCs.

WsImproved$(\mathcal{M}, s_1, s_2, R)$

10.1: Construct the partition $A_1, \ldots, A_h$ (* Assume that $h > 1$ *)
10.2: **for all** $i \leftarrow 1, 2, \ldots h-1$ **do**
10.3: **if** $\mathbf{P}(s_1, A_i) = \mathbf{P}(s_2, A_i) = 0$ **then**
10.4: **raise error**
10.5: **else if** $\mathbf{P}(s_1, A_i) = 0$ or $\mathbf{P}(s_2, A_i) = 0$ **then**
10.6: **return false**
10.7: **else**
10.8: $\gamma_i \leftarrow \frac{\mathbf{P}(s_1, A_i)}{\mathbf{P}(s_2, A_i)}$
10.9: **if** $\gamma_i \neq \gamma_j$ for some $i, j < h$ **then**
10.10: **return false**
10.11: **return** (γ_1 is valid for $\mathcal{N}(\gamma_1)$)

s_1 nor s_2, but only (some of) their successors, so it is impossible that both $\mathbf{P}(s_1, A_i) = 0$ and $\mathbf{P}(s_2, A_i) = 0$. W. l. o. g., assume that $\mathbf{P}(s_1, A_i) > 0$. There exists $t \in post_i(s_1)$ such that $\mathbf{P}(s_1, t) > 0$. Obviously $\delta_1(t) = 1$, thus:

$$0 < \mathbf{P}(s_1, t) = \frac{K_1 \Delta(t, U_2)}{\delta_1(t)} = K_1 \Delta(t, U_2)$$

which implies that $\exists u_2 \in A_i$ with $\Delta(t, u_2) > 0$. Again by the property of the weight function Δ, it holds: $\mathbf{P}(s_2, u_2) = K_2 \Delta(U_1, u_2) \geq K_2 \Delta(t, u_2) > 0$. Now we prove the second part. It holds that:

$$\begin{aligned}
\mathbf{P}(s_1, A_i) &= \sum_{a_i \in A_i} \mathbf{P}(s_1, a_i) = \sum_{a_i \in post_i(s_1)} \mathbf{P}(s_1, a_i) \\
&\overset{(*)}{=} \sum_{a_i \in post_i(s_1)} \frac{K_1 \Delta(a_i, U_2)}{\delta_1(a_i)} \overset{(!)}{=} K_1 \cdot \sum_{a_i \in post_i(s_1)} \Delta(a_i, U_2) \\
&\overset{(\dagger)}{=} K_1 \cdot \sum_{a_i \in post_i(s_1)} \Delta(a_i, A_i) = K_1 \cdot \sum_{a_i \in A_i} \Delta(a_i, A_i)
\end{aligned}$$

where $(*)$ follows from Condition 2b of Definition 4.3.1, (!) follows from the equation $\delta_1(a_i) = 1$ for all $a_i \in post_i(a_1)$ with $i < n$, and $(\dagger)$ follows from the fact that if $a \in post_i(s_1)$, then $\Delta(a, b) = 0$ for $b \in U_2 \backslash post_i(s_2)$. In the same way, we get $\mathbf{P}(s_2, A_i) = K_2 \cdot \sum_{a_i \in A_i} \Delta(A_i, a_i)$. Therefore, $\gamma_i = \frac{K_1}{K_2}$ for $1 \leq i < h$. ■

For the case $h > 1$, the above lemma allows to check whether $s_1 \precsim_R s_2$ efficiently. For this case we replace lines 9.6–9.7 of Ws by the sub-algorithm WsImproved in Algorithm 10. The partition $A_1, \ldots, A_h$ is constructed in line 10.1. Lines 10.2–10.10 follow directly from Lemma 6.2.11: if $\gamma_i \neq \gamma_j$ for some $i, j < h$, we conclude from Lemma 6.2.11 that $s_1 \not\precsim_R s_2$. Line 10.11 follows from the following lemma, which is the counterpart of Lemma 6.2.1:

Lemma 6.2.12. *For $(s_1, s_2) \in R$, assume that there exists a state $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$, and $s_2' \in post(s_2)$ such that $s_2' \notin R(s_1)$. Assume that $h > 1$, and assume* WsImproved$(\mathcal{M}, s_1, s_2, R)$ *reaches line 10.11. Then, $s_1 \precsim_R s_2$ iff γ_1 is valid for $\mathcal{N}(\gamma_1)$.*

Proof. Assume first that $s_1 \precsim_R s_2$. According to Lemma 6.2.1, there exists a valid γ^* for $\mathcal{N}(\gamma^*)$. As in the proof of Lemma 6.2.1, $\gamma^* = \frac{K_1}{K_2}$ is valid for $\mathcal{N}(\gamma^*)$. If Ws reaches line 10.11, by Lemma 6.2.11, we have $\gamma_1 = \frac{K_1}{K_2}$, hence, γ_1 is valid for $\mathcal{N}(\gamma_1)$. The other direction follows directly from Lemma 6.2.1. ■

Example 6.2.6. Consider again the DTMC in Figure 6.2, together with the relation $R = \{(s_1, s_2), (s_1, v_2), (v_1, s_2), (u_1, u_2), (o_1, o_2), (o_1, v_2), (v_1, o_3), (v_1, v_2), (o_2, o_1)\}$. By definition, we have $H = R \backslash \{(o_2, o_1)\}$, $A_1 = \{u_1, u_2\}$, and $A_2 = \{s_1, s_2, v_1, v_2, o_1, o_2, o_3\}$. In this case we have $h = 2$. Recall that $MU_1 = \{u_1, o_1\}$, $MU_2 = \{u_2, o_2, o_3\}$, $PV_1 = \{v_1\}$, $PV_2 = \{v_2\}$. We have $\mathbf{P}(s_1, A_1) = \frac{1}{4}$ and $\mathbf{P}(s_2, A_1) = \frac{1}{8}$. Hence, $\gamma_1 = \frac{\mathbf{P}(s_1, A_1)}{\mathbf{P}(s_2, A_1)} = 2$. As we have shown in Example 6.2.3, 2 is valid for the network $\mathcal{N}(2)$. Hence, $s_1 \precsim_R s_2$.

Assume that $(s_1, s_2) \in R_1$ such that $h > 1$ in the first iteration of SimRel$_w$. We consider the set A_1 and let $\gamma_1 = \frac{\mathbf{P}(s_1, A_1)}{\mathbf{P}(s_2, A_1)}$. If A_1 is not split in the next iteration, γ_1 would not change, and hence, we can reuse the network constructed in the last iteration. Assume that in the next iteration A_1 is split into two sets A_1^a and A_1^b. There are two possible cases:

- either $\frac{\mathbf{P}(s_1, A_1^a)}{\mathbf{P}(s_2, A_1^b)} = \frac{\mathbf{P}(s_1, A_1^a)}{\mathbf{P}(s_2, A_1^b)}$. This implies that both of them are equal to γ_1. If all A_i are split like A_1, we just check whether γ_1 is valid for $\mathcal{N}(\gamma_1)$.
- or $\frac{\mathbf{P}(s_1, A_1^a)}{\mathbf{P}(s_2, A_1^b)} \neq \frac{\mathbf{P}(s_1, A_1^a)}{\mathbf{P}(s_2, A_1^b)}$. This case is simple, we conclude $s_1 \not\precsim_R s_2$ because of Lemma 6.2.11.

This indicates that once in the first iteration γ_1 is determined for (s_1, s_2), either it does not change throughout the iterations, or we conclude that $s_1 \not\precsim_R s_2$ directly. The above analysis can be generalised to the case in which A_1 is split into more than two sets. As the network $\mathcal{N}(\gamma_1)$ is fixed, we can apply an algorithm similar to Smf, which solves the maximum flow problems during all subsequent iterations using only one parametric maximum flow, as for strong simulation.

The above analysis implies that if $h > 1$ for all (s_1, s_2) in the initial R_1, we could even establish the time bound $\mathcal{O}(m^2 n)$, the same as for strong simulation. Since in the worst case it could be the case that $h = 1$ for all $(s_1, s_2) \in R$, the algorithm WsImproved does not improve the worst case complexity.

Since the case that the network cannot be partitioned ($h = 1$) is the one that requires most of our attention, we suggest a heuristic approach that can reduce the number of occurrences of this case. We may choose to run some iterations incompletely (as long as the last iteration is run completely). If iteration i is incomplete, we first check for each pair $(s_1, s_2) \in R_i$ whether the corresponding h_i is greater than 1. If not, we skip the test and just add (s_1, s_2) to R_{i+1}. The intuition is that in the next complete iteration $i' > i$, for each such pair (s_1, s_2) we hope to get $h_{i'} > 1$ because some other elements of R_i have been thrown out. We only perform the expensive computation if in iteration i for every pair $(s_1, s_2) \in R_i$ it holds that $h = 1$.

6.3 An Algorithm for CTMCs

Let $\mathcal{M} = (S, \mathbf{R}, L)$ be a CTMC. We now discuss how to handle CTMCs. Recall that in Definition 4.3.2, we have the rate condition 3′: $K_1\mathbf{R}(s_1, S) \leq K_2\mathbf{R}(s_2, S)$. To determine $\precsim_{\mathcal{M}}$, we simplify the algorithm for DTMCs. If $K_1 > 0$ and $K_2 = 0$, $s_1 \not\precsim_R s_2$ because of the rate condition. Hence, we do not need to check the reachability condition, and lines 9.3–9.5 of the algorithm $\textsc{Ws}(\mathcal{M}, s_1, s_2, R)$ can be skipped. For states s_1, s_2 and relation R, we use $\mathcal{N}(\gamma)$ to denote the network defined in the embedded DTMC $emb(\mathcal{M})$. To check the additional rate condition we use the following lemma:

Lemma 6.3.1. *Let s_1 R s_2. Assume that there exists $s_1' \in post(s_1)$ such that $s_1' \notin R^{-1}(s_2)$. Then, $s_1 \precsim_R s_2$ in $\mathcal{M}$ iff there exists $\gamma \leq \mathbf{R}(s_2, S)/\mathbf{R}(s_1, S)$ such that γ is valid for $\mathcal{N}(\gamma)$.*

Proof. ($\Longrightarrow$): Assume first $s_1 \precsim_R s_2$ in $\mathcal{M}$. Let $\delta_i, U_i, V_i, K_i, \Delta$ (for $i = 1, 2$) as described in Definition 4.3.2. Obviously, s_1' must be in U_1, implying that $K_1 > 0$. Because of the rate condition it holds that $K_1\mathbf{R}(s_1, S) \leq K_2\mathbf{R}(s_2, S)$, which implies that $K_2 > 0$. It is sufficient to show that $\gamma := K_1/K_2$ is valid for $\mathcal{N}(\gamma)$. Exactly as in the proof of Lemma 6.2.1 (the *only if* direction), we can construct a valid flow f for $\mathcal{N}(\gamma)$. Thus, γ is valid for $\mathcal{N}(\gamma)$ and $\gamma \leq \mathbf{R}(s_2, S)/\mathbf{R}(s_1, S)$.

($\Longleftarrow$): By assumption, γ is valid for $\mathcal{N}(\gamma)$. We may assume that there exists a valid flow function f for $\mathcal{N}(\gamma)$. We define $\delta_i, V_i, U_i, K_i, \Delta$ (for $i = 1, 2$) as in the proof (the *if* direction) of Lemma 6.2.1. Recall that s_1' must be in U_1, implying that $f(\nearrow, s_1') > 0$. Thus, there must be a node $\overline{s}$ in $\mathcal{N}(\gamma)$ with $f(\overline{s}, \searrow) > 0$, which implies that $s \in U_2$. Thus we have $K_2 > 0$. Using the proof (the *if* direction) of Lemma 6.2.1, it holds that $s_1 \precsim_R s_2$ in $emb(\mathcal{M})$, moreover, it holds that $\gamma = K_1/K_2$. By assumption it holds that $\gamma \leq \mathbf{R}(s_2, S)/\mathbf{R}(s_1, S)$ which is exactly the rate condition. ■

To check the rate condition for the case $h > 1$, we replace line 10.11 of the algorithm WsImproved by:

$$\textbf{return } (\gamma_1 \leq \gamma^* \wedge \gamma_1 \text{ is valid for } \mathcal{N}(\gamma_1))$$

where $\gamma^* = \mathbf{R}(s_2, S)/\mathbf{R}(s_1, S)$ can be computed directly. In case $h = 1$, we replace line 9.7 of Ws by:

$$\textbf{return } (\exists i \in \{1, \ldots, j\}.\ b_i \leq \gamma^* \wedge b_i \text{ is valid for } \mathcal{N}(b_i))$$

to check the rate condition. Or, equivalently, we can check whether the minimal valid breakpoint γ_m is smaller than or equal to γ^*. The binary search algorithm introduced for DTMCs can also be modified slightly to find the minimal valid breakpoint. The idea is that, if we find a valid breakpoint, we first save it, and then continue the binary search on the left side. If another breakpoint is valid, we save the smaller one. As the check for the reachability condition disappears for CTMCs, we get even a better bound for sparse CTMCs:

Theorem 6.3.2. *If the fanout g of $\mathcal{M}$ is bounded by a constant, the time complexity for CTMC is $\mathcal{O}(n^4)$.*

Proof. In the proof of Theorem 6.2.10 we have shown that the algorithm Ws has complexity $\mathcal{O}(|V_1|\,|V_2|^2)$. As we do not check the reachability condition for CTMCs, the overall complexity of $\textsc{SimRel}_w(\mathcal{M})$ reduces to: $\sum_{s_1\in S}\sum_{s_2\in S}\sum_{i=1}^{l}\left(|post(s_1)|+|V_1|\,|V_2|^2\right)$ (see Inequality 6.1), which is bounded by $2kgm^2$. Since k is bounded by n^2, the time complexity is bounded by $4gm^2n^2$. If g is a constant, we have $m \leq gn$, hence, the time complexity is $4g^3n^4 \in \mathcal{O}(n^4)$. ■

6.4 Experimental Results

Consider the DTMC in Figure 5.11. For this DTMC, our algorithm provides the weak simulation relation $R = \precsim \cup \{(2i-1, 2i+2) \mid i = 2,\ldots,k-1\} \cup \{(i, 2j) \mid j = 2,\ldots,k-1 \wedge i > 2j\}$. We first discuss that R is indeed a weak simulation. Recall the strong simulation is $\prec = I \cup \{(2i-1, 2i) \mid i = 1,\ldots,k\}$ where $I = \mathcal{I}(S) \cup \{(0, i) \mid i = 3, 4,\ldots, 2k\}$. Since strong simulation is finer than then weak simulation, it holds that $\prec \subseteq \precsim$. Thus, for all pair (s_1, s_2) with $s_1 \prec s_2$ the conditions with respect to weak simulation up to R (cf. 6.1.1) are satisfied. The other pairs in R also satisfy condition of Definition 6.1.1:

- Consider the pair $(2i-1, 2i+2)$ with $i = 2,\ldots,k-1$. We define the functions δ_i as follows: $\delta_1(0) = \delta_1(2i-3) = \delta_1(2i-2) = 1$, and $\delta_2(2i+2) = \delta_2(2i-3) = \delta_2(2i-2) = 0$. Thus, $V_1 = U_2 = \emptyset$. Obviously, it holds that $(2i-1, v) \in R$ for $v \in V_2$. The reachability condition can be checked directly.
- Now consider the pair $(i, 2j)$ where $j = 2,\ldots,k-1$ and $i > 2j$. For this pair, we define δ_i such that $U_1 = V_2 = \emptyset$. It is easy to check that for each $s \in V_1$, it holds that $(s, 2j) \in R$.

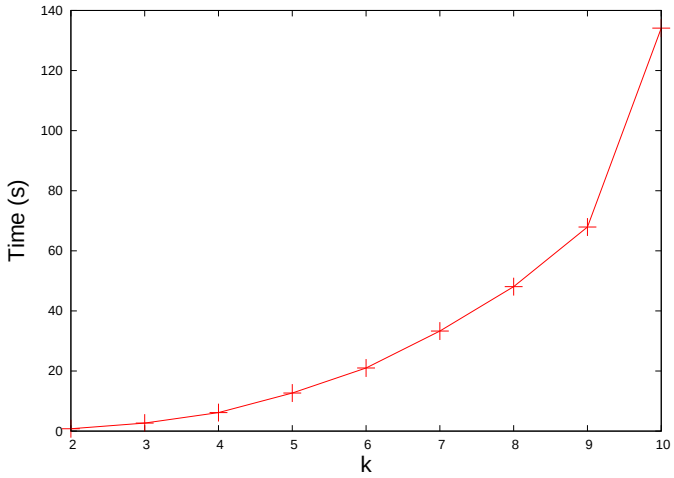

Figure 6.9: The running time for the DTMC in Figure 5.11.

Figure 5.11 plots the time needed to compute the weak simulation for several k. Recall for strong simulations the number of iterations needed is $k+1$. For weak simulation, the iterations needed are $2k+1$. As we observe from the figure, even for small number of $k = 10$, more than two minutes are needed, whereas for strong simulation only 0.01 second is needed. Indeed, the performance is too high for it to be of practical value.

6.5 Bibliographic Notes

In the non-probabilistic models, the computation of weak and branching bisimulation is theoretically dominated by the need to compute the transitive closure of internal transitions. This alone has complexity $\mathcal{O}(n^3)$ (disregarding some very specialized algorithms for transitive closure such as [40]). Based on the transitive closure, a partition refinement based approach [88] can then be used to achieve an algorithm with complexity $\mathcal{O}(n^3 + m^* \log n)$, where m^* denotes the size of the transitive closure of the model. Since m^* is bounded by n^2, the above complexity is in $\mathcal{O}(n^3)$.

In an action-labelled variant of DTMCs, weak bisimulation [12] is introduced to abstract internal behaviour. Because of the probabilistic branching, the induced weak transition relation is in general infinite, implying the closure based method can not be applied. Baier and Hermanns [12] exploited the fact that weak bisimulation and branching bisimulation coincide for this model, and devised a modification of the partition refinement technique to arrived at an $\mathcal{O}(n^3)$ algorithm for deciding such weak bisimulations. Weak simulation is decidable in polynomial time [13] by reducing it to a linear programming problems. In the context of PAs, decision algorithms with exponential complexity have been proposed for weak bisimulations in [30]. For alternating models (which is a subclass of PAs), polynomial algorithms have been studied in [92]. Polynomial algorithms have also been proposed for delay bisimulation and simulation relations [19, 107] (which are coarser than strong bisimulation and simulation relations respectively, and finer than weak bisimulation and simulation relations respectively).

Also closely related is the decision algorithm for weak bisimulation for interactive Markov chains (IMCs) [65]. As for non-probabilistic models, the complexity is $\mathcal{O}(m^* \log n + n^3)$ where the transitive closure is bounded by $m^* \leq n^2$. A dedicated algorithm for weak bisimulation for acyclic IMCs is studied in [42]. Even though the worst case complexity is still cubic in the number of states, as noted in [58], the transitive closure computation does *not* dominate in practical applications (then the above complexity becomes $\mathcal{O}(m^* \log n)$).

6.6 Summary

In this chapter we have developed algorithms for deciding weak simulation for DTMCs and CTMCs respectively. Since the membership of a state can be split arbitrarily into visible and invisible parts, maximum flow based method is not directly applicable. We considered a parametric network, and then computed a sequence of finite key values, called breakpoints, using the parametric maximum flow algorithm. Only these breakpoints need to be considered: For each of the breakpoints, the corresponding weight function for the conditional distributions can be checked, as for strong simulations, via maximum flow algorithms. We arrived at an algorithm with time complexity $\mathcal{O}(m^2 n^3)$ and space complexity $\mathcal{O}(n^2)$.

Chapter 7

Simulation Based Minimisation

If a simulation preorder $\precsim$ is computed, the corresponding simulation equivalence $\precsim \cap \succsim$ can be obtained. Then, the simulation quotient automaton can be constructed from it. For strong simulation, the quotient automaton can be constructed with complexity $\mathcal{O}(m^2 n)$ for all models considered. For DTMCs and CTMCs, it is shown [18] that strong simulation equivalence and strong bisimulation coincide. Thus, the quotient automaton induced by strong simulation equivalence is the same as the one induced by strong bisimulation, which can be determined with complexity [45] $\mathcal{O}(m \log n)$. For PAs and CPAs, strong simulation equivalence is strictly coarser than the strong bisimulation equivalence. Since it is coarser, the induced quotient automaton is potentially smaller than strong bisimulation quotient automaton. This is not surprising, as PAs subsume labelled transition systems, in which strong simulation equivalence is strictly coarser than strong bisimulation. In this chapter, we propose an efficient partition refinement based algorithm for computing simulation quotient automata for PAs and CPAs.

We first discuss the smallest quotient automaton induced by strong simulation preorder for probabilistic automaton. Then, we discuss how to incorporate the partition refinement scheme into the algorithm for deciding strong simulation preorder for PAs. As in the non-probabilistic setting, we first show that strong simulation relations can also be characterised by partition pairs, thus the problem can be reduced to generalised coarsest partition problems (GCPPs). Since in PAs, states have in general non-trivial distributions instead of a single state as successors, a new proof technique is needed for the partition refinement scheme: In the non-probabilistic setting, edges have no labels and predecessor-based method can be used to refine the partition. This cannot be extended to the probabilistic setting in an obvious way, since, in PAs, states have successor distributions equipped with action labels. We propose a graph-based analysis to refine the partition for PAs. As in [54], the relation over the partition is refined according to stability conditions. We arrive at an algorithm with space complexity $\mathcal{O}(n_\sim^2 + n \log n_\diamond)$ where $n_\sim$ and $n_\diamond$ denote the number of states in the quotient automaton with respect to the strong bisimulation and simulation preorder respectively. The $\mathcal{O}(n_\sim^2)$ part is needed to refine the partition. For saving the partition relation, $n_\diamond^2 \in \mathcal{O}(n_\sim^2)$ is needed. The $\mathcal{O}(n \log n_\diamond)$ part is needed to save to which simulation equivalence class a state belongs to. The time complexity is rather excessive $\mathcal{O}(mn_\diamond + m_\diamond^2 n_\diamond^4 + m_\sim^2 n_\diamond^2)$ where $m_\sim$ and $m_\diamond$ denote the number of transitions in the strong bisimulation, and simulation quotient respectively. Similar to algorithms for deciding simulation preorder for PAs (cf. Chap-

ter 5), one can use the parametric maximum flow idea to improve the time complexity. However, more memory is then needed due to the storage of the networks and the flow values of the corresponding networks across iterations. We show combined with the parametric maximum flow technique, our algorithm has time complexity $\mathcal{O}(mn_\diamond + m_\sim^2 n_\diamond^2)$ and space complexity $\mathcal{O}(m_\diamond^2 + n_\sim^2 + n \log n_\diamond)$.

We have implemented both the space-efficient and time-efficient variants of the partition refinement based algorithm. Experimental results show that the space-efficient algorithm is very effective: not only the space-efficiency is improved drastically, often orders of magnitude less time is required. As for strong simulations, both regular and random experiments show that the parametric maximum flow based implementation does not perform better in general.

Organisation of this Chapter. In Section 7.1 we introduce the notion of quotient automata and show that every probabilistic automaton has a quotient automaton which is the smallest in size, and this quotient automaton can be obtained by the strong simulation preorder. In Section 7.2, we show that strong simulation relations can also be characterised by partition pairs. Using this, we develop a partition refinement based algorithm for computing the strong simulation preorder in Section 7.3. Finally, we report experimental results in Section 7.5 and discuss related works in Section 7.6. We conclude this chapter in Section 7.7.

In this chapter we do not consider weak simulations. If no confusion arises, we use "simulation" to refer to "strong simulation".

7.1 The Quotient Automata

The strong simulation we have considered until now relates states within one PA. It can be lifted to the automaton level. Intuitively, a PA $\mathcal{M}$ is simulated by $\mathcal{M}'$ if the initial state of $\mathcal{M}$ is simulated by the initial state of $\mathcal{M}'$. Simulation between PAs relates their initial states. Thus, in this section, we equip each single PA with an additional initial state: a PA is then a tuple $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$ where s_0 is the initial state. It is then a notational convenience to define the direct sum of two PAs:

Definition 7.1.1 (Direct Sum)**.** *Consider the two PAs Let $\mathcal{M}_1 = (S_1, s_1, Act_1, \mathbf{P}_1, L_1)$ and $\mathcal{M}_2 = (S_2, s_2, Act_2, \mathbf{P}_2, L_2)$ with disjoint set of states. The direct sum of $\mathcal{M}_1$ and $\mathcal{M}_2$, denoted by $\mathcal{M}_1 \oplus \mathcal{M}_2$, is a PA $(S, s_1, Act, \mathbf{P}, L)$ with $S = S_1 \cup S_2$ as its set of states, the transitions are defined by $\mathbf{P} = \mathbf{P}_1 \cup \mathbf{P}_2$, and the labelling function $L : S \to 2^{AP}$ is defined by: $L(s) = L_1(s)$ if $s \in S_1$, and $L(s) = L_2(s)$ if $s \in S_2$.*

The choice of the initial state is arbitrary: The direct sum is only used to define a relation on the common state space, thus the initial state is of no interest. Now we give the definition of simulation for PAs:

Definition 7.1.2 (Simulation for PAs)**.** *Let $\mathcal{M}_1 = (S_1, s_1, Act_1, \mathbf{P}_1, L_1)$ and $\mathcal{M}_2 = (S_2, s_2, Act_2, \mathbf{P}_2, L_2)$ be two PAs, and let $\mathcal{M}_1 \oplus \mathcal{M}_2 = (S, s_1, Act, \mathbf{P}, L)$ be their direct sum. Then, we say that $\mathcal{M}_2$ simulates $\mathcal{M}_1$, denoted by $\mathcal{M}_1 \precsim_\oplus \mathcal{M}_2$, if there exists a simulation $R \subseteq S \times S$ over $\mathcal{M}_1 \oplus \mathcal{M}_2$ such that $s_1 \; R \; s_2$.*

If $\mathcal{M}_1 \precsim_\oplus \mathcal{M}_2$ and $\mathcal{M}_2 \precsim_\oplus \mathcal{M}_1$, we say that they are simulation equivalent, and write $\mathcal{M}_1 \simeq_\oplus \mathcal{M}_2$.

Computing the simulation preorder for a given PA does not depend on the initial state, thus, in the previous chapters, we have omitted the initial state. However, the definition of simulation relation between PAs requires that the initial states of the PAs are related. Thus, in the discussion of this section, we assume that there is an initial state for a PA.

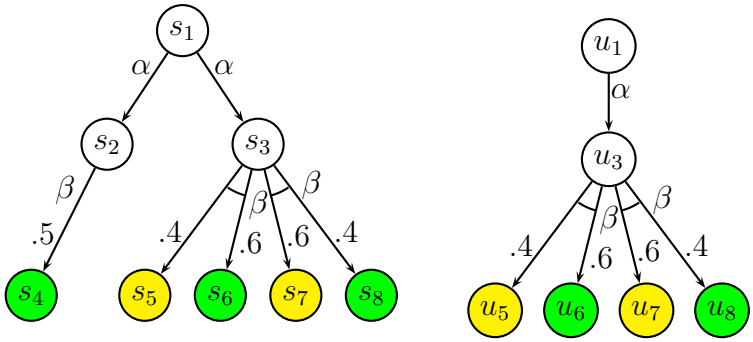

Figure 7.1: Two PAs for illustrating the simulation relations.

Example 7.1.1. Let $\mathcal{M}_1$ denote the PA on the left side of Figure 7.1, and let $\mathcal{M}_2$ denote the PA on the right side. Assume that the initial state of $\mathcal{M}_1$ is s_1, and the initial state of $\mathcal{M}_2$ is u_1. The direct sum $\mathcal{M}_1 \oplus \mathcal{M}_2$ is exactly the PA considered in Example 4.1.4. Since in $\mathcal{M}_1 \oplus \mathcal{M}_2$ it holds that $s_1 \simeq u_1$, we have that $\mathcal{M}_1 \precsim_\oplus \mathcal{M}_2$, $\mathcal{M}_2 \precsim_\oplus \mathcal{M}_1$ and that $\mathcal{M}_1 \simeq_\oplus \mathcal{M}_2$.

Partitions. A partition of S is a set Σ which consists of pairwise disjoint subsets of S such that $S = \cup_{B\in\Sigma} B$. The elements of a partition are also referred to as *blocks*. A partition Σ is *finer* than Σ' if for each block $Q \in \Sigma$ there exists a unique block $Q' \in \Sigma'$ such that $Q \subseteq Q'$. If Σ is finer than Σ', the *parent* block of $B \in \Sigma$ with respect to Σ', denoted by $Par_{\Sigma'}(B)$, is defined as the unique block $B' \in \Sigma'$ with $B \subseteq B'$. For $s \in S$, let $[s]_\Sigma$ denote the unique block in Σ containing state s. If Σ is clear from the context, we write simply $[s]$. For a distribution $\mu \in Dist(S)$ and a partition Σ over S, we define $lift_\Sigma(\mu) \in Dist(\Sigma)$, the induced lifted distribution with respect to Σ, by: $lift_\Sigma(\mu)(B) = \sum_{s\in B} \mu(s)$ for $B \in \Sigma$. In case that the distribution $\mu \in Dist(S)$ is sub-stochastic, $lift_\Sigma(\mu)(B)$ is also sub-stochastic. For technical reasons, we let $\{\bot\}$ denote the unique block containing $\bot$, and let $\Sigma_\bot$ denote the set $\Sigma \cup \{\{\bot\}\}$, and we write that $lift_\Sigma(\mu)(\{\bot\}) := 1 - \mu(S)$.

For a given PA $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$, a partition Σ over S is called consistent with respect to the labelling function L, if for all $B \in \Sigma$ and for all $s, s' \in B$ it holds that $L(s) = L(s')$. Intuitively, if Σ is consistent with respect to L, states in the same block have the same labelling. Recall $s \precsim s'$ implies that $L(s) = L(s')$. In this chapter we consider only partitions which are consistent with respect to L. For consistent partition Σ and $B \in \Sigma$, we write $L(B)$ to denote the labelling of B.

The partition Σ over S induces an equivalence relation $\equiv_\Sigma$ defined by: $s \equiv_\Sigma s'$ iff $[s] = [s']$. If R is an equivalence relation, we let $S/_R$ denote the set of equivalence

classes, which can also be considered a partition of S. Let $\mathcal{I}(S) = \{(s,s) \mid s \in S\}$ denotes the identity relation. For an arbitrary relation R with $\mathcal{I}(S) \subseteq R$, let R^* denote the transitive closure of it, which is a preorder. It induces an equivalence relation $\equiv_{R^*}$ defined by: $s \equiv_{R^*} s'$ if sR^*s' and $s'R^*s$. As a shorthand notation, we let $S/_{R^*}$ denote the corresponding set of equivalence classes $S/_{\equiv_{R^*}}$.

The Quotient Automata. Let $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$ be a PA, and consider the partition Σ over S. For notational convenience, we use $\mu \in Dist(S)$ to denote a distribution over S, and $\pi_\Sigma \in Dist(\Sigma)$ to denote a lifted distribution over the partition Σ. If the partition Σ is clear from the context, we use π instead of π_Σ. For a set $B \subseteq S$, we write

- $B \xrightarrow{\alpha} \pi_\Sigma$ if there exists $s \in B$ and $s \xrightarrow{\alpha} \mu$ with $\pi_\Sigma = lift_\Sigma(\mu)$,
- $B \overset{\alpha}{\rightarrowtail} \pi_\Sigma$ if for all $s \in B$ there exists $s \xrightarrow{\alpha} \mu$ with $\pi_\Sigma = lift_\Sigma(\mu)$.

The transition $B \xrightarrow{\alpha} \pi_\Sigma$ is also called an $\exists$-transition of B with respect to Σ, and $B \overset{\alpha}{\rightarrowtail} \pi_\Sigma$ is also called a $\forall$-transition of B with respect to Σ.

Definition 7.1.3 ($\exists$-Quotient Automaton)**.** *Let $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$ be a PA, and Σ be a partition over S. The $\exists$-quotient automaton $\exists\mathcal{M}/_\Sigma$ is the tuple $(\Sigma, B_0, Act, \mathbf{P}_\exists, L')$ where B_0 is the unique block containing s_0, and the transition matrix is defined by: $\mathbf{P}_\exists = \{(B, \alpha, \pi_\Sigma) \mid B \in \Sigma \wedge B \xrightarrow{\alpha} \pi_\Sigma\}$, and the labelling function is defined by $L'(B) = L(B)$.*

Intuitively, in the $\exists$-quotient automaton, the set of transitions are the $\exists$-transitions with respect to Σ. For block B, $L'(B)$ is well defined because we have assumed that the partition Σ is consistent with respect to L. If no confusion arises, we use B both as a state in the $\exists$-quotient automaton, and as a set of states in $\mathcal{M}$.

Some notations for $\exists\mathcal{M}/_\Sigma$ are in order. For $s \in \Sigma$ and $\alpha \in Act(s)$, let

$$Steps_{\Sigma,\alpha}(s) = \{\pi \in Dist(\Sigma) \mid s \xrightarrow{\alpha} \mu \wedge \pi = lift_\Sigma(\mu)\}$$

denote the set of lifted distributions with respect to Σ for all α-successor distributions of s. For $B \in \Sigma$ let $Steps_{\Sigma,\alpha}(B) = \cup_{s \in B} Steps_{\Sigma,\alpha}(s)$.

Similarly, we define the $\forall$-quotient automaton:

Definition 7.1.4 ($\forall$-Quotient Automaton)**.** *Let $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$ be a PA, and Σ be a partition over S. The $\forall$-quotient automaton $\forall\mathcal{M}/_\Sigma$ is defined as the tuple $(\Sigma, B_0, Act, \mathbf{P}_\forall, L')$ where the transition matrix is defined by: $\mathbf{P}_\forall = \{(B, \alpha, \pi_\Sigma) \mid B \in \Sigma \wedge B \overset{\alpha}{\rightarrowtail} \pi_\Sigma\}$, and the labelling function is defined by $L'(B) = L(B)$.*

The labelling function L' is defined the same as for the $\exists$-quotient automaton. The following lemma is a direct consequence of Definition 7.1.3 and 7.1.4.

Lemma 7.1.1. *Let $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$ be a PA, and Σ be a partition over S. Then, the $\forall$-quotient automaton $\forall\mathcal{M}/_\Sigma$ can be obtained from the $\exists$-quotient automaton $\exists\mathcal{M}/_\Sigma$ by eliminating the little brothers in $\exists\mathcal{M}/_\Sigma$:*

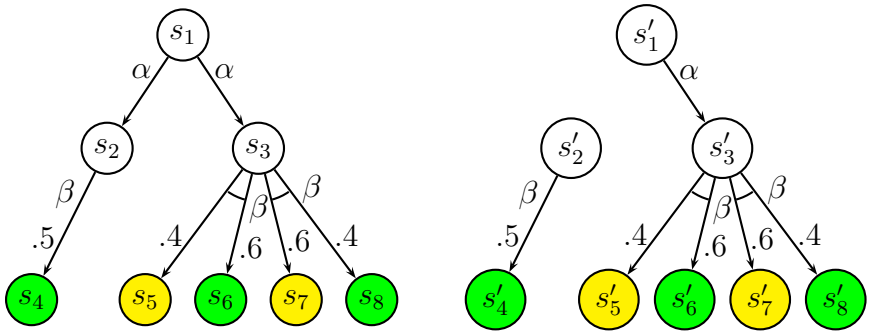

Figure 7.2: A PA $\mathcal{M}$ (on the left side) and the PA $\mathcal{M}'$ obtained by eliminating the little brothers (on the right side).

Example 7.1.2. Consider the direct sum $\mathcal{M}_1 \oplus \mathcal{M}_2$ in Example 7.1.1, and let $\Sigma = \cup_{i=1}^{5} B_i$ be a partition over $S_1 \cup S_2$ where B_i is defined by: $B_i = \{s_i, u_i\}$ if $i = 1, 3$, $B_2 = \{s_2\}$, $B_4 = \{s_4, s_6, s_8, u_6, u_8\}$ and $B_5 = \{s_5, s_7, u_5, u_7\}$. The $\exists$-quotient automaton $\exists\mathcal{M}/_\Sigma$ is the PA on the left side (reachable from state s_1) in Figure 7.1 (with the appropriate relabelling of states), and the $\forall$-quotient automaton $\forall\mathcal{M}/_\Sigma$ is the PA on the right side (reachable from state u_1).

7.1.1 The Minimal Quotient Automaton

Let $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$ be a PA. In this section we show that there exists a PA which is simulation equivalent with $\mathcal{M}$, and is the smallest in size.

In the non-probabilistic setting [29], the notion of *little brothers* is introduced which states that state s_1 is a little brother of s_2 if they have a common predecessor s_3, and s_2 simulates s_1 but not the other way around. Recall $\precsim$ denotes the simulation preorder of $\mathcal{M}$. We lift the notion of little brothers to PAs:

Definition 7.1.5. *Let $s \in S$ be a state, and let $\alpha \in Act(s)$ be an enabled action out of s. For two distributions $\mu, \mu' \in Steps_\alpha(s)$, we say that μ is a little brother of μ' if it holds that $\mu \sqsubseteq_{\precsim} \mu'$ and $\mu' \not\sqsubseteq_{\precsim} \mu$.*

Intuitively, μ is a little brother of μ' if there exists a state s and an action α, such that they both are α-successor distributions of s, and that there exists a weight function for (μ, μ') with respect to $\precsim$ but not the other way around.

Example 7.1.3. Consider the PA depicted on the left part of Figure 7.2. Assume that s_1 is the initial state. Let μ_1 and μ_2 denote the left and right α-successor distributions of s_1 respectively. By Definition 7.1.5, μ_1 is a little brother of μ_2.

In the following we show that by eliminating the little brothers from each state $s \in S$ in a PA we get a simulation equivalent PA. We let the set S' denote a copy $s' \in S'$ of each state $s \in S$ where $S' = \{s' \mid s \in S\}$. For $\mu \in Dist(S)$, let μ' denote the distribution in $Dist(S')$ such that $\mu'(s') = \mu(s)$ for all $s \in S$. The PA $\mathcal{M}' = (S', s'_0, Act, \mathbf{P}', L')$, obtained from $\mathcal{M}$ by eliminating little brothers, is defined by: $\mathbf{P}' \subseteq \mathbf{P}$ such that if $(s, \alpha, \mu) \in \mathbf{P}$ and μ is not a little brother implies that $(s', \alpha, \mu') \in \mathbf{P}'$. The labelling function is defined by $L'(s') = L(s)$ for all $s \in S$.

Lemma 7.1.2. *Let $\mathcal{M}$ be a PA. We consider $\mathcal{M}'$ which is the PA obtained from $\mathcal{M}$ by eliminating all little brothers. Then, $\mathcal{M} \simeq_{\oplus} \mathcal{M}'$.*

Proof. Let $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$, and let $\mathcal{M}' = (S', s_0', Act, \mathbf{P}', L')$ denote the obtained PA by eliminating little brothers of $\mathcal{M}$. Let $\simeq$ and $\precsim$ denote the simulation equivalence and simulation preorder respectively, in the direct sum $\mathcal{M} \oplus \mathcal{M}'$. It is sufficient to show that $s_0 \simeq s_0'$.

We first show that $R = \{(s_1, s_2') \in S \times S' \mid s_1 \precsim_{\mathcal{M}} s_2\}$ is a simulation relation over $\mathcal{M} \oplus \mathcal{M}'$. Let $(s_1, s_2') \in R$. Thus $s_1 \precsim_{\mathcal{M}} s_2$ implying that $L(s_1) = L(s_2) = L(s_2')$. Let $\alpha \in Act(s_1)$ and let $s_1 \xrightarrow{\alpha} \mu_1$. We consider two cases. In the first case, assume that there is μ_2 such that $s_1 \xrightarrow{\alpha} \mu_2$, and μ_1 is a little brother of μ_2. Without loss of generality, assume that μ_2 is not a little brother of any other α-distribution of s_1. Thus, we have that $s' \xrightarrow{\alpha} \mu_2'$ where $\mu_2'(s') = \mu_2(s)$ for all $s \in S$. Obviously, it holds that $\mu_2 \sqsubseteq_R \mu_2'$: the weight function can be defined by $\Delta(s, s') = \mu_2(s)$ for all $s \in S_{\perp}$. Since μ_1 is a little brother of μ_2, it holds that $\mu_1 \sqsubseteq_{\precsim_{\mathcal{M}}} \mu_2$. By definition of R it holds that $\precsim_{\mathcal{M}} \circ R \subseteq R$. Lemma 4.1.2 implies that $\mu_1 \sqsubseteq_{\precsim_{\mathcal{M}} \circ R} \mu_2'$, and moreover $\mu_1 \sqsubseteq_R \mu_2'$. The other case, namely if μ_1 is not a little brother of any other distributions, can be shown similarly. Thus R is a simulation relation over $\mathcal{M} \oplus \mathcal{M}'$. Since $s_0 \precsim_{\mathcal{M}} s_0$, we have that $(s_0, s_0') \in R$, implying that $s_0 \precsim s_0'$.

We show the other direction. By assumption $\mathcal{M}'$ is obtained by eliminating little brothers from $\mathcal{M}$. We consider the relation $R = \{(s', s) \in S' \times S \mid s \in S\}$. Since for each $s' \xrightarrow{\alpha} \mu'$, there exists $s \xrightarrow{\alpha} \mu$ with $\mu(s) = \mu'(s')$ for all $s \in S$. $\mu' \sqsubseteq_R \mu$ holds with the weight function Δ defined by: $\Delta(s', s) = \mu(s)$ for all $s \in S_{\perp}$, implying that R is a simulation relation over $\mathcal{M} \oplus \mathcal{M}'$. Since $(s_0', s_0) \in R$, we conclude that $s_0' \precsim s_0$. ■

Recall that the preorder $\precsim$ on S induces an equivalence relation $\simeq$. The following lemma states that $\mathcal{M}$ and its $\forall$-quotient automaton with respect to $\simeq$ are simulation equivalent.

Lemma 7.1.3. *Given a PA $\mathcal{M}$, the equivalence relation $\simeq_{\mathcal{M}}$ over $\mathcal{M}$ induces a partition of S defined by: $\Sigma = \{\{s' \mid s' \in S \wedge s \simeq_{\mathcal{M}} s'\} \mid s \in S\}$. Then, $\forall\mathcal{M}/_{\Sigma}$ and $\mathcal{M}$ are simulation equivalent: $\forall\mathcal{M}/_{\Sigma} \simeq_{\oplus} \mathcal{M}$.*

Proof. Let $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$ be the given PA. Recall that the $\forall$-quotient automaton $\forall\mathcal{M}/_{\Sigma}$ is a five tuple: $(\Sigma, B_0, Act, \mathbf{P}_{\forall}, L)$ in which B_0 is the unique block containing the initial state s_0. Let $\simeq$ and $\precsim$ denote the simulation equivalence and simulation preorder respectively, in the direct sum $\mathcal{M} \oplus \forall\mathcal{M}/_{\Sigma}$. It is sufficient to show $s_0 \simeq B_0$. We show only $s_0 \precsim B_0$ (the other direction is simpler and can be shown in a similar way).

Let the relation $R \subseteq S \times \Sigma$ defined as follows: $R = \{(s, B) \mid s \in S \wedge B \in \Sigma \wedge s \in B\}$. We first show that R is a simulation relation over $\mathcal{M} \oplus \forall\mathcal{M}/_{\Sigma}$. Let $(s, B) \in R$, and assume that $s \xrightarrow{\alpha} \mu$. By definition of strong simulation (cf. Definition 4.1.4) and definition of $\forall$-quotient automaton (cf. Definition 7.1.4), it is sufficient to show that there exists $B \overset{\alpha}{\rightarrowtail} \pi_{\Sigma}$ such that $\mu \sqsubseteq_R \pi_{\Sigma}$. Let $B = \{b_1, \ldots, b_k\}$, then, it holds that $b_1 \simeq_{\mathcal{M}} b_2 \ldots \simeq_{\mathcal{M}} b_k$ and that $s = b_i$ for some $i \in \{1, \ldots, k\}$. Thus, by definition of simulation relations, there exists an infinite sequence $\mu_1, \mu_2, \mu_3, \ldots$ such that

1. $b_i \xrightarrow{\alpha} \mu_j$ provided that $j \mod k = i$, and

2. $\mu \sqsubseteq_{\simeq_{\mathcal{M}}} \mu_1$ and $\mu_i \sqsubseteq_{\simeq_{\mathcal{M}}} \mu_{i+1}$ for all $i \geq 1$.

Since we have only finite many distributions, there must exists a l such that $\mu_i = \mu_j$ for all $i, j \geq l$. Let $\pi_\Sigma = \mathit{lift}_\Sigma(\mu_l)$, then, we have that $B \overset{\alpha}{\rightarrowtail} \pi_\Sigma$. Because of the transitivity of $\simeq_{\mathcal{M}}$, we have that $\mu \sqsubseteq_{\simeq_{\mathcal{M}}} \mu_l$. Now it remains to show that $\mu \sqsubseteq_R \pi_\Sigma$. We define the function $\Delta : S_\perp \times \Sigma_\perp \to [0,1]$ as follows: $\Delta(s, B) = \mu(s)$ if $s \in B$, and 0 otherwise. We show that the defined function is a weight function for (μ, π_Σ) with respect to R:

1. By definition of the weight function, $\Delta(s, B) > 0$ implies that either $(s, B) \in R$ or $s = \perp$.

2. We show the Condition 2 of Definition 4.1.1. For $s \in S_\perp$, we want to show that $\mu(s) = \sum_{B \in \Sigma_\perp} \Delta(s, B)$. By definition of Δ, we have $\mu(s) = \Delta(s, B')$ where B' is the unique block in Σ containing s. For $B \neq B'$ it holds that $\Delta(s, B) = 0$, implying that $\mu(s) = \sum_{B \in \Sigma_\perp} \Delta(s, B)$.

3. Now let $B \in \Sigma_\perp$. By definition, we have $\pi_\Sigma(B) = \sum_{s \in B} \mu_l(s)$. Since for $s \notin B$ we have $\Delta(s, B) = 0$, which implies that $\pi_\Sigma(B) = \sum_{s \in S_\perp} \Delta(s, B)$.

Thus, R is a simulation relation. By definition of R it holds that $(s_0, B_0) \in R$, thus $s_0 \precsim B_0$. ■

By Lemma 7.1.1, the $\forall$-quotient automaton can be obtained from the $\exists$-quotient automaton by eliminating little brothers in it. Combining Lemma 7.1.2 and the above lemma, we have that $\mathcal{M}$, its $\forall$-quotient automaton, and its $\exists$-quotient automaton are pairwise simulation equivalent. Recall that for $\mathcal{M}$, $n = |S|$ denotes the number of the states, and $m = \sum_{s \in S} \sum_{\alpha \in Act(s)} \sum_{\mu \in Steps_\alpha(s)} |\mu|$ denotes the size of the transitions. The following lemma states that the $\forall$-quotient automaton of $\mathcal{M}$ is the smallest one among those PAs which are simulation equivalent to $\mathcal{M}$.

Lemma 7.1.4. *Let $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$ be a PA in which all states are reachable from s_0. Let $\mathcal{M}' = (S', s_0', Act, \mathbf{P}', L')$ be any other PA which is simulation equivalent with $\mathcal{M}$. Let Σ denote the partition of S induced by $\simeq_{\mathcal{M}}$. Moreover, let m_Σ, n_Σ be the size of transitions and states of $\forall\mathcal{M}/_\Sigma$, m', n' be the size of transitions and states of $\mathcal{M}'$ respectively. Then, it holds that $n_\Sigma \leq n'$ and $m_\Sigma \leq m'$.*

Proof. Let Σ denote the partition of S induced by $\simeq_{\mathcal{M}}$. Let $\simeq$ and $\precsim$ denote the simulation equivalence and simulation preorder respectively, in the direct sum $\forall\mathcal{M}/_\Sigma \oplus \mathcal{M}'$.

Let $[s]_\Sigma$ be an arbitrary state in $\forall\mathcal{M}/_\Sigma$. We show that there exists a state $s' \in S'$ with $[s]_\Sigma \simeq s'$. By assumption $[s]_\Sigma$ is reachable from $[s_0]_\Sigma$, i.e., there exists a sequence $([s_0]_\Sigma, \alpha_0, \pi_0), \ldots, ([s_{k-1}]_\Sigma, \alpha_{k-1}, \pi_{k-1}), [s_k]_\Sigma$ with $s_k = s$, and $[s_i]_\Sigma \xrightarrow{\alpha_i} \pi_i$ and $\pi_i([s_{i+1}]_\Sigma) > 0$ for $i = 0, \ldots, k-1$. It is sufficient to show, by induction on i, there exists s_i' for $i = 1, \ldots, k$ such that $[s_i]_\Sigma \simeq s_i'$.

- By assumption it holds that $\forall\mathcal{M}/_\Sigma \simeq_\oplus \mathcal{M}'$, which implies that $[s_0]_\Sigma \simeq s_0'$. Thus the base case $i = 0$ holds.

- For the induction step: assume that $[s_i]_\Sigma \simeq s'_i$ for $i \leq k-1$. This implies that $[s_i]_\Sigma \precsim s'_i$ and $s'_i \precsim [s_i]_\Sigma$. Since $[s_i]_\Sigma \xrightarrow{\alpha_i} \pi_i$, there exists $s'_i \xrightarrow{\alpha_i} \pi'_i$ such that $\pi_i \sqsubseteq_\precsim \pi'_i$. Then, $s'_i \precsim [s_i]_\Sigma$ implies also that there must exists $[s_i]_\Sigma \xrightarrow{\alpha_i} \pi''_i$ such that $\pi'_i \sqsubseteq_\precsim \pi''_i$. Since $\precsim$ is transitive, by Lemma 4.1.3, we have that $\pi_i \sqsubseteq_\precsim \pi''_i$. Observe that the $\forall$-quotient automata there are no little brothers, thus it must holds that $\pi_i = \pi''_i$. Since $\precsim$ is a preorder, applying Lemma 5.3.5 of [6] (cf. Lemma 4.1.5), it holds that $\pi_i(A) = \pi'_i(A)$ for all simulation equivalence class A of $\simeq$. Let $A = [s_{i+1}]_\simeq$, thus it holds that $[s_{i+1}]_\Sigma \subseteq [s_{i+1}]_\simeq$. Then, $\pi_i([s_{i+1}]_\Sigma) > 0$ implies that $\pi'_i([s_{i+1}]_\simeq) > 0$, thus there must exist $s' \in [s_{i+1}]_\simeq$ with $s' \in S'$ and $[s_{i+1}]_\Sigma \simeq s'$.

Assume that $n_\Sigma > n'$ for the sake of contradiction. Since $n_\Sigma > n'$, there must be at least two states $[s_1]_\Sigma, [s_2]_\Sigma$ in $\forall\mathcal{M}/_\Sigma$ such that there is one state s' in $\mathcal{M}'$ such that $[s_1]_\Sigma \simeq s'$ and $[s_2]_\Sigma \simeq s'$. By the transitivity of $\simeq$ we have that $[s_1]_\Sigma \simeq [s_2]_\Sigma$ which is a contradiction because by construction of $\forall\mathcal{M}/_\Sigma$ no two states are simulation equivalent. Thus it must holds that $n_\Sigma \leq n'$.

Now we show that $m_\Sigma \leq m'$. Assume that $[s]_\Sigma$ is an arbitrary state of $\forall\mathcal{M}/_\Sigma$, and assume that $\alpha \in Act([s]_\Sigma)$ and that π is an arbitrary α-successor distribution of $[s]_\Sigma$. Then, using similar argument above, there must be a distinguished state $s' \in S'$ with $[s]_\Sigma \simeq s'$. Moreover, there must exist an α-successor distribution π' of s' with $\pi \sqsubseteq_\precsim \pi'$ and $\pi' \sqsubseteq_\precsim \pi$. It holds also that $\pi(A) = \pi'(A)$ for all simulation equivalence class of $\simeq$. Observe that each such equivalence class A contains at most a single state in $\forall\mathcal{M}/_\Sigma$, thus $|\pi| \leq |\pi'|$. It is then also easy to see that for two α-successor distributions π_1 and π_2 of $[s]_\Sigma$, there must exist two distinguished α-successor distributions π'_1 and π'_2 with $\pi_i \sqsubseteq_\precsim \pi'_i$ and $\pi'_i \sqsubseteq_\precsim \pi_i$ for $i = 1, 2$, proving that $m_\Sigma \leq m'$. ■

In the above lemma, we require that all states in the PA are reachable from the initial state. This is not a real restriction. As in the non-probabilistic setting [29], by pruning the unreachable states of $\mathcal{M}$ we get a PA which is simulation equivalent to $\mathcal{M}$. This can be performed by a single search algorithm. Thus, to construct the minimal quotient automaton for $\mathcal{M}$, the central problem is to decide the simulation preorder.

7.1.2 Safety and Liveness Properties

For PAs, strong (probabilistic) simulation equivalence is strictly coarser than strong (probabilistic) bisimulation. We briefly discuss which classes of properties are preserved by strong (probabilistic) simulation quotient automata. Strong (probabilistic) simulation is known to preserve the safe fragment of PCTL [101]: If $\mathcal{M}_1 \precsim \mathcal{M}_2$, then if $\mathcal{M}_2$ satisfies a safety formula Φ, then $\mathcal{M}_1$ satisfies Φ as well. There is a duality between safety and liveness fragments of PCTL formulas, thus the above statement is equivalent to: If $\mathcal{M}_1 \precsim \mathcal{M}_2$, then if $\mathcal{M}_1$ satisfies a liveness formula Φ, then $\mathcal{M}_2$ satisfies Φ as well. Given a PA $\mathcal{M}$, by Lemma 7.1.3, the $\forall$-quotient automaton $\forall\mathcal{M}/_\precsim$ are simulation equivalent to $\mathcal{M}$. Thus, both safe and liveness fragments of PCTL properties are preserved by the minimal quotient automaton.

7.2 Simulation Characterised by Partition Pairs

In the non-probabilistic setting, the simulation preorder for unlabelled graph is characterised by partition pairs [29, 54] which consist of a partition of the state space and a binary relation over the partition. Then, a partition refinement approach is introduced based on partition pairs. In this section, we adapt the notion of partition pairs to PAs, and then we show that we can characterise simulation relations for PAs by partition pairs. This is the basis for the partition refinement approach which will be introduced in the next section. In the remainder of this section, we fix a PA $\mathcal{M} = (S, Act, \mathbf{P}, L)$: The initial state does not play a role in the computation of the simulation preorder, it is again omitted in this and subsequent sections.

We say that the pair $(B, B') \in \Sigma \times \Sigma$ respects the labelling function L if $L(B) = L(B')$. Now we give the definition of partition pairs.

Definition 7.2.1 (Partition Pair)**.** *A partition pair over S is a pair $\langle \Sigma, \Gamma \rangle$ where Σ is a partition of S, and $\Gamma \subseteq \Sigma \times \Sigma$ is a reflexive relation over Σ satisfying the condition: all pair $(B, B') \in \Gamma$ respects the labelling function L.*

We also call Γ the partition relation. Let Υ denote the set of all partition pairs over S. For $\langle \Sigma, \Gamma \rangle \in \Upsilon$ and $B, B' \in \Sigma$, we also write also $B \Gamma B'$ if $(B, B') \in \Gamma$. A partition pair induces a binary relation on S as follows:

Definition 7.2.2 (Induced Relation)**.** *The partition pair $\langle \Sigma, \Gamma \rangle \in \Upsilon$ induces the binary relation on S by: $\precsim_{\langle \Sigma, \Gamma \rangle} = \{(s, s') \mid [s] \Gamma [s']\}$.*

Let $\langle \Sigma, \Gamma \rangle, \langle \Sigma', \Gamma' \rangle \in \Upsilon$. If Σ if finer than Σ', and $\precsim_{\langle \Sigma, \Gamma \rangle} \subseteq \precsim_{\langle \Sigma', \Gamma' \rangle}$ holds, we say that Γ is finer than Γ'. Now we introduce a partial order on Υ:

Definition 7.2.3 (Partial Order)**.** *We define an order $\ltimes \subseteq \Upsilon \times \Upsilon$ as follows: $\langle \Sigma, \Gamma \rangle \ltimes \langle \Sigma', \Gamma' \rangle$ if Σ is finer than Σ' and Γ is finer than Γ'.*

If $\langle \Sigma, \Gamma \rangle \ltimes \langle \Sigma', \Gamma' \rangle$ we say $\langle \Sigma, \Gamma \rangle$ is finer than $\langle \Sigma', \Gamma' \rangle$. Obviously the defined relation is a partial order: $\ltimes$ satisfies the reflexivity, antisymmetry and transitivity conditions. Now we introduce the stability of partition pairs.

Definition 7.2.4 (Stable Partition Pairs)**.** *A partition pair $\langle \Sigma, \Gamma \rangle \in \Upsilon$ is* stable *if for each $B \Gamma B'$ and $B \xrightarrow{\alpha} \pi_\Sigma$, there exists $B' \overset{\alpha}{\rightarrowtail} \pi'_\Sigma$ such that $\pi_\Sigma \sqsubseteq_\Gamma \pi'_\Sigma$.*

The stable condition in the above definition is illustrated in Figure 7.3, where an $\exists$-transition starts from some state inside the block B and is labelled with $\exists$, and a $\forall$-transition starts from the block B and is labelled with $\forall$. Now we consider an example.

Example 7.2.1. Consider the direct sum $\mathcal{M}_1 \oplus \mathcal{M}_2$ in Example 7.1.1, and consider the partition pair $\langle \Sigma_1, \Gamma_1 \rangle$ defined by: $\Sigma_1 = \{B_1, B_4, B_5\}$ with $B_1 = \{s_1, u_1, s_2, s_3, u_3\}$, $B_4 = \{s_4, s_6, s_8, u_6, u_8\}$ and $B_5 = \{s_5, s_7, u_5, u_7\}$, Γ_1 is $\mathcal{I}(\Sigma)$, i.e., the identical relation over Σ. This partition pair is not stable: consider the partition relation (B_1, B_1), and the transition $B_1 \xrightarrow{\alpha} \mu$ with $\mu(s_2) = 1$ and 0 otherwise. Since states s_2, s_3, u_3 have no α-successor distributions, there does not exist the required $\forall$-transition out of B_1, thus the condition in Definition 7.2.4 is violated.

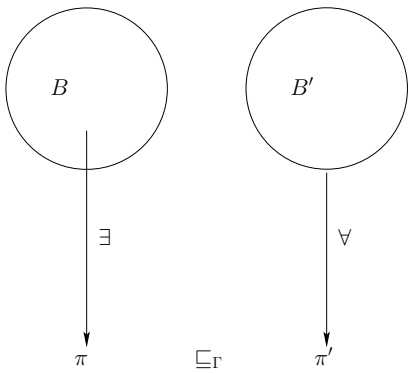

Figure 7.3: A figure for illustrating the stable condition for partition pairs. An $\exists$-transition starts from some state inside the block B and is labelled with $\exists$. Similarly, a $\forall$-transition starts from the block B and is labelled with $\forall$.

Consider another partition pair $\langle \Sigma_2, \Gamma_2 \rangle$ with $\Sigma_2 = \cup_{i=1}^{5} B_i$ be a partition over $S_1 \cup S_2$ where B_i is defined by: $B_2 = \{s_2\}$, $B_i = \{s_i, u_i\}$ if $i = 1, 3$, B_4, B_5 defined as above. The partition relation Γ_2 is the identical relation with an additional pair (B_2, B_3). It is easy to check that this partition pair is stable.

Let Υ_{sta} denote the set of all stable partition pairs. We show that a stable partition pair induces a simulation relation.

Theorem 7.2.1 (Induced Simulation Relation)**.** *Let $\langle \Sigma, \Gamma \rangle \in \Upsilon_{sta}$ be a stable partition pair. Then, the induced relation $\precsim_{\langle \Sigma, \Gamma \rangle}$ is a simulation relation.*

Proof. Let $R = \precsim_{\langle \Sigma, \Gamma \rangle}$. For $(s, s') \in R$, by definition of $\precsim_{\langle \Sigma, \Gamma \rangle}$, we have that $([s]_\Sigma, [s']_\Sigma) \in \Gamma$. Thus we have that $L([s]_\Sigma) = L([s']_\Sigma)$ which implies that $L(s) = L(s')$. Let $s \xrightarrow{\alpha} \mu$ with $\mu \in \mathit{Dist}(S)$. We show that there exists $s' \xrightarrow{\alpha} \mu'$ and $\mu \sqsubseteq_R \mu'$. We observe that $s \xrightarrow{\alpha} \mu$ implies that $[s]_\Sigma \xrightarrow{\alpha} \pi_\Sigma$ with $\pi_\Sigma = \mathit{lift}_\Sigma(\mu)$. Since $\langle \Sigma, \Gamma \rangle$ is a stable partition pair, there exists a distribution $\pi'_\Sigma \in \mathit{Dist}(\Sigma)$ such that $[s']_\Sigma \overset{\alpha}{\rightarrowtail} \pi'_\Sigma$ and $\pi_\Sigma \sqsubseteq_\Gamma \pi'_\Sigma$. Let $s' \xrightarrow{\alpha} \mu'$ with $\pi'_\Sigma = \mathit{lift}_\Sigma(\mu')$, and let Δ_Γ denote the corresponding weight function for $(\pi_\Sigma, \pi'_\Sigma)$ with respect to Γ.

In the following we construct a weight function Δ for (μ, μ') with respect to R. For this purpose we first introduce some variables. For each $s \in S_\perp$, we introduce two variables x_s and x'_s which are initialised by: $x_s = \mu(s)$ and $x'_s = \mu'(s)$. For $(B, B') \in \Gamma$, we introduce a variable $x_{B,B'}$ which is initialised to $\Delta_\Gamma(B, B')$. By definition, it holds that $\mathit{lift}_\Sigma(\mu)(B) = \sum_{s \in B} x_s$ and that $\mathit{lift}_\Sigma(\mu')(B) = \sum_{s' \in B} x'_{s'}$. Since $\mathit{lift}_\Sigma(\mu) \sqsubseteq_R \mathit{lift}_\Sigma(\mu')$, after initialisation, it holds that:

$$\sum_{B \in \Sigma_\perp} \sum_{s \in B} x_s = \sum_{B \in \Sigma_\perp} \sum_{B' \in \Sigma_\perp} x_{B,B'} = \sum_{B' \in \Sigma_\perp} \sum_{B \in \Sigma_\perp} x_{B,B'} = \sum_{B' \in \Sigma_\perp} \sum_{s' \in B} x'_{s'} \tag{7.1}$$

The function $\Delta : S_\perp \times S_\perp \to [0, 1]$ is defined as follows. If there exists $x_{B,B'} > 0$, we perform the operation *discharge*:

1. let $s \in B$ such that $x_s > 0$, let $s' \in B'$ such that $x'_{s'} > 0$,

2. define $\Delta(s, s') = \min\{x_s, x_{B,B'}, x'_{s'}\}$,
3. update the variables by: $x_s = x_s - \Delta(s, s')$, $x_{B,B'} = x_{B,B'} - \Delta(s, s')$ and $x'_{s'} = x'_{s'} - \Delta(s, s')$.

Observe that the above operation preserves Equation 7.1: since at step 3 each of the term is decreased by the value $\Delta(s, s')$. After one *discharge* operation at least one variable becomes 0, thus the operation can be applied at most $2|S| + |S|^2$ times after which all of the variables are 0, and we obtain the function Δ. Now we show that the constructed Δ is the desired weight function. By construction, $\Delta(s, s') > 0$ implies that $x_{B,B'} > 0$ with $s \in B$ and $s' \in \Sigma$. This implies that $(B, B') \in \Gamma$. By definition of the relation R we have that $(s, s') \in R$, thus the first condition of weight function conditions holds. Let $s \in S_\perp$. If $\mu(s) = 0$, the variable x_s is initialised to 0. And in *discharge* operation only weights are assigned to pair (s, s') with $x_s > 0$, thus we have $\Delta(s, S) = 0$. Now assume that $\mu(s) > 0$. In this case we show that the following invariant holds during the *discharge* operations:

$$\mu(s) = \sum_{s' \in S_\perp} \Delta(s, s') + x_s$$

Before any *discharge* operations, both side of the equation is $\mu(s)$. Assume that at some *discharge* operation the variable x_s is decreased (otherwise the right side of the equation does not change) by the amount of $\Delta(s, s')$ for some $s' \in S_\perp$. Observe that before this discharge operation it holds that $\Delta(s, s') = 0$ (otherwise one of the variables $x_s, x_{B,B'}, x'_{s'}$ must be 0 which is not possible). Moreover, the decreased amount $\Delta(s, s')$ is exactly the new weight assigned to the pair (s, s'). Thus the right side of the equation does not change. At the end if no *discharge* operations are available anymore, we have that $x_s = 0$, in which case we have $\mu(s) = \sum_{s' \in S_\perp} \Delta(s, s')$. Thus the second condition of the weight function definition holds at the end. The third condition of weight function conditions is symmetric to the second one, and can be shown similarly. ■

The induced simulation relation of the stable partition pair $\langle \Sigma_2, \Gamma_2 \rangle$ in Example 7.2.1 is a simulation relation $\precsim$. In the following we give the definition that a set of states is stable with respect to a partition pair:

Definition 7.2.5. *Let $\langle \Sigma, \Gamma \rangle$ be a partition pair and let $B \in \Sigma$. Assume that $Q \subseteq B$. We say that Q is stable with respect to $\langle \Sigma, \Gamma \rangle$ if $Q \xrightarrow{\alpha} \pi_\Sigma$ implies that there exists $Q \overset{\alpha}{\rightarrowtail} \pi'_\Sigma$ such that $\pi_\Sigma \sqsubseteq_\Gamma \pi'_\Sigma$.*

Figure 7.4 illustrates the stable condition for a set of states in the previous definition. Assume that Σ' is a refinement of Σ. Then, we say that Σ' is stable with respect to $\langle \Sigma, \Gamma \rangle$ if each $B \in \Sigma'$ is stable with respect to $\langle \Sigma, \Gamma \rangle$.

Simulations & Stable Partition Pairs. We define a function which establishes connections between simulation[1] relations and stable partition pairs. Recall that $\mathcal{I}(S) = \{(s, s) \mid s \in S\}$ denotes the identity relation over S. We consider the set $\Xi := \{R \subseteq S \times S \mid \mathcal{I}(S) \subseteq R\}$ of relations containing the identity relation. We define the function $H : \Xi \to \Upsilon$ by:

[1]Recall we use simulation to refer to strong simulation in this chapter.

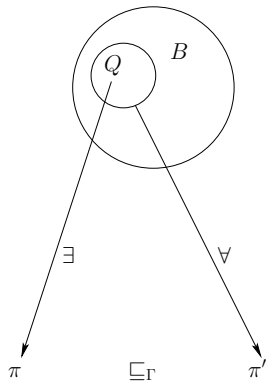

Figure 7.4: A figure for illustrating the stable condition for a subset of states of B. An $\exists$-transition starts from some state inside the set Q and is labelled with $\exists$. Similarly, a $\forall$-transition starts from the set Q and is labelled with $\forall$.

- $H(R) = (S/_{R^*}, \Gamma_R)$ where Γ_R is defined by: $B\Gamma_R B'$ if sR^*s' for all $s \in B$ and $s' \in B'$.

For $R \in \Xi$, $H(R)$ is a pair $(S/_{R^*}, \Gamma_R)$ where the partition is induced by the equivalence relation $\equiv_{R^*}$. Intuitively, $B\Gamma_R B'$ if states in B' are reachable from states in B in the transitive closure R^*. If R is a simulation relation, sR^*s' implies that $L(s) = L(s')$ which implies that Γ_R respects the labelling function. Let $\Upsilon^{\diamond}_{sta} \subseteq \Upsilon_{sta}$ be the set of stable partition pairs in which the partition relation is a preorder. The following lemma states that if R is a preorder and a simulation relation, the image of it is an element of $\Upsilon^{\diamond}_{sta}$:

Lemma 7.2.2. *Assume $R \in \Xi$ is a preorder and a simulation relation. Then, $H(R) \in \Upsilon^{\diamond}_{sta}$.*

Proof. Since R is a preorder, it holds that $R^* = R$. We first show that $H(R) = (S/_R, \Gamma_R) \in \Upsilon_{sta}$. Let $(B, B') \in \Gamma_R$ and let $B \xrightarrow{\alpha} \pi$ with $\pi \in \mathit{Dist}(S/_R)$, thus there exists $s \in B$ with $s \xrightarrow{\alpha} \mu$. It is sufficient to show that there exists $B' \overset{\alpha}{\rightarrowtail} \pi'$ with $\pi' \in \mathit{Dist}(S/_R)$ satisfying $\pi \sqsubseteq_{\Gamma_R} \pi'$. Without loss of generality, assume that $B' = \{s_1, \ldots, s_n\}$, thus, these states are pairwise simulation equivalent. Since $(B, B') \in \Gamma_R$, we have that $s \precsim s_i$ for all $i \in \{1, \ldots, n\}$. We consider the infinite sequence of states $s, (s_1, \ldots, s_n)^\omega$ where the every state is simulated by the following state. By definition of simulation, for the transition $s \xrightarrow{\alpha} \mu$, there exists an infinite distributions $\mu_1, \mu_2, \ldots$ such that $s_i \xrightarrow{\alpha} \mu_j$ where $j \mod n = i$ and $\mu_i \sqsubseteq_R \mu_{i+1}$ for $i \geq 1$. Now we use a similar argument as in non-probabilistic systems [54]: Since the set of distributions reachable from B' is finite, there must exists k such that $s_1 \xrightarrow{\alpha} \mu_k$ and $s_1 \xrightarrow{\alpha} \mu_{k+n} = \mu_k$. Since R is a preorder, thus transitive, which implies that $\mu_i \sqsubseteq_R \mu_j$ for all $i, j \in \{k, \ldots, k+n\}$. Since R is a preorder, applying Lemma 5.3.5 of [6] (cf. Lemma 4.1.5), it holds that $\mu_i(A) = \mu_j(A)$ for all $i, j \in \{k, \ldots, k+n\}$ and for all $A \in S/_R$. Hence, the lifted distributions are the same: $\mathit{lift}_{S/_R}(\mu_k) = \ldots = \mathit{lift}_{S/_R}(\mu_{k+n})$. Let π' denote this lifted distribution. Let Δ denote the weight function for (μ, μ_k) with respect to R. We define Δ' by $\Delta'(B, B') = \sum_{s \in B, s' \in B'} \Delta(s, s')$. It is routine to verify that Δ' is a weight

function for (π, π') with respect to Γ_R, which implies that $\pi \sqsubseteq_{\Gamma_R} \pi'$. Thus, we have $(S/_R, \Gamma_R) \in \Upsilon_{sta}$. The fact that R is a preorder implies that Γ_R is also a preorder, thus, we have that $(S/_R, \Gamma_R) \in \Upsilon^{\diamond}_{sta}$. ■

Let $\Xi^{\precsim} \subseteq \Xi$ be the set consisting of $R \in \Xi$ which is a preorder and a simulation relation. We show that the function obtained from H with restricted domain $\Xi^{\precsim}$ and co-domain $\Upsilon^{\diamond}_{sta}$, is bijective.

Lemma 7.2.3. *Let the function $h : \Xi^{\precsim} \to \Upsilon^{\diamond}_{sta}$ defined by: $h(R) = H(R)$ if $R \in \Xi^{\precsim}$. Then, h is bijective.*

Proof. We first show that h is injective. Assume that $R_1, R_2 \in \Xi^{\precsim}$ with $R_1 \neq R_2$. Without loss of generality, assume $(s, s') \in R_1$ and $(s, s') \notin R_2$. Since R_1 and R_2 are preorders and simulation relations, $(s, s') \in R_1$ implies that s and s' are in the same class of $S/_{R_1^*}$, and $(s, s') \notin R_2$ implies that s, s' are in different classes of $S/_{R_2^*}$. Hence, $h(R_1) \neq h(R_2)$.

Now we show that h is surjective. Let $\langle \Sigma, \Gamma \rangle \in \Upsilon^{\diamond}_{sta}$. To show h is surjective, it is sufficient to show that $h(\precsim_{\langle \Sigma, \Gamma \rangle}) = \langle \Sigma, \Gamma \rangle$ and $\precsim_{\langle \Sigma, \Gamma \rangle} \in \Xi^{\precsim}$. The former is obvious by the construction of the relation $\precsim_{\langle \Sigma, \Gamma \rangle}$, thus it remains to show that $\precsim_{\langle \Sigma, \Gamma \rangle}$ is a preorder and a simulation relation. By Lemma 7.2.1, it is a simulation relation. The partition relation Γ is reflexive. By the construction of $\precsim_{\langle \Sigma, \Gamma \rangle}$, it is reflexive as well. The transitivity follows by exploiting the fact that Γ is transitive, thus, $\precsim_{\langle \Sigma, \Gamma \rangle}$ is a preorder. ■

Recall that $\precsim$ is a preorder, and is the largest simulation relation in $\mathcal{M}$. We use $\langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle$ to denote the partition pair $h(\precsim)$. Thus, $\precsim$ can be obtained via computing $\langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle$. In the following lemma we show that $\langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle$ is the unique, maximal element of $\Upsilon^{\diamond}_{sta}$:

Theorem 7.2.4 (Unique, Maximal Element)**.** *The partition pair $\langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle$ is the unique, maximal element of $\Upsilon^{\diamond}_{sta}$.*

Proof. We first show that H is monotone: $R_1 \subseteq R_2$ implies that $H(R_1) \ltimes H(R_2)$. Observe that $R_1 \subseteq R_2$ implies that $R_1^* \subseteq R_2^*$. Thus, $S/_{R_1^*}$ is finer than $S/_{R_2^*}$. It remains to show that $\precsim_{H(R_1)} \subseteq \precsim_{H(R_2)}$. Let $(s, s') \in \precsim_{H(R_1)}$. By definition s can reach s' in the transitive closure R_1^*. Since $R_1^* \subseteq R_2^*$, s can also reach s' in the transitive closure R_2^*, thus $(s, s') \in \precsim_{H(R_2)}$ which implies that $\precsim_{H(R_1)} \subseteq \precsim_{H(R_2)}$.

Now we prove the lemma. By Lemma 7.2.2, $\langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle = h(\precsim) \in \Upsilon^{\diamond}_{sta}$. Let $\langle \Sigma, \Gamma \rangle$ be an arbitrary element of $\Upsilon^{\diamond}_{sta}$. To prove that it is maximal, it is sufficient to prove $\langle \Sigma, \Gamma \rangle \ltimes \langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle$. By Lemma 7.2.1, $\precsim_{\langle \Sigma, \Gamma \rangle}$ is a simulation relation, thus we have $\precsim_{\langle \Sigma, \Gamma \rangle} \subseteq \precsim = \precsim_{\langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle}$. Exploiting the monotonicity of the function h we have that $h(\precsim_{\langle \Sigma, \Gamma \rangle}) \ltimes h(\precsim_{\langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle})$. By Lemma 7.2.3, we have that $h(\precsim_{\langle \Sigma, \Gamma \rangle}) = h(h^{-1}\langle \Sigma, \Gamma \rangle) = \langle \Sigma, \Gamma \rangle$ and that $h(\precsim_{\langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle}) = h(h^{-1}\langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle) = \langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle$, thus $\langle \Sigma, \Gamma \rangle \ltimes \langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle$. Now it remains to prove that it is unique. Let $\langle \Sigma_*, \Gamma_* \rangle$ be another maximal element of $\Upsilon^{\diamond}_{sta}$. Thus we have that $\langle \Sigma_*, \Gamma_* \rangle \ltimes \langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle$ and $\langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle \ltimes \langle \Sigma_*, \Gamma_* \rangle$. Hence, $\langle \Sigma_*, \Gamma_* \rangle = \langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle$ as the relation $\ltimes$ is antisymmetric. ■

Thus, to determine the simulation preorder $\precsim$, it is sufficient to compute the partition pair $\langle \Sigma^{\diamond}, \Gamma^{\diamond} \rangle$. As in [54] we refer to it as the generalised coarsest partition problem (GCPP).

7.3 Solving the GCPP

As before, we fix a PA[2] $\mathcal{M} = (S, Act, \mathbf{P}, L)$. In this section we propose an algorithm for solving the GCPP, i.e., computing the partition pair $\langle \Sigma^\diamond, \Gamma^\diamond \rangle$ based on the partition refinement strategy. The idea is that we start with the partition pair $\langle \Sigma_0, \Gamma_0 \rangle$ which is coarser than $\langle \Sigma^\diamond, \Gamma^\diamond \rangle$, and refine it with respect to the stability conditions. The Algorithm SimQuo is presented in Algorithm 11. As an initial partition pair we take

$$\Sigma_0 = \{\{s' \in S \mid L(s) = L(s') \wedge Act(s) = Act(s')\} \mid s \in S\}.$$

Intuitively, states with the same labelling and enabled actions are put in the the same initial block. By construction Σ_0 is consistent with respect to L. The initial partition relation is defined by:

$$\Gamma_0 = \{(B, B') \in \Sigma_0 \times \Sigma_0 \mid L(B) = L(B') \wedge Act(B) \subseteq Act(B')\}.$$

Obviously, Γ_0 respects the labelling function L. It is easy to see that for partition pair $\langle \Sigma_i, \Gamma_i \rangle$ which is finer than $\langle \Sigma_0, \Gamma_0 \rangle$, Σ_i is consistent with respect to L, and Γ_i respects L as well.

In lines 11.5–11.15 of the algorithm, a finite sequence of partition pairs $\langle \Sigma_i, \Gamma_i \rangle$ with $i = 0, 1, \ldots, l$ is generated. We will show that it satisfies the following properties:

- Γ_i is acyclic for $i = 0, 1, \ldots, l$,
- $\langle \Sigma_i, \Gamma_i \rangle$ is coarser than $\langle \Sigma^\diamond, \Gamma^\diamond \rangle$ for $i = 0, 1, \ldots, l$,
- $\langle \Sigma_{i+1}, \Gamma_{i+1} \rangle$ is finer than $\langle \Sigma_i, \Gamma_i \rangle$ for $i = 0, 1, \ldots, l-1$,
- $\langle \Sigma_l, \Gamma_l \rangle = \langle \Sigma^\diamond, \Gamma^\diamond \rangle$.

The core task consists of how to refine the partition pair $\langle \Sigma_i, \Gamma_i \rangle$ satisfying the above conditions.

In the non-probabilistic setting, a space-efficient algorithm [54] is proposed for a directed graph $G = (V, E)$. A refinement operator[3] was used to generate the partition pair $\langle \Sigma_{i+1}, \Gamma_{i+1} \rangle$ from $\langle \Sigma_i, \Gamma_i \rangle$ satisfying all of the properties mentioned above. The refinement of blocks works as follows. For each block $B \in \Sigma_i$ let

$$E^{-1}(B) = \{s \in V \mid \exists s' \in B.(s, s') \in E\}$$

denote the set of predecessors of states in B. Then, using B' as a splitter, B is split into two part: $B_1 = B \cap E^{-1}(B')$ and $B_2 = B \setminus B_1$. The predecessor based method for splitting blocks, however, cannot be applied to the probabilistic setting. The reason is that in PAs states have successor distributions instead of a single successor state.

[2]Solving the GCPP does not depend on the initial state of the automaton. Thus, the initial state is also irrelevant for the discussion in this section.

[3]The refinement operator must guarantee that the refined partition relation Γ_{i+1} must be acyclic. Recently, van Glabbeek and Ploeger [111] have shown that the operator in [54] was flawed, and provided a non-trivial fix for the operator.

Algorithm 11 Quotient algorithm to decide $\langle \Sigma^\diamond, \Gamma^\diamond \rangle$ over $\mathcal{M}$.

SIMQUO($\mathcal{M}$)

11.1: $i \leftarrow 0$
11.2: $\Sigma_0 = \{\{s' \in S \mid L(s) = L(s') \wedge Act(s) = Act(s')\} \mid s \in S\}$
11.3: $\Gamma_0 \leftarrow \{(B, B') \in \Sigma_0 \times \Sigma_0 \mid L(B) = L(B') \wedge Act(B) \subseteq Act(B')\}$
11.4: **repeat**
11.5: $\quad \Sigma_{i+1} \leftarrow \emptyset$, $\Gamma_{i+1} \leftarrow \emptyset$
11.6: $\quad$ **for all** $B \in \Sigma_i$ **do**
11.7: $\quad\quad \Sigma_{i+1} \leftarrow \Sigma_{i+1} \cup \text{SPLIT}(B, \Sigma_i)$
11.8: $\quad \Gamma_{i+1} \leftarrow \{(Q, Q') \in \Sigma_{i+1} \times \Sigma_{i+1} \mid Par_{\Sigma_i}(Q) \neq Par_{\Sigma_i}(Q') \wedge (Par_{\Sigma_i}(Q), Par_{\Sigma_i}(Q')) \in \Gamma_i$ or $Par_{\Sigma_i}(Q) = Par_{\Sigma_i}(Q') \wedge Reach(Q, Q')\}$
11.9: $\quad$ Construct the $\exists$-quotient automaton $\exists\mathcal{M}/_{\Sigma_{i+1}}$
11.10: $\quad$ **repeat**
11.11: $\quad\quad$ **for all** $(Q, Q') \in \Gamma_{i+1}$ **do**
11.12: $\quad\quad\quad$ **if not** $(\forall Q \overset{\alpha}{\rightarrowtail} \pi_{\Sigma_{i+1}} \Rightarrow \exists Q' \xrightarrow{\alpha} \pi'_{\Sigma_{i+1}} \wedge \pi_{\Sigma_{i+1}} \sqsubseteq_{\Gamma_{i+1}} \pi'_{\Sigma_{i+1}})$ **then**
11.13: $\quad\quad\quad\quad \Gamma_{i+1} \leftarrow \Gamma_{i+1} \setminus \{(Q, Q')\}$
11.14: $\quad$ **until** Γ_{i+1} does not change
11.15: $\quad i{+}{+}$
11.16: **until** $\langle \Sigma_i, \Gamma_i \rangle = \langle \Sigma_{i-1}, \Gamma_{i-1} \rangle$

Moreover, the checking of the correspondence between distributions used for simulation involves weight functions which require additional attention. We propose a graph based analysis to refine the partition (lines 11.5–11.7) in Subsection 7.3.1. Then, we discuss how to refine the partition relation (lines 11.8–11.15) in Subsection 7.3.2.

7.3.1 Refinement of the Partition

Consider the partition pair $\langle \Sigma_i, \Gamma_i \rangle \in \Upsilon$ with $\langle \Sigma^\diamond, \Gamma^\diamond \rangle \ltimes \langle \Sigma_i, \Gamma_i \rangle$. The refinement operator SPLIT consists of finding a finer partition Σ_{i+1} which is stable with respect to $\langle \Sigma_i, \Gamma_i^* \rangle$. For $B \in \Sigma_i$, $\text{SPLIT}(B, \Sigma_i) = \{Q_1, \ldots, Q_k\}$ is a partition over B such that for all Q_i it should hold: if $Q_i \xrightarrow{\alpha} \pi_{\Sigma_i}$, there exists $Q_i \overset{\alpha}{\rightarrowtail} \pi'_{\Sigma_i}$ such that $\pi_{\Sigma_i} \sqsubseteq_{\Gamma_i^*} \pi'_{\Sigma_i}$ (cf. Definition 7.2.5). To construct this partition, we first construct the exists-quotient automaton $\exists\mathcal{M}/_{\Sigma_i}$, and then start with the following partition of B:

$$V_B = \{\{s' \in S \mid \forall \alpha \in Act(s).\ Steps_{\Sigma_i,\alpha}(s) = Steps_{\Sigma_i,\alpha}(s')\} \mid s \in S\} \tag{7.2}$$

By the construction of the set V_B, the $\forall$-transitions and $\exists$-transitions of $Q \in V_B$ with respect to the partition Σ_i coincide, i.e., $Q \overset{\alpha}{\rightarrowtail} \pi_{\Sigma_i}$ if and only if $Q \xrightarrow{\alpha} \pi_{\Sigma_i}$. The partition V_B is finer than the partition for B we are searching for. We construct now a graph $G_B = (V_B, E_B)$ for the block B, in which for $Q, Q' \in V_B$, we add the edge $(Q, Q') \in E_B$ if the following condition holds:

$$\forall \pi_{\Sigma_i} \in Steps_{\Sigma_i,\alpha}(Q).\ \exists \pi'_{\Sigma_i} \in Steps_{\Sigma_i,\alpha}(Q').\ \pi_{\Sigma_i} \sqsubseteq_{\Gamma_i} \pi'_{\Sigma_i} \tag{7.3}$$

The above condition is checked in the $\exists$-quotient automaton with respect to the partition Σ_i. The condition $\pi_{\Sigma_i} \sqsubseteq_{\Gamma_i} \pi'_{\Sigma_i}$ can be checked via maximum flow computations [9]. We obtain the partition $\textsc{Split}(B, \Sigma_i)$ by constructing the *maximal strongly connected components* (SCCs) of G_B. Let $\textsc{Split}(B, \Sigma_i)$ denote the partition for B obtained by contracting the SCCs of G_B:

$$\textsc{Split}(B, \Sigma_i) = \{\cup_{X \in C} X \mid C \text{ is an SCC of } G_B\} \tag{7.4}$$

Moreover, as in Algorithm SimQuo, let $\Sigma_{i+1} = \cup_{B \in \Sigma_i} \textsc{Split}(B, \Sigma_i)$. We first prove that the initial partition relation is coarser than $\Gamma^\diamond$.

Lemma 7.3.1. *Let Σ_0 and Γ_0 as defined above. It holds that $\langle \Sigma^\diamond, \Gamma^\diamond \rangle \ltimes \langle \Sigma_0, \Gamma_0 \rangle$.*

Proof. First we show that $\Sigma^\diamond$ is finer than Σ_0. Let $B \in \Sigma^\diamond$. For all $s, s' \in B$, by definition of $\Sigma^\diamond$ we have that $s \simeq s'$, which implies that $L(s) = L(s')$ and $Act(s) = Act(s')$. Thus they belong to the same block in Σ_0 by construction of Σ_0, which implies that $\Sigma^\diamond$ is finer than Σ_0. It remains to prove that $\Gamma^\diamond$ is finer than Γ_0, i.e., $\precsim_{\langle \Sigma^\diamond, \Gamma^\diamond \rangle} \subseteq \precsim_{\langle \Sigma_0, \Gamma_0 \rangle}$. Let $(s, s') \in \precsim_{\langle \Sigma^\diamond, \Gamma^\diamond \rangle}$, which implies that $s \precsim s'$. By definition, we have $L(s) = L(s')$ and $Act(s) \subseteq Act(s')$. Thus, by the construction of Γ_0, $(s, s') \in \precsim_{\langle \Sigma_0, \Gamma_0 \rangle}$. ■

Lemma 7.3.2. *Let $\langle \Sigma, \Gamma \rangle \in \Upsilon$ with $\langle \Sigma^\diamond, \Gamma^\diamond \rangle \ltimes \langle \Sigma, \Gamma \rangle$. Then, $\mu \sqsubseteq_{\precsim} \mu'$ implies that $lift_\Sigma(\mu) \sqsubseteq_\Gamma lift_\Sigma(\mu')$.*

Proof. Assume that $\mu \sqsubseteq_{\precsim} \mu'$ and let Δ denote the corresponding weight function. We define $\Delta_\Sigma : \Sigma_\perp \times \Sigma_\perp \to [0, 1]$ by:

$$\Delta_\Sigma(B, B') = \sum_{s \in B} \sum_{s' \in B'} \Delta(s, s').$$

Assume that $\Delta_\Sigma(B, B') > 0$ and assume that $B \neq \perp$. There must exist $s \in B, s' \in B'$ with $\Delta(s, s') > 0$ which implies that $s \precsim s'$. It holds that $\precsim = \precsim_{\langle \Sigma^\diamond, \Gamma^\diamond \rangle} \subseteq \precsim_{\langle \Sigma, \Gamma \rangle}$. By the definition of $\precsim_{\langle \Sigma^\diamond, \Gamma^\diamond \rangle}$ and $\precsim_{\langle \Sigma, \Gamma \rangle}$, it holds that $B \Gamma B'$. Thus the first condition of weight function holds. To show the second condition let $B \in \Sigma_\perp$. Then we have $lift_\Sigma(\mu)(B) = \mu(B) = \sum_{s \in B} \Delta(s, S)$. Thus, it holds that

$$lift_\Sigma(\mu)(B) = \sum_{s \in B} \Delta(s, S) = \sum_{s \in B} \sum_{B' \in \Sigma_\perp} \sum_{s' \in B'} \Delta(s, s') = \sum_{B' \in \Sigma_\perp} \Delta_\Sigma(B, B')$$

The third condition for weight function can be shown in a similar way. Thus Δ_Σ is a weight function for $(lift_\Sigma(\mu), lift_\Sigma(\mu'))$ with respect to Γ. ■

The following lemma shows that the obtained partition Σ_{i+1} is coarser than $\Sigma^\diamond$.

Lemma 7.3.3. *For all $i \geq 0$, it holds that: Σ_{i+1} is finer than Σ_i. Moreover, $\Sigma^\diamond$ is finer than Σ_i.*

Proof. Since Σ_{i+1} is obtained from Σ_i by refining blocks in Σ_i, it is finer than Σ_i for all $i \geq 0$. Now we show that $\Sigma^\diamond$ is finer than Σ_i by induction on i. The basis follows directly from Lemma 7.3.1. For the induction step assume that the statement holds for

i. By induction hypothesis, there exists $B \in \Sigma_i$ with $s, s' \in B$. It is sufficient to show that for all $s, s' \in S$ with $s \simeq s'$ there exists $Q \in \Sigma_{i+1}$ with $s, s' \in Q$.

If it holds that $\mathit{Steps}_{\Sigma_i,\alpha}(s) = \mathit{Steps}_{\Sigma_i,\alpha}(s')$ for all $\alpha \in \mathit{Act}(s)$, they must belong to a sub-block of B by construction of the graph G_B. Assume the other case that $\mathit{Steps}_{\Sigma_i,\alpha}(s) \neq \mathit{Steps}_{\Sigma_i,\alpha}(s')$ for some $\alpha \in \mathit{Act}(s)$. In this case, there exists $Q, Q' \in V_B$ with $Q \neq Q'$ and $s \in Q, s' \in Q'$. Assume that $\pi_{\Sigma_i} \in \mathit{Steps}_{\Sigma_i,\alpha}(Q)$. By the definition of V_B, we have that $s \xrightarrow{\alpha} \pi_{\Sigma_i}$. Let $s \xrightarrow{\alpha} \mu$ with $\pi_{\Sigma_i} = \mathit{lift}_{\Sigma}(\mu)$. Since $s \precsim s'$, there exist $s' \xrightarrow{\alpha} \mu'$ with $\mu \sqsubseteq_{\precsim} \mu'$. Let $\pi'_{\Sigma_i} = \mathit{lift}_{\Sigma_i}(\mu')$. By Lemma 7.3.2, we have that $\pi_{\Sigma_i} \sqsubseteq_{\Gamma_i} \pi'_{\Sigma_i}$. Condition 7.3 is satisfied, thus $(Q, Q') \in E_B$. Similarly, we have also that $(Q', Q) \in E$. Thus Q, Q' belong to the same SCC which implies that s, s' are in the same sub-block of Σ_{i+1} because of Equation 7.4. ■

We show that for acyclic relation R, the relation $\sqsubseteq_R$ is anti-symmetric:

Lemma 7.3.4. *Assume that $R \subset S \times S$ is acyclic and let $\mu, \mu' \in \mathit{Dist}(S)$ with $\mu(S) > 0$. If $\mu \sqsubseteq_R \mu'$ and $\mu' \sqsubseteq_R \mu$, then $\mu = \mu'$.*

Proof. Consider the infinite chain $\mu \sqsubseteq_R \mu' \sqsubseteq_R \mu \sqsubseteq_R \mu' \ldots$. Let $\Delta_1, \Delta_2, \Delta_3, \ldots$ be the corresponding weight functions. We construct an infinite sequence of states as follows. Let s_1 be an arbitrary state in $\mathit{Supp}(\mu)$. Hence, it holds that $0 < \mu(s_1) = \Delta(s_1, S)$. Let $s_2 \in \mathit{Supp}(\mu')$ such that $\Delta_1(s_1, s_2) > 0$. Repeating this procedure we get the infinite sequence $s_1, s_2, s_3, \ldots$ satisfying the property: $\Delta_i(s_i, s_{i+1}) > 0$ for all $i = 1, 2, \ldots$. By definition of weight function, we have that $(s_i, s_{i+1}) \in R$ for all $i = 1, 2, \ldots$. For odd index i it holds that $s_i \in \mathit{Supp}(\mu)$, and for even index i it holds that $s_i \in \mathit{Supp}(\mu')$. Without loss of generality, we assume that $|\mathit{Supp}(\mu)| \leq |\mathit{Supp}(\mu')|$. Let $k = 2|\mathit{Supp}(\mu)|$. The acyclicity of R guarantees that $s_k = s_{k+1} = \ldots$. This holds for all such infinite sequences, which implies that for $j \geq k$, $\Delta_j(s, s') = 0$ if $s \neq s'$. This implies that we must have that $\Delta_j(s, s) = \mu(s) = \mu(s')$ because of the definition of weight function, thus $\mu = \mu'$. ■

The following lemma shows that, for acyclic Γ_i, the partition Σ_{i+1} is stable with respect to $\langle \Sigma_i, \Gamma_i^* \rangle$:

Lemma 7.3.5. *Assume that Γ_i is a acyclic. For all $i \geq 0$, Σ_{i+1} is stable with respect to $\langle \Sigma_i, \Gamma_i^* \rangle$.*

Proof. Let $B \in \Sigma_i$, and let $\textsc{Split}(B, \Sigma_i)$ be the partition of B as defined above. It is sufficient to show that all $Q \in \textsc{Split}(B, \Sigma_i)$ is stable with respect to $\langle \Sigma_i, \Gamma_i^* \rangle$. Let $G_B = (V_B, E_B)$ the graph constructed for the block B, and let $Q \in \textsc{Split}(B, \Sigma_i)$. By construction Q is an SCC in G_B. Thus, we may assume there is a loop $(Q_1 \ldots, Q_n)^\omega$ with $(Q_i, Q_{i+1}) \in E_B$ for $i = 1, \ldots, n-1$ and $(Q_n, Q_1) \in E_B$ and $Q = \cup_{i=1}^n Q_i$. Let $Q \xrightarrow{\alpha} \pi_1$ with $\pi_1 \in \mathit{Steps}_{\Sigma_i,\alpha}(Q)$, it suffices to show that there exists $\pi \in \mathit{Steps}_{\Sigma_i,\alpha}(Q)$ with $Q \overset{\alpha}{\rightarrowtail} \pi$ such that $\pi_1 \sqsubseteq_{\Gamma_i^*} \pi$. Without loss of generality, let $Q_1 \xrightarrow{\alpha} \pi$. By construction of the edges of the graph there must exists an infinite sequence of distributions $\pi_2, \pi_3, \ldots$ with $\pi_k \in \mathit{Steps}_{\Sigma_i,\alpha}(Q_j)$ provided that $k \mod n = j$ such that it holds: $\pi_j \sqsubseteq_{\Gamma_i} \pi_{j+1}$ for

all $j \geq 1$. Since the set of α-successor distributions of Q_1 is finite, there exists a number l with $l \mod n = 1$ such that $\pi_l = \pi_{l+n}$ and that

$$\pi_l \sqsubseteq_{\Gamma_i} \pi_{l+1} \ldots \sqsubseteq_{\Gamma_i} \pi_{l+n} = \pi_l.$$

By Lemma 4.1.3, Γ_i^* is transitive implies that $\sqsubseteq_{\Gamma_i^*}$ is also transitive. Thus $\pi_j \sqsubseteq_{\Gamma_i^*} \pi_{j'}$ for all $j, j' \in \{l, \ldots, l+n\}$. Recall that Γ_i is by assumption acyclic. This implies that Γ_i^* is also acyclic. By applying Lemma 7.3.4 we have that $\pi_j = \pi_{j'}$ for all $j, j' \in \{l, \ldots, l+n\}$. Hence we have found the desired distribution $\pi = \pi_l$. ■

7.3.2 Refinement of the Partition Relations

Similar to the refinement of partitions, at the end of iteration i, we aim to get the partition relation Γ_{i+1} which is finer than Γ_i, but still coarser than $\Gamma^\diamond$. At line 11.8, Γ_{i+1} is initialised such that it contains (Q, Q') if

- either Q, Q' have different parent blocks $B \neq B'$ with $B = Par_{\Sigma_i}(Q)$ and $B' = Par_{\Sigma_i}(Q')$ such that $(B, B') \in \Gamma_i$ holds,
- or they have same parent block B and the SCC for Q' can be reached by the SCC for Q in the graph G_B. This constraint is abbreviated by $Reach(Q, Q')$ (line 11.8).

To get a coarser partition relation, we want to remove from Γ_{i+1} those pairs (B, B') satisfying the condition: no state in B can be simulated by any state in B'. Conversely, we want to keep those pairs (B, B') satisfying the condition that there exists at least a state in B which can be simulated by at least another state in B'. This condition, however, depends on the concrete transitions of state $s \in B$. To be able to work completely on the quotient automaton $\exists\mathcal{M}/_{\Sigma_{i+1}}$, we consider the weakness of the above condition:

$$\forall B \overset{\alpha}{\rightarrowtail} \pi_{\Sigma_{i+1}} \Rightarrow \exists B' \overset{\alpha}{\rightarrow} \pi'_{\Sigma_{i+1}} \wedge \pi_{\Sigma_{i+1}} \sqsubseteq_{\Gamma_{i+1}} \pi'_{\Sigma_{i+1}} \tag{7.5}$$

Intuitively, if there is a $\forall$-transition out of B, we require that this transition must be simulated by at least an $\exists$-transition out of B'. Note the similarity to Condition 7.3: we consider only transitions of the form $B \overset{\alpha}{\rightarrowtail} \pi_{\Sigma_{i+1}}$ from B (line 11.12 in SimQuo). Again, the condition $\pi_{\Sigma_{i+1}} \sqsubseteq_{\Gamma_{i+1}} \pi'_{\Sigma_{i+1}}$ could be checked via maximum flow computations [9].

Lemma 7.3.6. *For all $i \geq 0$, the partition relation Γ_{i+1} is finer than Γ_i. Moreover, $\Gamma^\diamond$ is finer than Γ_i.*

Proof. By Lemma 7.3.3, Σ_{i+1} is finer than Σ_i. Initially Γ_{i+1} is finer than Γ_i. During the algorithm only pairs are removed from Γ_{i+1}, it remains finer than Γ_i. We prove now that $\Gamma^\diamond$ is finer than the partition relation Γ_i by induction on i. The basic case $i = 0$ follows from Lemma 7.3.1. Assume that the statement holds for i: $\Gamma^\diamond$ is finer than Γ_i. We show that $\Gamma^\diamond$ is finer than Γ_{i+1} in two steps: (i) after the initialisation at line 11.8, $\Gamma^\diamond$ is finer than Γ_{i+1}, and (ii) at the end of the inside repeat-loop (lines 11.10–11.14), $\Gamma^\diamond$ is finer than Γ_{i+1}.

We first show (i). Let $(Q, Q') \in \Gamma^\diamond$, by induction hypothesis, there must exists $(B_i, B'_i) \in \Gamma_i$ with $Q \subseteq B_i$ and $Q' \subseteq B'_i$. By Lemma 7.3.3, there exists $B_{i+1}, B'_{i+1} \in \Sigma_{i+1}$

such that $Q \subseteq B_{i+1} \subseteq B_i$ and $Q' \subseteq B'_{i+1} \subseteq B'_i$. It is sufficient to show that $(B_{i+1}, B'_{i+1}) \in \Gamma_{i+1}$ at line 11.15. Depending on whether the parent blocks B_i and B'_i are the same, we have two cases. The case $B_i \neq B'_i$ is simple, as it holds that $B_i = Par_{\Sigma_i}(B_{i+1}), B'_i = Par_{\Sigma_i}(B'_{i+1})$ and $(B_i, B'_i) \in \Gamma_i$, which implies that $(B_{i+1}, B'_{i+1}) \in \Gamma_{i+1}$. Now we consider the case $B_i = B'_i$. We show that the SCC representing B'_{i+1} can be reached by the SCC representing B_{i+1} in the graph $G_{B_i} = (V_{B_i}, E_{B_i})$. There must exists $X \in V_{B_i}$ with $X \subseteq B_{i+1}$ such that $X \cap Q \neq \emptyset$. Assume that $\pi_{\Sigma_i} \in Steps_{\Sigma_i,\alpha}(X)$. By the definition of V_{B_i}, for $s \in X \cap Q$ we have that $s \xrightarrow{\alpha} \mu$ with $\mu \in Steps_\alpha(s)$ and $\pi_{\Sigma_i} = lift_{\Sigma_i}(\mu)$. Let $\mu \in Steps_\alpha(s)$ with $\pi_{\Sigma_i} = lift_{\Sigma_i}(\mu)$. Let s' be an arbitrary state in Q'. Since $(Q, Q') \in \Gamma^\diamond$, it holds that $s \precsim s'$, which implies that there exists $s' \xrightarrow{\alpha} \mu'$ with $\mu \sqsubseteq_{\precsim} \mu'$. Let $\pi'_{\Sigma_i} = lift_{\Sigma_i}(\mu')$. By induction hypothesis, $\Gamma^\diamond$ is finer than Γ_i. Applying Lemma 7.3.2, we get $\pi_{\Sigma_i} \sqsubseteq_{\Gamma_i} \pi'_{\Sigma_i}$. Let X' be the unique element of V_{Σ_i} containing s'. Thus, $X' \subseteq B'_{i+1}$ and $\pi'_{\Sigma_i} \in Steps_{\Sigma_i,\alpha}(X')$, which implies that $(X, X') \in E_{B_i}$. Thus, X' is reachable from X. Hence $Reach(B_{i+1}, B'_{i+1})$ holds.

Now we show the part (ii). Because of (i), we may assume that $\Gamma^\diamond$ is finer than Γ_{i+1} at the beginning of the inside repeat-loop. We show that the invariant holds for this loop: $\Gamma^\diamond$ is finer than Γ_{i+1}. Let (Q, Q') be an arbitrary element in $\Gamma^\diamond$, and let $(B_{i+1}, B'_{i+1}) \in \Gamma_{i+1}$ with $Q \subseteq B_{i+1}$ and $Q' \subseteq B'_{i+1}$. Similar to (i), it is sufficient to show that (B_{i+1}, B'_{i+1}) will still be in Γ_{i+1} line 11.15. We show that (B_{i+1}, B'_{i+1}) satisfies Condition 7.5. Assume that $B_{i+1} \xrightarrow{\alpha} \pi_{\Sigma_{i+1}}$. Then, for $s \in Q$, we have that $s \xrightarrow{\alpha} \pi_{\Sigma_{i+1}}$, and let $\mu \in Steps_\alpha(s)$ with $\pi_{\Sigma_{i+1}} = lift_{\Sigma_{i+1}}(\mu)$. Let s' be an arbitrary state in Q'. Since $s \precsim s'$, there exists $s' \xrightarrow{\alpha} \mu'$ with $\mu \sqsubseteq_{\precsim} \mu'$. Let $\pi'_{\Sigma_{i+1}} = lift_{\Sigma_{i+1}}(\mu')$. Since $\Gamma^\diamond$ is finer than Γ_{i+1}, applying Lemma 7.3.2, we get $\pi_{\Sigma_{i+1}} \sqsubseteq_{\Gamma_{i+1}} \pi'_{\Sigma_{i+1}}$, thus Condition 7.5 holds. ■

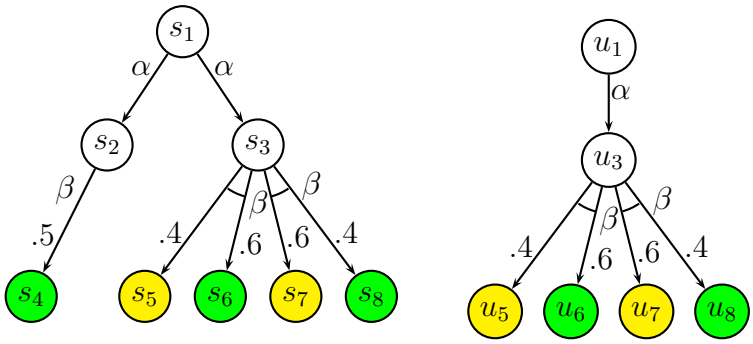

Figure 7.5: A PA for illustration of the algorithm.

Example 7.3.1. As our running example we consider the PA depicted in Figure 7.5 (cf. Example 7.2.1). Initially, we have the partition pair $\langle \Sigma_0, \Gamma_0 \rangle$ with the partition $\Sigma_0 = \{B_1, B_2, B_4, B_5\}$ where $B_1 = \{s_1, u_1\}$, $B_2 = \{s_2, s_3, u_3\}$, $B_4 = \{s_4, s_6, s_8, u_6, u_8\}$ and $B_5 = \{s_5, s_7, u_5, u_7\}$, and the partition relation $\Gamma_0 = \mathcal{I}(\Sigma_0)$ which is the identical relation over Σ_0. At the beginning of the first iteration, the refined partition B_1 will be constructed. Blocks B_4 and B_5 contain only absorbing states, thus can not be refined in the first iteration. The block B_1 remains also the same as each state $s \in B_1$ has an α-successor distribution leading to block B_2 with probability 1. The block B_2 will

then be refined into two sub-blocks $Q_1 = \{s_2\}$, and $Q_2 = \{s_3, s_4\}$, which leads to the partition $\Sigma_1 = \{B_1, Q_1, Q_2, B_4, B_5\}$. The corresponding partition relation Γ_1 after line 11.8 is $\mathcal{I}(\Sigma_1) \cup \{(Q_1, Q_2)\})$. This partition relation can be shown to be stable (cf. Example 7.2.1), thus $\langle \Sigma_1, \Gamma_1 \rangle = \langle \Sigma^\diamond, \Gamma^\diamond \rangle$.

7.3.3 Correctness

In this section we show the correctness of the algorithm SimQuo. By Lemmata 7.3.3 and 7.3.6, we see that the partition pair $\langle \Sigma_{i+1}, \Gamma_{i+1} \rangle$ obtained in the algorithm is finer than $\langle \Sigma_i, \Gamma_i \rangle$, and coarser than $\langle \Sigma^\diamond, \Gamma^\diamond \rangle$. The following lemma shows that the partition relation Γ_i is acyclic:

Lemma 7.3.7 (Acyclicity)**.** *For all $i \geq 0$, the partition relation Γ_i is acyclic.*

Proof. We prove by induction on i. In the first iteration the statement holds: since the inclusion relation $\subseteq$ is transitive, no cycles except self loop exist in Γ_0. Now consider iteration i. By induction hypothesis assume that the partition relation Γ_i at the beginning of iteration i is acyclic. We shall show that Γ_{i+1} is acyclic until the end of i-te iteration. Consider the initial value of Γ_{i+1} at line 11.8 at iteration i. At this position we may still assume that Γ_i is acyclic by induction hypothesis. This implies that during the initialisation of Γ_{i+1} only sub-blocks from some same parent block $B \in \Sigma_i$ can form cycles of length $n > 1$. Assume such a cycle is formed from B: $Q_1 \Gamma_{i+1} Q_2 \Gamma_{i+1} \ldots, Q_n \Gamma_{i+1} Q_1 \ldots$ with $n > 1$. Since $Q_1 \Gamma_{i+1} Q_2$ implies that $\mathit{Reach}(Q_1, Q_2)$ and the reachability is transitive, we get that $Q_1, \ldots, Q_n$ must belong to the same SCC in G_B which is a contradiction. Thus, Γ_{i+1} is acyclic after initialisation. Since afterwards pairs will only be removed from Γ_{i+1}, it remains acyclic. ■

Theorem 7.3.8 (Correctness)**.** *Assume that* SimQuo *terminates at iteration l, then, $\langle \Sigma^\diamond, \Gamma^\diamond \rangle = \langle \Sigma_l, \Gamma_l \rangle$.*

Proof. By termination we have that $\langle \Sigma_l, \Gamma_l \rangle = \langle \Sigma_{l+1}, \Gamma_{l+1} \rangle$. By Lemma 7.3.7 the partition relation Γ_{l+1} is acyclic. Applying Lemma 7.3.5 we have that Σ_{l+1} is stable with respect to $\langle \Sigma_l, \Gamma_l^* \rangle$ which implies that Σ_l is stable with respect to $\langle \Sigma_l, \Gamma_l^* \rangle$. We first prove that the partition pair $\langle \Sigma_l, \Gamma_l^* \rangle$ is stable. Let $(B, B') \in \Gamma_l^*$, and $B \xrightarrow{\alpha} \pi_1$ with $\pi_1 \in \mathit{Dist}(\Sigma_l)$. Since Σ_l is stable with respect to $\langle \Sigma_l, \Gamma_l^* \rangle$, there must exist $\pi_1' \in \mathit{Dist}(\Sigma_l)$ with $B \overset{\alpha}{\rightarrowtail} \pi_1'$ such that $\pi_1 \sqsubseteq_{\Gamma_l^*} \pi_1'$. Since $(B, B') \in \Gamma_l^*$, there is a sequence $B_1, \ldots, B_n$ such that $B_1 \Gamma_l B_2 \Gamma_l \ldots B_n$ with $B_1 = B$ and $B_n = B'$ and $n \geq 2$. Σ_l is stable with respect to $\langle \Sigma_l, \Gamma_l^* \rangle$ implies that the block B_i is stable with respect to $\langle \Sigma_l, \Gamma_l^* \rangle$ for all $i = 1, \ldots, n$. Moreover, pairs in Γ_l satisfy Condition 7.5. Thus there exists distributions $\pi_i, \pi_i' \in \mathit{Dist}(\Sigma_l)$ for $i = 1, \ldots, n$ and such that it holds $B_i \xrightarrow{\alpha} \pi_i$, $B_i \overset{\alpha}{\rightarrowtail} \pi_i'$, and:

$$\pi_1 \sqsubseteq_{\Gamma_l^*} \pi_1' \sqsubseteq_{\Gamma_l} \pi_2 \sqsubseteq_{\Gamma_l^*} \pi_2' \sqsubseteq_{\Gamma_l} \ldots \pi_n \sqsubseteq_{\Gamma_l^*} \pi_n'$$

By Lemma 4.1.3, Γ_l^* is transitive implies that $\sqsubseteq_{\Gamma_l^*}$ is also transitive. Thus we have that $\pi_1 \sqsubseteq_{\Gamma_l^*} \pi_n'$ which implies that the partition pair $\langle \Sigma_l, \Gamma_l^* \rangle$ is stable. By Lemma 7.2.4 we have that $\langle \Sigma_l, \Gamma_l^* \rangle \ltimes \langle \Sigma^\diamond, \Gamma^\diamond \rangle$. By Lemmata 7.3.3 and 7.3.6 we have that $\langle \Sigma^\diamond, \Gamma^\diamond \rangle \ltimes \langle \Sigma_l, \Gamma_l \rangle$. Hence, $\langle \Sigma^\diamond, \Gamma^\diamond \rangle = \langle \Sigma_l, \Gamma_l \rangle$. ■

7.3.4 On Acyclicity

In the proof of the correctness of the algorithm, we used Lemma 7.3.7 which states that the partition relation Γ_i is acyclic for all i. In [54], a partition refinement based space efficient algorithm is introduced for computing simulation for labelled transition systems (the labels are coded as initial partition of states). The author claim that the partition relation of their algorithm is acyclic. However, van Glabbeek and Ploeger [111] have shown that the analysis in [54] is flawed, and provided a nontrivial fix. In this section we apply our algorithm to some examples presented in [111]. Recall that PAs subsume labelled transition systems, thus our algorithm can be applied for labelled transition systems directly.

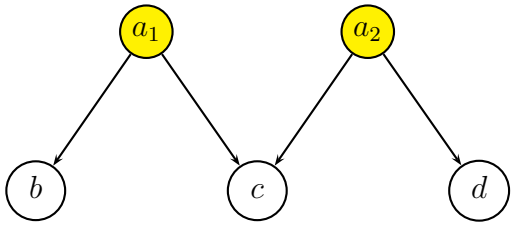

Figure 7.6: A transition system.

Counterexample 1 in [111]. Consider a part of the transition system depicted in Figure 7.6. Assume that every transition is labelled with the same action which is omitted in the figure. Assume that at the beginning of some iteration we have the partition pair $\langle \Sigma, \Gamma \rangle$ with $\Sigma = \{\alpha, \beta, \gamma, \delta\}$ with $\alpha = \{a_1, a_2\}$, $\beta = \{b\}$, $\gamma = \{c\}$ and $\delta = \{d\}$. The partition relation is given by: $\Gamma = \mathcal{I}(\Sigma) \cup \{(\beta, \delta), (\delta, \gamma)\}$. As shown in [111], the refined partition pair $\langle \Sigma', \Gamma' \rangle$ according to [54] is: $\Sigma' = \{\alpha_1, \alpha_2, \beta, \gamma, \delta\}$ with $\alpha_1 = \{a_1\}$ and $\alpha_2 = \{a_2\}$, and a cyclic partition relation $\Gamma' = \mathcal{I}(\Sigma') \cup \{(\alpha_1, \alpha_2), (\alpha_2, \alpha_1), (\beta, \delta), (\delta, \gamma)\}$.

Now we apply our algorithm to this example. The block α is the only block which can be refined. We construct the graph G_α for the block α: observe that $\mathit{Steps}(a_1) = \{\beta, \gamma\}$ and $\mathit{Steps}(a_2) = \{\gamma, \delta\}$. Thus, $V_\alpha = \{\{a_1\}, \{a_2\}\}$. The constraint 7.3 holds for both the pair $(\alpha_1, \alpha_2\}$ and (α_2, α_1), thus we have the set of edges $E_\alpha = \{(\alpha_1, \alpha_2), (\alpha_2, \alpha_1)\}$. Thus the two nodes in V_α form an SCC, and will be contracted together. Thus in this iteration step, the block α will not be split.

The partition relation Γ is not transitive: (β, δ) is not in the partition relation, but is in the closure Γ^*. Since the partition relation $\Gamma^\diamond$ in the partition pair $\langle \Sigma^\diamond, \Gamma^\diamond \rangle$ transitive, and the partition pair $\langle \Sigma, \Gamma \rangle$ is coarser than $\langle \Sigma^\diamond, \Gamma^\diamond \rangle$ (cf. Lemmata 7.3.3 and 7.3.6), in some later iteration either (β, δ) or (δ, γ) will be thrown out of the partition relation. After that, the block α will be split into two blocks α_1 and α_2. Unfortunately, we could not use the knowledge, that one of the pair in Γ will be thrown out of it in a later iteration, to split the block α in the current iteration: since we would then get a cyclic partition relation.

In the above example, we have assumed that in some iteration we get an partition relation Γ which is not transitive. Again, we take an example from [111] to illustrate that this is possible.

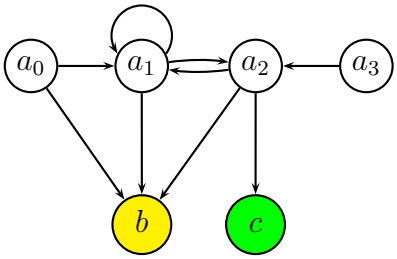

Figure 7.7: A transition system for illustrating that the partition relation can be non-transitive.

Example 3 in [111]. Consider the transition system depicted in Figure 7.7. The initial partition pair is $\langle \Sigma_0, \Gamma_0 \rangle$ where $\Sigma_0 = \{\alpha, \beta, \gamma\}$ with $\alpha = \{a_0, a_1, a_2, a_3\}$, $\beta = \{b\}$, $\gamma = \{c\}$. The initial partition relation is the identical relation $\mathcal{I}(\Sigma_0)$ over Σ_0. Since $\mathit{Steps}(a_0) = \mathit{Steps}(a_1) = \{\alpha, \beta\}$, $\mathit{Steps}(a_2) = \{\alpha, \beta, \gamma\}$ and $\mathit{Steps}(a_3) = \{\alpha\}$, the set $V_\alpha = \{\alpha_1, \alpha_2, \alpha_3\}$ with $\alpha_1 = \{a_0, a_1\}, \alpha_2 = \{a_2\}$, $\alpha_3 = \{a_3\}$. The set of edges is given by $E_\alpha = \{(\alpha_1, \alpha_2), (\alpha_3, \alpha_1), (\alpha_3, \alpha_2)\}$. No nodes of the graph can be contracted, thus we get the partition $\Sigma_1 = \{\alpha_1, \alpha_2, \alpha_3, \beta, \gamma\}$. The partition relation is $\Gamma_1 = \mathcal{I}(\Sigma_1) \cup \{(\alpha_1, \alpha_2), (\alpha_3, \alpha_1), (\alpha_3, \alpha_2)\}$. Then, in the inside repeat loop, the pair (α_3, α_2) is thrown out of the partition relation: since α_3 has an $\exists$-transition to α_2, but α_2 does not have a successor block which can simulate α_2 with respect to Γ_1. Let $\Gamma_1' = \Gamma_1 \setminus \{(\alpha_3, \alpha_2)\}$ denote the partition relation at the end of the first loop. Thus, at the starting of the second loop, we start with a partition relation which is not transitive.

7.3.5 Complexity

The following lemma shows that the number of iterations of the algorithm SimQuo is linear in $|\Sigma^\diamond|$:

Lemma 7.3.9. *Assume that* SimQuo *terminates at iteration l, then, $l \in \mathcal{O}(|\Sigma^\diamond|)$.*

Proof. By Lemma 7.3.3 we have that Σ_{i+1} is finer than Σ_i. It is sufficient to show that for[4] $1 \leq i \leq l-1$, Σ_{i+1} is strictly finer than Σ_i. For the sake of contradiction, assume that $\Sigma_i = \Sigma_{i+1}$. In this case, we have that at line 11.8 the initial value of Γ_{i+1} is equal to Γ_i. Between lines 11.10–11.14, since $\Sigma_{i+1} = \Sigma_i$ and $\Gamma_{i+1} = \Gamma_i$, the algorithm would terminate at iteration i, implying that $i = l$ which is a contradiction. ■

For the complexity analysis, we introduce some notations. As before, we let n, m denote the number of states and transitions of $\mathcal{M}$ respectively. We let $\Sigma_\sim$ denote the partition induced by the bisimulation relation $\sim$, and let $n_\sim$ and $m_\sim$ denote the number of states and transitions of the $\exists$-quotient[5] automaton $\exists\mathcal{M}/_{\Sigma_\sim}$. Let $n_\diamond$ and $m_\diamond$ denote the number of states and transitions of the quotient automaton $\exists\mathcal{M}/_{\Sigma^\diamond}$.

[4] As shown in Example 7.3.1, in the first iteration it could be the case that $\Sigma_1 = \Sigma_0$ but $\Gamma_1 \neq \Gamma_0$.

[5] In fact, the $\exists$-quotient automaton and the $\forall$-quotient automaton with respect to the bisimulation relation $\sim$ coincide.

Theorem 7.3.10. *The algorithm* SimQuo *has time complexity* $\mathcal{O}(mn_\diamond + m_\diamond^2 n_\diamond^4 + m_\sim^2 n_\diamond^2)$, *and space complexity* $\mathcal{O}(n_\sim^2 + n \log n_\diamond)$.

Proof. Let $\langle \Sigma_i, \Gamma_i \rangle$ denote the partition pair at the beginning of iteration i, and let n_i and m_i denote the number of states and transitions of the $\exists$-quotient automaton $\exists\mathcal{M}/_{\Sigma_i}$. In this iteration this partition pair is refined to $\langle \Sigma_{i+1}, \Gamma_{i+1} \rangle$.

We first consider the time and space needed for refining the partition Σ_i in iteration i. For $B \in \Sigma_i$, the graph $G_B = (V_B, E_B)$ is constructed where the set of vertices V_B and the set of edges E_B is constructed according to Equations 7.2 and 7.3 respectively. For $Q, Q' \in V_B$, an edge between Q and Q' is added if it holds that for all $\pi \in \mathit{Steps}_{\Sigma_i,\alpha}(Q)$ there exists $\pi' \in \mathit{Steps}_{\Sigma_i,\alpha}(Q')$ such that $\pi \sqsubseteq_{\Gamma_i} \pi'$. For this a bipartite network $\mathcal{N}(\pi, \pi', \Gamma_i)$ from (π, π') with respect to Γ_i is constructed. Then, by Lemma 4.2.1, the condition holds iff the maximum flow of the network has value 1. In $\mathcal{N}(\pi, \pi', \Gamma_i)$ the vertices can be partitioned into two subsets V_1 and V_2 such that all edges have one endpoint in V_1 and another in V_2. The complexity of this operation [3] is $\mathcal{O}(|V_1||V_2|^2)$. Recall that $|V_1|$ is linear in the order of $|\pi|$, and $|V_2|$ is linear in the order of $|\pi'|$. Moreover, both $|\pi|$ and $|\pi'|$ are in the order of $\mathcal{O}(n_i)$. Consider the partition $\Sigma^* := \cup_{B \in \Sigma_i} V_B$ of S and the quotient automaton $\mathcal{M}/_{\Sigma^*}$. By the definition of strong bisimulation, it is easy to show inductively that Σ^* is coarser than $\Sigma_\sim$. Thus, all of the checks corresponding to the construction of the sets E_B for all $B \in \Sigma_i$ (cf. Condition 7.3) take time:

$$\sum_{B \in \Sigma_i} \sum_{Q \in V_B} \sum_{Q' \in V_B} \sum_{\alpha \in Act(Q)} \sum_{\pi \in \mathit{Steps}_{\Sigma_i,\alpha}(Q)} \sum_{\pi' \in \mathit{Steps}_{\Sigma_i,\alpha}(Q')} |\pi| \, |\pi'|^2 \leq m_\sim^2 n_i$$

The time needed for constructing the SCCs for all G_B is in the order of $\mathcal{O}(m_\sim)$ [110]. The space needed for this phase is $\mathcal{O}(n_\sim^2 + n \log n_{i+1})$: the first part is due to the size of the edges E_B for all $B \in \Sigma_i$, and the second part is needed to save to which block in Σ_{i+1} a state belongs to.

Now we consider the time and space needed for refining the partition relation Γ_{i+1}. In line 11.8 Γ_{i+1} is initialised which takes time $\mathcal{O}(n_{i+1}^2 m_{i+1})$, and in line 11.9, the quotient automaton $\exists\mathcal{M}/_{\Sigma_{i+1}}$ is constructed which takes time $\mathcal{O}(m)$. The dominating part is the inside repeat-loop between lines 11.10–11.14. Similar to the above analysis for refining the blocks, in each iteration inside the repeat-loop, the time complexity is $\mathcal{O}(m_{i+1}^2 n_{i+1})$. At each iteration at least one pair from Γ_{i+1} is removed, which implies that the number of iterations is bounded by $|\Gamma_{i+1}| \leq n_{i+1}^2$. Thus, the complexity for this part is $\mathcal{O}(m_{i+1}^2 n_{i+1}^3)$. The space is in the order of $\mathcal{O}(n_{i+1}^2)$ for maintaining the partition relation Γ_{i+1}.

Since $m_i \leq m_\diamond \leq m_\sim$ and $n_i \leq n_\diamond \leq n_\sim$ for all i, the time complexity for iteration i is bounded by $\mathcal{O}(m + m_\diamond^2 n_\diamond^3 + m_\sim^2 n_\diamond)$. By Lemma 7.3.9, the number of iterations of SimQuo is in $\mathcal{O}(n_\diamond)$. Thus, the overall time complexity is $\mathcal{O}(mn_\diamond + m_\diamond^2 n_\diamond^4 + m_\sim^2 n_\diamond^2)$. The space complexity is $\mathcal{O}(n_\sim^2 + n \log n_\diamond)$. ■

The above time complexity is rather excessive. Similar to algorithms for deciding simulation preorder for PAs (cf. the algorithm ActSmf in Chapter 5), one can use the parametric maximum flow (PMF) idea to amortize computations which results in time-efficient algorithm. The penalty is that more memory is needed due to the need to store the networks across iterations.

Theorem 7.3.11. *Using PMF, the algorithm* SimQuo *has time complexity* $\mathcal{O}(mn_\diamond + m_\sim^2 n_\diamond^2)$*, and space complexity* $\mathcal{O}(m_\diamond^2 + n_\sim^2 + n \log n_\diamond)$.

Proof. We first analyse the time complexity. PMF is used inside repeat-loop (lines 11.10–11.14) to achieve better time complexity: instead of taking time $\mathcal{O}(m_{i+1}^2 n_{i+1}^3)$ in each iteration in the inside repeat-loop, the whole time spent in the inside loop can be achieved with complexity $\mathcal{O}(m_{i+1}^2 n_{i+1})$ using PMF (cf. Theorem 5.2.8). Since $m_\diamond \leq m_\sim$, combining the proof of Theorem 7.3.10, we get the time complexity $\mathcal{O}(mn_\diamond + m_\sim^2 n_\diamond^2)$.

Now we analyse the space complexity. PMF saves networks and flows on it such that it can be reused in the next iteration. The corresponding space complexity is $\mathcal{O}(m_{i+1}^2)$. Thus, the algorithm with PMF has space complexity $\mathcal{O}(m_\diamond^2 + n_\sim^2 + n \log n_\diamond)$. ■

7.4 Simulation Quotient for CPAs

In this section we consider CPAs. We first show that simulation preserves safety properties. Then, we discuss how our partition refinement based algorithm can be extended to CPAs.

7.4.1 Safety Properties

We discuss also which classes of properties are preserved by strong (probabilistic) simulation equivalence. We first recall briefly safety and liveness properties expressed by CSL [18], and the semantics of CSL-safety formulas over CPAs.

The CSL state formulas (Φ) and path formulas (ϕ) are defined by:

$$\Phi := a \mid \neg a \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathcal{P}_{\trianglelefteq p}(\phi)$$
$$\phi := \mathcal{X}^{\leq t}\, \Phi \mid \Phi\, \mathcal{U}^{\leq t}\, \Phi$$

If $\trianglelefteq \in \{<, \leq\}$, we have the set of safety CSL formulas. If $\trianglelefteq \in \{>, \geq\}$ instead, we have the set of liveness formulas. Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA. The semantics of the Boolean operators and the path formulas is the same as for CTMCs [10].

Let the set of (time-abstract) finite paths, denoted by *Path*, denote the set of finite sequence $\sigma = (s_0, \alpha_0, r_0), (s_1, \alpha_1, r_1), \ldots, s_n$ satisfying $s_i \xrightarrow{\alpha_i} r_i$ and that $r_i(s_{i+1}) > 0$ for $i = 1, \ldots, n-1$. For notational convenience, the path σ is also written by $s_0 \xrightarrow{\alpha_0, r_0} s_1 \xrightarrow{\alpha_1, r_1}, \ldots, s_n$. The length of the path σ is defined by $len(\sigma) = n$, and the last state s_n of the path is denoted by $last(\sigma)$.

A (time-abstract) scheduler is a function $A : Path \to Dist(Act \times Rate(S))$ such that $A(\sigma)(\alpha, r) > 0$ implies that $(last(\sigma), \alpha, r) \in R$. A scheduler A is *deterministic* if it is a function $Path \to Act \times Rate(S)$, i. e., schedulers that only assign probabilities 0 or 1 to actions. A deterministic scheduler A induces a CTMC $\mathcal{M}_A = (S_A, \mathbf{R}_A, L_A)$ where $S_A = Path$, $L_A(\sigma) = L(last(\sigma))$ and the transition rate matrix $\mathbf{R}_A$ is defined by: $\mathbf{R}_A(\sigma, \sigma \xrightarrow{\alpha, r} s) = r(s)$ if $A(\sigma) = (\alpha, r)$, and 0 otherwise. Let $Prob_s^{\mathcal{M}_A}$ denote the unique probability measure starting from state s under the scheduler A. If $\mathcal{M}$ is clear from

the context, we write $Prob_s^A$. The semantics of the probabilistic formula is defined by: $s \models \mathcal{P}_{\trianglelefteq p}(\phi)$ iff $\sup_A Prob_s^{\mathcal{M}_A}(\phi) \trianglelefteq p$ where $\trianglelefteq \in \{<, \leq\}$ and $Prob_s^{\mathcal{M}_A}(\phi)$ is a shorthand notation for the probability of set of timed and infinite paths satisfying ϕ [14]. The following lemma shows that simulation preserves safe fragments of CSL formulas[6]:

Lemma 7.4.1. *For CPAs, strong (probabilistic) simulation preserves all safe fragments of CSL formulas.*

Proof. Let $\mathcal{M}$ be a CPA and $s, s' \in S$. Assume that $s \precsim_{\mathcal{M}} s'$. We want to show that for safety CSL formula Φ it holds: $s' \models \Phi$ implies $s \models \Phi$. We prove this by structural induction on Φ. We consider only the probabilistic formula $\mathcal{P}_{\leq p}(\phi)$, as other formulas can be handled in an obvious way. By the semantics of the probabilistic formula, it then is sufficient to prove that for any scheduler A (that describes the behaviour starting from s), there is a scheduler A' such that:

$$Prob^A(s, \phi) \leq Prob^{A'}(s', \phi) \tag{7.6}$$

Without loss of generality, we may assume that A is a deterministic scheduler[7]. Obviously, A induces an (in general, infinite-state) CTMC. Since strong simulation on CTMCs preserves all safety CSL formulas [18], it remains to be shown that given A, there is a scheduler A' such that state s' in $\mathcal{M}_{A'}$ strongly simulates s in $\mathcal{M}_A$ in their direct sum. We observe that only states reachable from s (and s') in $\mathcal{M}_A$ (and $\mathcal{M}_{A'}$) are of interest.

Let $\mathcal{M}_A(n)$ denote the CTMC obtained by unfolding $\mathcal{M}$ respecting the scheduler A, starting from s, n steps. The states are paths of $\mathcal{M}$ with length at most n. Hence, only the first n-truncation of the scheduler A is used to construct $\mathcal{M}_A(n)$. We prove by induction on n that we can construct the n-truncation of the scheduler A' such that s' in $\mathcal{M}_{A'}(n)$ strongly simulates s in $\mathcal{M}_A(n)$ in their direct sum. This will imply the required inequality 7.6.

We consider the case $n = 0$. $\mathcal{M}_A(0)$ consists of only one state s, and $\mathcal{M}_{A'}(0)$ consists of only one state s'. Hence, the relation $R_0 = \{(s, s')\}$ is a strong simulation relation. Now assume that the statement is true for all $i \leq n$. Let R_n denote the corresponding strong simulation relation. We prove that it holds also for $n{+}1$. $\mathcal{M}_A(n{+}1)$ is constructed from $\mathcal{M}_A(n)$ by an additional unfolding. Consider an arbitrary state $\sigma' = s'_0 \xrightarrow{\alpha_0, r'_0} s'_1 \xrightarrow{\alpha_1, r'_1} \dots s'_n$ in $\mathcal{M}_{A'}(n)$ (where $s' = s'_0$). By construction of A', we

[6] We show the lemma with respect to time-abstract schedulers, as introduced in [14]. Our result can be easily proved, as some nice properties of time-abstract schedulers in [14] can be exploited. For timed reachability, a more general class of schedulers, namely timed schedulers, has been considered [120, 87]. Unfortunately, our result can not be extended in an obvious way for the timed schedulers. Recently, Neuhäußer and Katoen [87] have given semantics of CSL over CTMDP and they have considered the timed schedulers. They have shown that bisimulation preserves all CSL properties. An extension of the proof strategy of [87] might be possible.

[7] In [14], it is shown that deterministic schedulers suffice for maximum probabilities to reach a set of goal states within time t for CTMDPs. Since reachability probabilities do not reason about the transition labels, their result can be adapted in an obvious way to CPAs. Let $\mathcal{M} = (S, Act, \mathbf{R}, L)$ be a CPA, $s \in S$ and let Φ be a safe CSL formula. By structural induction on Φ, it is easy to verify that to check whether $s \models \Phi$, it is sufficient to consider only deterministic schedulers.

can always find a corresponding state $\sigma = s_0 \xrightarrow{\alpha_0,r_0} s_1 \xrightarrow{\alpha_1,r_1} \dots s_n$ in $\mathcal{M}_A(n)$ such that $s_i \precsim_{\mathcal{M}} s'_i$ and $\mu(r_i) \sqsubseteq_{\precsim_{\mathcal{M}}} \mu(r'_i)$ and $r_i(S) \leq r'_i(S)$ for all $i = 0, 1, \dots, n-1$, and $s = s_0$. Assume that $A(\sigma)(\alpha_n, r_n) = 1$. Since $s_n \precsim_{\mathcal{M}} s'_n$, there exists a transition $s'_n \xrightarrow{\alpha_n} r'_n$ such that $\mu(r_n) \sqsubseteq_{\precsim_{\mathcal{M}}} \mu(r'_n)$ and $r_n(S) \leq r'_n(S)$. We choose $A'(\sigma')(\alpha, r'_n) = 1$. We let $R_{n+1} = R_n \cup \{(\sigma, \sigma') \mid len(\sigma) = len(\sigma') = n+1 \wedge last(\sigma) \precsim_{\mathcal{M}} last(\sigma')\}$. By construction, R_{n+1} is a strong simulation, and thus, s' in $\mathcal{M}_{A'}(n)$ strongly simulates s in $\mathcal{M}_A(n)$ in their direct sum.

Since the combined transitions for CPAs induce exponential distributions, the proof can be adapted for strong probabilistic simulations by using $s \overset{\alpha}{\rightsquigarrow} r'$ instead of $s \xrightarrow{\alpha} r$. ■

For CTMDPs, strong simulation and strong probabilistic simulation also trivially coincide, as for MDPs. Another result is that the strong simulation quotient and the strong bisimulation for CTMDPs also coincide. A corresponding result for MDPs has been shown in [6, Theorem 3.4.15].

Lemma 7.4.2. *For CTMDP $\mathcal{M} = (S, Act, \mathbf{R}, L)$, it holds: $\sim = \precsim \cap \succsim$.*

Proof. Let $R := \precsim \cap \succsim$. The direction $\sim \subseteq R$ is trivial. We show that R is a strong bisimulation. Obviously, it is an equivalence relation. Let $s, s' \in S$ with $s \; R \; s'$ and $s \xrightarrow{\alpha} r$. Since $s \precsim s'$ holds, there exists $s' \xrightarrow{\alpha} r'$ with $\mu(r) \sqsubseteq_{\precsim} \mu(r')$ and $r(S) \leq r'(S)$. Similarly, as $s' \precsim s$ holds, there exists $s \xrightarrow{\alpha} r''$ with $\mu(r') \sqsubseteq_{\precsim} r(\mu'')$ and $r'(S) \leq r''(S)$. Since $\mathcal{M}$ is a CTMDP, we have that $r = r''$. Thus, we have that $r(S) = r'(S)$. Now applying [6, Lemma 5.3.5], we have that $\mu(r)(C) = \mu(r')(C)$ for all equivalence relation C of R. Hence, R is a strong bisimulation. ■

7.4.2 The Minimal Quotient Automaton for CPAs

Along the same line as for PAs, we briefly give the definition of minimal quotient automaton for CPAs.

For a CPA $\mathcal{M}$ and a rate function $r \in Dist(S)$ with $r(S) > 0$ and a partition Σ over S, we define $lift_\Sigma(r) \in Rate(\Sigma)$, the induced lifted rate function with respect to Σ, by: $lift_\Sigma(r)(B) = \sum_{s \in B} r(s)$ for $B \in \Sigma$. Similar as for PAs, for a set $B \subseteq S$, we write

- $B \xrightarrow{\alpha} r'$ if there exists $s \in B$ and $s \xrightarrow{\alpha} r$ with $r' = lift_\Sigma(r)$,
- $B \overset{\alpha}{\rightarrowtail} r'$ if for all $s \in B$ there exists $s \xrightarrow{\alpha} r$ with $r' = lift_\Sigma(r)$.

The notion of $\exists-$Quotient and $\forall$-quotient automata, little brothers for CPAs can be defined as Definition 7.1.3, Definition 7.1.4 and Definition 7.1.5, by replacing the distribution π by rate function r appropriately.

Lemma 7.4.3. *Let $\mathcal{M}$ be a CPA, and let Σ be the partition induced by $\simeq_{\mathcal{M}}$ of $\mathcal{M}$. Let $\mathcal{M}'$ be the CPA obtained by $\mathcal{M}$ by eliminating little brothers, $\forall$-quotient automaton $\forall\mathcal{M}/_\Sigma$ and $\exists$-quotient automaton $\exists\mathcal{M}/_\Sigma$. Then, all of these automata are pairwise simulation equivalent.*

The proof for the lemma is similar as the proofs for Lemmata 7.1.2 and 7.1.3 for PAs. Lemma 7.4.1 shows that simulation preserves safety CSL formulas. Analogously, for CPA $\mathcal{M}$, since the exists and for all quotient automata are simulation equivalent to $\mathcal{M}$, the quotient automaton thus preserves both safe and live fragments of CSL formulas.

7.4.3 Algorithm for CPAs

We discuss briefly how the partition refinement based algorithm can be extended to compute the simulation preorder for CPAs. Firstly, the notion of partition pair can be extended to CPAs, again by replacing the distributions by rate function appropriately. In the decision algorithm QuoSim we must take care of the rate condition. All we must change is to replace every occurrence of π by r, and the weight function condition of the form $\mu_1 \sqsubseteq_R \mu_2$ by $r_1 \sqsubseteq_R r_2$ accordingly. Recall $r_1 \sqsubseteq_R r_2$ is a shorthand notation for $\mu(r_1) \sqsubseteq_R \mu(r_2)$ and $r_1(S) \leq r_2(S)$ which could be checked via solving a maximum flow problem. Other notations are extended with respect to rate functions in an obvious way. It is routine to extend the correctness and complexity proofs for PAs to CPAs.

7.5 Experimental Results

In this section, we evaluate our new partition refinement based algorithm for deciding simulation preorder for PAs. Depending whether PMF is used in the algorithm, we have implemented both the space-efficient and time-efficient variants of the partition refinement based algorithm.

Dining Cryptographers. Consider the dining cryptographer models taken from the PRISM web-site. We take the most space efficient configuration (corresponds the configuration 0000 in Section 5.4) and refer to it as the *Original* algorithm in the sequel. Other configurations use more memory, and are at most faster by a factor of two, thus are not considered here. We compare it to our new partition refinement based algorithm: the configuration QuoPMF for the algorithm using PMF and the configuration Quotient for the algorithm without using PMF.

In Table 7.1 experiments are shown: in the upper part of both tables only one state label is considered, in the middle part uniform distribution of two different labels is considered, in the lower part we have uniform distribution of three different labels. For 6 cryptographers and one or two labels, the configuration Original runs out of memory; this is denoted by –. The number of the simulation equivalence classes is given in row #blocks, and the number of iterations of the refinement loops for the configurations Quotient and QuoPMF is given in row #refinement.

As expected, in the configuration Original the memory is indeed the bottleneck, while the partition refinement based algorithm uses significant less memory. More surprisingly is that partition refinement based algorithm often requires orders of magnitude less time, especially for small number of labels. The reason is that for this case study the simulation quotient automaton has much less states than the original automaton. Moreover, in the quotient automaton, most of the transitions fall into the same lifted distributions, thus making the maximum flow computation cheaper. Another observation is that the number of different labels affect the performance of all of the configurations, but in a different way. For the configuration Original more labels indicate that the initial relation is smaller thus always less time and memory are needed. For both Quotient and QuoPMF more labels give a finer initial partition, which means also a large quotient

Table 7.1: Time and memory used for Dining Cryptographers.

Cryptographers	3	4	5	6
States	381	2166	11851	63064
Transitions	780	5725	38778	246827
	Time (s)			
Original	0.52	20.36	987.40	–
Quotient	0.03	0.76	19.52	533.40
QuoPMF	0.03	0.73	18.93	528.00
#blocks	10	24	54	116
#refinement	3	3	3	3
Original	0.13	4.67	266.04	–
Quotient	0.05	0.93	18.53	394.80
QuoPMF	0.05	0.96	19.46	420.60
#blocks	63	247	955	3377
#refinement	4	4	4	4
Original	0.07	2.42	150.74	13649.30
Quotient	0.06	2.31	60.01	1185.16
QuoPMF	0.07	3.04	81.14	1536.78
#blocks	96	554	2597	8766
#refinement	3	4	4	5
	Memory (MB)			
Original	0.95	27.41	763.09	–
Quotient	0.02	0.11	0.71	4.35
QuoPMF	0.02	0.14	0.89	5.25
Original	0.21	4.68	104.46	–
Quotient	0.02	0.12	0.93	7.07
QuoPMF	0.02	0.21	2.42	26.02
Original	0.14	2.69	58.92	1414.57
Quotient	0.02	0.18	2.32	22.67
QuoPMF	0.03	0.41	10.75	124.53

automaton during the refinement loops. For this example the running time for one or two labels are almost the same, whereas with three labels more time is needed.

It is notable that QuoPMF does not perform well at all, even though it has better theoretical complexity in time. This observation is the same as the experimental results in Section 5.4: the corner cases (number of iterations in the inside repeat-loop is bounded by $n_\diamond^2$) which blow up the worst case complexity are rare in practice.

Self Stabilising Algorithm We now consider the self stabilising algorithm due to Israeli and Jalfon, also taken from the PRISM web-site. As the previous case study, in the upper, middle and lower part of the table we have one, two and three different uniformly distributed labels respectively. For 13 processes and one label, the configuration QuoPMF runs out of memory which is denoted by –. For this case study, we observe

Table 7.2: Time and memory used for the self stabilising algorithm.

Processes	10	11	12	13
States	1023	2047	4095	8191
Transitions	8960	19712	43008	93184
	Time (s)			
Original	11.35	53.66	259.18	1095.96
Quotient	20.25	138.60	470.84	2440.83
QuoPMF	28.17	177.54	655.09	–
#blocks	974	1987	4024	8107
#refinement	6	6	7	7
Original	1.73	8.68	37.63	199.31
Quotient	10.60	52.60	234.96	1248.30
QuoPMF	14.57	73.06	325.82	1704.87
#blocks	1019	2042	4090	8185
#refinement	5	6	6	7
Original	0.61	2.47	13.56	66.62
Quotient	10.36	39.02	260.09	900.99
QuoPMF	14.29	54.34	360.63	1235.27
#blocks	1015	2042	4085	8185
#refinement	6	5	8	6
	Memory (MB)			
Original	5.88	20.10	91.26	362.11
Quotient	0.36	1.24	4.50	17.04
QuoPMF	93.40	375.47	1747.35	–
Original	0.92	3.34	12.42	47.25
Quotient	0.38	1.29	4.63	17.35
QuoPMF	17.93	80.14	338.45	1379.45
Original	0.47	1.42	5.28	18.38
Quotient	0.38	1.29	4.62	17.35
QuoPMF	2.24	11.97	28.93	142.68

that the simulation quotient automaton has almost the same number of states as the original one. Thus, Original is the fastest configuration. Another observation is that the configuration Quotient needs almost the same amount of memory for three different number of labels. Recall that the space complexity of the configuration Quotient is $\mathcal{O}(n_\diamond^2 + n \log n_\diamond)$. Note the number of blocks differs only slightly for different number of labels, thus almost the same amount of memory is needed for this configuration.

Random Models. Most of the real models have a sparse structure: the number of successor distributions and the size of each distribution are small. Now we consider randomly generated PAs in which we can also observe how the algorithms behave for dense models. We consider random model with 200 states, in which there are two actions $|Act| = 2$, the size of each α-successor distribution in the model is uniform distributed

Table 7.3: Random models with various maximal distribution size D.

D	5	7	9	11	13	15	17
Transitions	1927	2717	3121	3818	4040	4711	5704
	Time (s)						
Original	0.50	1.10	1.80	3.19	3.76	6.04	10.26
Quotient	0.58	0.56	0.56	0.60	0.63	0.64	0.72
QuoPMF	0.54	0.54	0.52	0.59	0.60	0.60	0.70
#refinement	4	3	3	3	3	3	3
	Memory (kB)						
Original	138.23	137.58	108.18	132.85	115.10	131.88	145.19
Quotient	37.89	47.69	52.91	61.44	64.68	72.58	84.99
QuoPMF	263.77	179.51	128.60	144.11	107.94	83.46	110.10

Table 7.4: Random models with various maximal number of successor distributions MS.

MS	10	15	20	25	30	35
Transitions	3732	5283	7432	9250	11217	12659
	Time (s)					
Original	2.62	6.40	25.49	26.18	29.92	18.63
Quotient	1.15	2.97	6.82	4.88	4.44	2.83
QuoPMF	1.26	3.56	7.68	4.98	4.51	2.82
#blocks	200	200	200	13	22	9
#refinement	4	5	9	6	4	3
	Memory (kB)					
Original	348.79	437.73	501.16	567.91	575.46	628.32
Quotient	61.07	81.00	108.54	121.71	147.15	165.33
QuoPMF	1063.00	1663.16	2831.99	149.80	184.65	171.88

between $\{2, \ldots, D\}$, and the number of successor distributions for each state is uniform distributed between $\{1, \ldots, MS\}$. Only one state label is considered.

In Table 7.3 we set $MS = 5$ and consider various values of D. Because of the large distribution size, in all of these random models the simulation quotient automaton is the same as the corresponding original automaton, thus there is no reduction at all. Even in this extreme case, the partition refinement based methods reduce the memory by approximately 30%. Because of the large size of distributions, the corresponding maximum flow computations become more expensive for the configuration Original. In the partition refinement based approach the maximum flow computations are carried in the quotient automaton in each iteration, which saves considerable time. Thus the partition refinement based methods are faster, and scale much better than the configuration Original. Comparing with the configuration Quotient, the parametric maximum flow based method (configuration QuoPMF) uses more memory, and has only negligible time advantages.

In Table 7.4 we fix the maximal size of distribution to $D = 5$, and consider various values of MS. With the increase of MS, it is more probable that states are simulation equivalent, which means also that the number of blocks tends to be smaller for large MS.

Also for this kind of dense models, we observe that significant time and space advantages are achieved. Again, the PMF-based method does not perform better in time, and uses more memory.

7.6 Bibliographic Notes

In the non-probabilistic setting, a decision algorithm for simulation preorder has been proposed in [64] with complexity $\mathcal{O}(mn)$, where, as usual, n denotes the number of states and m denotes the number of transitions of labelled graphs. The space complexity is $\mathcal{O}(n^2)$ due to the need of saving the simulation relations. Since space could become the bottleneck in many applications [41], a space efficient algorithm has been introduced by Bustan and Grumberg [29]. With $n_\diamond$ denoting the number of simulation equivalence classes, the resulting space complexity is $\mathcal{O}(n_\diamond^2 + n \log n_\diamond)$, which can be considered optimal: the first part is needed to save the simulation preorder over the simulation equivalence classes, and the second part is needed to save to which simulation equivalence class a state belongs to. The corresponding time complexity obtained is rather excessive: $\mathcal{O}(n^2 n_\diamond^2 (n_\diamond^2 + m))$. Tan and Cleaveland [109] combined the techniques in [64] with the bisimulation minimisation algorithm [88], and achieved a better time complexity $\mathcal{O}(m \log n + m n_\sim)$, where $n_\sim$ denotes the number of bisimulation equivalence classes. The corresponding space complexity $\mathcal{O}(m + n_\sim^2)$.

Gentilini et. al. [54] incorporated the efficient algorithm of [64] into the partition refinement scheme and achieved a better time complexity $\mathcal{O}(m n_\diamond^2)$ while keeping the optimal space complexity $\mathcal{O}(n_\diamond^2 + n \log n_\diamond)$. This is achieved by characterising a simulation relation by a partition pair, which consists of a partition of the set of states and a relation over the partition. Then, the simulation problem can be reduced to a GCPP, which consists of determining the coarsest stable partition pair. The algorithm starts with the coarsest partition pair and refines both the partition and the relation over the partition according to stability conditions. In [94], an algorithm has been proposed with time complexity $\mathcal{O}(m n_\diamond)$ and space complexity $\mathcal{O}(n n_\diamond)$. Recently, van Glabbeek and Ploeger [111] have shown that the proofs in [54] were flawed, but have provided a fix for the main result. The algorithm we introduced in this chapter is based on the above partition refinement strategy. However, our space complexity is increased to $\mathcal{O}(n_\sim^2 + n \log n_\diamond)$. As discussed in Section 7.3, the predecessor based method for refining the partition does not work for PAs. For refining the partition, we have proposed a graph-based method, which has worst case space complexity $\mathcal{O}(n_\sim^2 + n \log n_\diamond)$.

7.7 Summary

In this chapter we proposed a partition refinement based space efficient algorithm for deciding simulation preorders. We discussed how to reduce the time complexity further by exploiting parametric maximum flow algorithm. Our implementation of the space-efficient and time-efficient variants of the algorithm has given experimental evidence, that compared to the original algorithm, not only the space-efficiency is improved drastically. Often the computation time is decreased by orders of magnitude.

Chapter 8

Conclusion and Future Works

8.1 Conclusion

In this thesis we have presented novel decision algorithms for various simulation relations for finite probabilistic systems. We first considered strong simulation. The straightforward algorithm starts with the largest relation, and refines it with respect to the conditions for strong simulations. Checking whether the conditions can be reduced to maximum flow problems. In the worst case, the algorithm terminates after n^2 iterations where n is the number of states, which is inefficient (with complexity $\mathcal{O}(n^7/\log n)$). We proposed drastic improvements by observing that the networks on which the maximum flows are calculated are very similar across iterations of the refinement loop. Adapting the parametric maximum flow algorithm to solve the maximum flows for the arising sequences of similar networks leads to an overall time complexity $\mathcal{O}(m^2 n)$ and space complexity $\mathcal{O}(m^2)$. For sparse models (state fanout is bounded by a constant k) our strong simulation algorithm has time and space complexity $\mathcal{O}(n^2)$. For weak simulation for Markov chains, the parametric maximum flow idea cannot be applied directly. Nevertheless, we manage to incorporate the parametric maximum flow algorithm into a decision algorithm with complexity $\mathcal{O}(m^2 n^3)$.

For probabilistic automata in discrete and continuous time, we considered also strong probabilistic simulation which is strictly coarser than strong simulation. We have shown that strong probabilistic simulation can be decided via solving LP problems, thus has polynomial complexity.

Since space complexity becomes the bottleneck of the algorithm in many applications, we discussed how to incorporate the partition refinement idea into the computation of the simulation preorder for PAs. The resulting algorithm has space complexity $\mathcal{O}(n_\sim^2 + n \log n_\diamond)$. However, we get a rather excessive time complexity. Experimental results show, however, that this method is very effective in time and memory: not only the space-efficiency is improved drastically, often orders of magnitude less time is required.

8.2 Future works

Compositional Aggregation. Systems are usually very large and complex as they are generated stepwise out of smaller building components via parallel compositions. For bisimulations, compositional aggregation based techniques have been successfully exploited for minimisation [26]: during generation of its state-space representation, composition and minimisation steps are intertwined along the structure of the compositional specification. As bisimulation, the simulation equivalence $\simeq$ is preserved by the parallel operator for PAs [98]. Thus, we can apply the compositional aggregation technique to our simulation setting: Instead of working on the final model, we could intertwine the computation of quotient automaton with respect to simulation and the parallel composition.

Acyclic Models. In [42], efficient algorithms have been developed for computing strong and weak bisimulations for acyclic interactive Markov chains [65]. For strong bisimulation the complexity is linear in the size of the transition relation, and for weak bisimulation, the complexity is linear in the size of the weak transition relation. The key element of the algorithm is to start from the bottom level of the acyclic model, and traverse the transitions backwards to refine the partitions on the higher level according to the bisimulation definition. This observation could also be used in deciding simulation quotient for acyclic models, which remains as our future work.

Infinite Systems. We have considered finite probabilistic systems in this thesis. With our algorithm, we are able to check whether the specification S simulates the implementation I provided both S and I are finite. While the specification is usually finite, however, the implementation is often infinite e.g. due to unbounded arithmetic variables or queues. Now we discuss how to check whether $I \precsim S$ holds, provided that the specification S is finite and the implementation I is infinite.

The goal is to find an intermediate finite model I', such that $I \precsim I' \precsim S$, then, by the transitivity of simulations, $I \precsim S$ also holds. For a finite partition of the state space of I, assume that the corresponding finite quotient automaton I' can be constructed with the property $I \precsim I'$ by the correctness of the construction[1]. Exploiting algorithm presented in this thesis, we can check whether $I' \precsim S$ holds. If it holds, we report $I \precsim S$ because of the transitivity of the simulation relation. The method is, however, inclusive if $I' \not\precsim S$: it could be either $I \not\precsim S$, or $I \precsim S$. For the former case, *counterexamples* shall be identified to witness the fact that $I \not\precsim S$. The latter case is possible since the abstraction I' might be chosen too coarser and there are more behaviour introduced which cannot be simulated by S. In this case we shall refine I', i.e., choose a finer partition and repeat the above steps.

[1] If I is represented by the guarded command language [67], such constructions are possible using predicate abstraction [57, 115].

Bibliography

[1] M. Abadi and L. Lamport. The existence of refinement mappings. In *Third Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 165–175, 1988.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[3] R. K. Ahuja, J. B. Orlin, C. Stein, and R. E. Tarjan. Improved algorithms for bipartite network flow. *SIAM J. Comput.*, 23(5):906–933, 1994.

[4] S. Andova and T. A. C. Willemse. Branching bisimulation for probabilistic systems: Characteristics and decidability. *Theor. Comput. Sci.*, 356(3):325–355, 2006.

[5] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In R. Alur and T. A. Henzinger, editors, *Computer aided verification (CAV)*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.

[6] C. Baier. On Algorithmic Verification Methods for Probabilistic Systems, 1998. Habilitations- schrift zur Erlangung der venia legendi der Fakultät für Mathematik and Informatik, Universität Mannheim.

[7] C. Baier, L. Cloth, B. R. Haverkort, M. Kuntz, and M. Siegle. Model checking action- and state-labelled Markov chains. In *Dependable Systems and Networks (DSN)*, pages 701–710. IEEE Computer Society, 2004.

[8] C. Baier, L. Cloth, B. R. Haverkort, M. Kuntz, and M. Siegle. Model checking Markov chains with actions and state labels. *IEEE Trans. Software Eng.*, 33(4):209–224, 2007.

[9] C. Baier, B. Engelen, and M. E. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *J. Comput. Syst. Sci.*, 60(1):187–231, 2000.

[10] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.

[11] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. In K. Jensen and A. Podelski, editors, *Tools and algorithms for the construction and analysis of systems (TACAS)*, volume 2988 of *LNCS*, pages 61–76. Springer, 2004.

[12] C. Baier and H. Hermanns. Weak bisimulation for fully probabilistic processes. In O. Grumberg, editor, *Computer Aided Verification (CAV)*, volume 1254 of *LNCS*, pages 119–130. Springer, 1997.

[13] C. Baier, H. Hermanns, and J.-P. Katoen. Probabilistic weak simulation is decidable in polynomial time. *Information processing letters*, 89(3):123–130, 2004.

[14] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theor. Comput. Sci.*, 345(1):2–26, 2005.

[15] C. Baier, H. Hermanns, J.-P. Katoen, and V. Wolf. Comparative branching-time semantics. In R. M. Amadio and D. Lugiez, editors, *CONCUR*, volume 2761 of *LNCS*, pages 482–497. Springer, 2003.

[16] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[17] C. Baier, J.-P. Katoen, H. Hermanns, and B. Haverkort. Simulation for continuous-time Markov chains. In L. Brim, P. Jančar, M. Křetínský, and A. Kučera, editors, *CONCUR*, volume 2421 of *LNCS*, pages 338–354. Springer, 2002.

[18] C. Baier, J.-P. Katoen, H. Hermanns, and V. Wolf. Comparative branching-time semantics for Markov chains. *Information and computation*, 200(2):149–214, 2005.

[19] C. Baier and M. Stoelinga. Norm functions for probabilistic bisimulations with delays. In J. Tiuryn, editor, *FoSSaCS*, volume 1784 of *LNCS*, pages 1–16. Springer, 2000.

[20] D. P. Bertsekas. *dynamic programming and optimal control*. Athena Scientific, 1995.

[21] S. Blom and S. Orzan. Distributed branching bisimulation reduction of state spaces. *Electr. Notes Theor. Comput. Sci.*, 89(1), 2003.

[22] B. Bloom and R. Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Sci. Comput. Program.*, 24(3):189–220, 1995.

[23] E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, R. Wimmer, and B. Becker. Compositional performability evaluation for statemate. In *Quantitative Evaluaiton of Systems (QEST)*, pages 167–178. IEEE Computer Society, 2006.

[24] J. Bogdoll, H. Hermanns, and L. Zhang. An experimental evaluation of probabilistic simulation. In *Formal Techniques for Networked and Distributed Systems (FORTE)*, volume 5048 of *LNCS*, pages 37–52. Springer, 2008.

[25] H. C. Bohnenkamp, P. van der Stok, H. Hermanns, and F. W. Vaandrager. Cost-optimization of the ipv4 zeroconf protocol. In *Dependable Systems and Networks (DSN)*, pages 531–540. IEEE Computer Society, 2003.

[26] H. Boudali, P. Crouzen, and M. Stoelinga. A compositional semantics for dynamic fault trees in terms of interactive markov chains. In K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, editors, *Automated Technology for Verification and Analysis (ATVA)*, volume 4762 of *LNCS*, pages 441–456, 2007.

[27] H. Boudali, P. Crouzen, and M. Stoelinga. Dynamic fault tree analysis using input/output interactive markov chains. In *Dependable Systems and Networks (DSN)*, pages 708–717. IEEE Computer Society, 2007.

[28] G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *CAV*, pages 274–287, 1999.

[29] D. Bustan and O. Grumberg. Simulation based minimization. In A. Voronkov, editor, *CADE*, volume 2392 of *LNCS*, pages 255–270. Springer, 2000.

[30] S. Cattani and R. Segala. Decision algorithms for probabilistic bisimulation. In L. Brim, P. Jancar, M. Kretínský, and A. Kucera, editors, *CONCUR*, volume 2421 of *LNCS*, pages 371–385. Springer, 2002.

[31] J. Cheriyan, T. Hagerup, and K. Mehlhorn. Can a maximum flow be computed in $o(nm)$ time? In M. S. Paterson, editor, *Automata, languages and programming*, volume 443 of *LNCS*, pages 235–248. Springer, 1990.

[32] J. Cheriyan and K. Mehlhorn. An analysis of the highest-level selection rule in the preflow-push max-flow. *Inf. Process. Lett.*, 69(5):239–242, 1999.

[33] L. Cheung, M. Stoelinga, and F. W. Vaandrager. A testing scenario for probabilistic processes. *J. ACM*, 54(6), 2007.

[34] I. Christoff. Testing equivalences and fully abstract models for probabilistic processes. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR*, volume 458 of *LNCS*, pages 126–140. Springer, 1990.

[35] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Logic of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.

[36] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[37] R. Cleaveland. On automatically explaining bisimulation inequivalence. In E. M. Clarke and R. P. Kurshan, editors, *Computer Aided Verification (CAV)*, volume 531 of *LNCS*, pages 364–372. Springer, 1990.

[38] R. Cleaveland, Z. Dayar, S. A. Smolka, and S. Yuen. Testing preorders for probabilistic processes. *Inf. Comput.*, 154(2):93–148, 1999.

[39] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM Trans. Program. Lang. Syst.*, 15(1):36–72, 1993.

[40] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *ACM Symposium on Theory of Computing*, 1987.

[41] C. Courcoubetis, M. Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. In E. M. Clarke and R. P. Kurshan, editors, *Computer Aided Verification (CAV)*, volume 531 of *LNCS*, pages 233–242. Springer, 1990.

[42] P. Crouzen, H. Hermanns, and L. Zhang. On the minimisation of acyclic models. In F. van Breugel and M. Chechik, editors, *CONCUR*, volume 5201 of *LNCS*, pages 295–309. Springer, 2008.

[43] S. Derisavi. A signature-based algorithm for optimal Markov chain lumping. In *Quantitative Evaluation of SysTems (QEST)*, pages 141–150. IEEE Computer Society, 2007.

[44] S. Derisavi. A symbolic algorithm for optimal Markov chain lumping. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *LNCS*, pages 139–154. Springer, 2007.

[45] S. Derisavi, H. Hermanns, and W. H. Sanders. Optimal State-Space Lumping in Markov Chains. *Inf. Process. Lett.*, 87(6):309–315, 2003.

[46] C. Derman. *Finite State Markovian Decision Processes*. Academic Press, 1970.

[47] J. Desharnais. *Labelled Markov processes*. PhD thesis, McGill University, 1999.

[48] J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, 179(2):163–193, 2002.

[49] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating labelled Markov processes. *Inf. Comput.*, 184(1):160–200, 2003.

[50] D. Dolev and A. C.-C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[51] A. Dovier, C. Piazza, and A. Policriti. A fast bisimulation algorithm. In *CAV*, pages 79–90, 2001.

[52] H. Fecher, M. Leucker, and V. Wolf. *Don't Know* in probabilistic systems. In A. Valmari, editor, *SPIN*, volume 3925 of *LNCS*, pages 71–88. Springer, 2006.

[53] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, 1989.

[54] R. Gentilini, C. Piazza, and A. Policriti. From bisimulation to simulation: Coarsest partition problems. *J. Autom. Reasoning*, 31(1):73–103, 2003.

[55] A. V. Goldberg. Recent developments in maximum flow algorithms (invited lecture). In S. Arnborg and L. Ivansson, editors, *6th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 1432 of *LNCS*, pages 1–10. Springer, 1998.

[56] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.

[57] S. Graf and H. Saidi. Construction of abstract state graphs with pvs. In O. Grumberg, editor, *Computer aided verification (CAV)*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

[58] J. Groote and F. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In *ICALP*, 1990.

[59] C. Groß, H. Hermanns, and R. Pulungan. Does clock precision influence zigbee's energy consumptions? In E. Tovar, P. Tsigas, and H. Fouchal, editors, *Principles of Distributed Systems (OPODIS)*, volume 4878 of *LNCS*, pages 174–188. Springer, 2007.

[60] D. Gusfield and É. Tardos. A faster parametric minimum-cut algorithm. *Algorithmica*, 11(3):278–290, 1994.

[61] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabitilies. In *IEEE Real-Time Systems Symposium*, pages 278–287, 1990.

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

[63] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.

[64] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 453–462. IEEE Computer Society, 1995.

[65] H. Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *LNCS*. Springer, 2002.

[66] H. Hermanns and S. Johr. Uniformity by construction in the analysis of nondeterministic stochastic systems. In *Dependable Systems and Networks (DSN)*, pages 718–728. IEEE Computer Society, 2007.

[67] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Tools and algorithms for the construction and analysis of systems (TACAS)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[68] M. Huth. An abstraction framework for mixed non-deterministic and probabilistic systems. In C. Baier, B. R. Haverkort, H. Hermanns, J.-P. Katoen, and M. Siegle, editors, *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 419–444. Springer, 2004.

[69] M. Huth. On finite-state approximants for probabilistic computation tree logic. *Theor. Comput. Sci.*, 346(1):113–134, 2005.

[70] D. T. Huynh and L. Tian. On some equivalence relations for probabilistic processes. *Fundam. Inform.*, 17(3):211–234, 1992.

[71] B. Jonsson. Simulations between specifications of distributed systems. In J. C. M. Baeten and J. F. Groote, editors, *CONCUR*, volume 527 of *LNCS*, pages 346–360. Springer, 1991.

[72] B. Jonsson and K. G. Larsen. Specification and refinement of probabilistic processes. In *Sixth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 266–277. IEEE Computer Society, 1991.

[73] P. C. Kanellakis and S. A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68, 1990.

[74] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.

[75] J.-P. Katoen, D. Klink, M. Leucker, and V. Wolf. Three-valued abstraction for continuous-time Markov chains. In W. Damm and H. Hermanns, editors, *Computer Aided Verification (CAV)*, volume 4590 of *LNCS*, pages 311–324. Springer, 2007.

[76] J.-P. Katoen, D. Klink, M. Leucker, and V. Wolf. Abstraction for stochastic systems by erlang's method of stages. In F. van Breugel and M. Chechik, editors, *CONCUR*, volume 5201 of *LNCS*. Springer, 2008.

[77] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. Van Nostrand, 1960.

[78] V. King, S. Rao, and R. E. Tarjan. A faster deterministic maximum flow algorithm. *J. Algorithms*, 17(3):447–474, 1994.

[79] R. Knast. Continuous-time probabilistic automata. *Information and Control*, 15(4):335–352, 1969.

[80] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[81] N. A. Lynch, R. Segala, and F. W. Vaandrager. Observing branching structure through probabilistic contexts. *SIAM J. Comput.*, 37(4):977–1013, 2007.

[82] S. Mangold, Z. Zhong, G. R. Hiertz, and B. Walke. IEEE 802.11e/802.11k wireless LAN: spectrum awareness for distributed resource sharing. *Wireless Communications and Mobile Computing*, 4(8):881–902, 2004.

[83] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: specification*. Springer, 1992.

[84] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. Modelling with generalized stochastic petri nets. *SIGMETRICS Performance Evaluation Review*, 26(2):2, 1998.

[85] R. Milner. An algebraic definition of simulation between programs. In *IJCAI*, pages 481–489, 1971.

[86] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[87] M. R. Neuhäußer and J.-P. Katoen. Bisimulation and logical preservation for continuous-time Markov decision processes. In L. Caires and V. T. Vasconcelos, editors, *CONCUR*, volume 4703 of *LNCS*, pages 412–427. Springer, 2007.

[88] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[89] D. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, pages 167–183, 1981.

[90] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *ACM Conference on Computer and Communications Security*, pages 245–254, 2000.

[91] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–200, 2001.

[92] A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In C. Palamidessi, editor, *CONCUR*, volume 1877 of *LNCS*, pages 334–349. Springer, 2000.

[93] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.

[94] F. Ranzato and F. Tapparo. A new efficient simulation equivalence algorithm. In *Symposium on Logic in Computer Science (LICS)*, pages 171–180. IEEE Computer Society, 2007.

[95] W. H. Sanders and J. F. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, 1991.

[96] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[97] R. Segala. A compositional trace-based semantics for probabilistic automata. In I. Lee and S. A. Smolka, editors, *CONCUR*, volume 962 of *LNCS*, pages 234–248. Springer, 1995.

[98] R. Segala. *Modeling and Verification of Randomized Distributed Realtime Systems*. PhD thesis, MIT, 1995.

[99] R. Segala. Testing probabilistic automata. In U. Montanari and V. Sassone, editors, *CONCUR*, volume 1119 of *LMCS*, pages 299–314. Springer, 1996.

[100] R. Segala. Probability and nondeterminism in operational models of concurrency. In C. Baier and H. Hermanns, editors, *CONCUR*, volume 4137 of *LNCS*, pages 64–78. Springer, 2006.

[101] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[102] R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. In B. Jonsson and J. Parrow, editors, *CONCUR*, volume 836 of *LNCS*, pages 481–496. Springer, 1994.

[103] R. Segala and A. Turrini. Comparative analysis of bisimulation relations on alternating and non-alternating probabilistic models. In *Quantitative Evaluaiton of Systems (QEST)*, pages 44–53. IEEE Computer Society, 2005.

[104] R. Segala and A. Turrini. Approximated computationally bounded simulation relations for probabilistic automata. In *CSF*, pages 140–156. IEEE Computer Society, 2007.

[105] K. Sen, M. Viswanathan, and G. Agha. Model-checking Markov chains in the presence of uncertainties. In H. Hermanns and J. Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3920 of *LNCS*, pages 394–410. Springer, 2006.

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

[107] M. Stoelinga. *Alea jacta est: Verification of Probabilistic, Real-Time and Parametric Systems*. PhD thesis, University of Nijmegen, 2002.

[108] D. Tabakov and M. Y. Vardi. Experimental evaluation of classical automata constructions. In G. Sutcliffe and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 3835 of *LNCS*, pages 396–411. Springer, 2005.

[109] L. Tan and R. Cleaveland. Simulation revisited. In T. Margaria and W. Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2031 of *LNCS*, pages 480–495. Springer, 2001.

[110] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[111] R. J. van Glabbeek and B. Ploeger. Correcting a space-efficient simulation algorithm. In A. Gupta and S. Malik, editors, *Computer Aided Verification (CAV)*, volume 5123 of *LNCS*, pages 517–529. Springer, 2008.

[112] R. J. van Glabbeek, S. A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Inf. Comput.*, 121(1):59–80, 1995.

[113] R. J. van Glabbeek, S. A. Smolka, B. Steffen, and C. M. N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *IEEE Symposium on Logic in Computer Science*, pages 130–141, 1990.

[114] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 327–338. IEEE, 1985.

[115] B. Wachter, L. Zhang, and H. Hermanns. Probabilistic model checking modulo theories. In *Quantitative Evaluation of SysTems (QEST)*, pages 129–138, 2007.

[116] R. Wimmer, M. Herbstritt, H. Hermanns, K. Strampp, and B. Becker. Sigref-a symbolic bisimulation tool box. In S. Graf and W. Zhang, editors, *Automated Technology for Verification and Analysis (ATVA)*, volume 4218 of *LNCS*, pages 477–492. Springer, 2006.

[117] R. Wimmer, H. Hermanns, and S. Derisavi. Symbolic partition refinement with dynamic balancing of time and space. In *Quantitative Evaluation of SysTems (QEST)*. IEEE Computer Society, 2008.

[118] V. Wolf, C. Baier, and M. E. Majster-Cederbaum. Trace machines for observing continuous-time markov chains. *Electr. Notes Theor. Comput. Sci.*, 153(2):259–277, 2006.

[119] V. Wolf, C. Baier, and M. E. Majster-Cederbaum. Trace semantics for stochastic systems with nondeterminism. *Electr. Notes Theor. Comput. Sci.*, 164(3):187–204, 2006.

[120] N. Wolovick and S. Johr. A characterization of meaningful schedulers for continuous-time Markov decision processes. In E. Asarin and P. Bouyer, editors, *Formal modeling and analysis of times systems (FORMATS)*, volume 4202 of *LNCS*, pages 352–367. Springer, 2006.

[121] L. Zhang. A space-efficient probabilistic simulation algorithm. In F. van Breugel and M. Chechik, editors, *CONCUR*, volume 5201 of *LNCS*, pages 248–263. Springer, 2008.

[122] L. Zhang and H. Hermanns. Deciding simulations on probabilistic automata. In K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, editors, *Automated Technology for Verification and Analysis (ATVA)*, volume 4762 of *LNCS*, pages 207–222. Springer, 2007.

[123] L. Zhang, H. Hermanns, F. Eisenbrand, and D. N. Jansen. Flow faster: Efficient decision algorithms for probabilistic simulations. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *LNCS*, pages 155–169. Springer, 2007.

[124] L. Zhang, H. Hermanns, F. Eisenbrand, and D. N. Jansen. Flow faster: Efficient decision algorithms for probabilistic simulations. *Special Issue on TACAS 2007, Logical Method in Computer Science (LMCS)*, 2008.