Recursive Probabilistic Models: efficient analysis and implementation

Dominik Wojtczak

Doctor of Philosophy
Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh
2008

(Graduation date: 2009)

Abstract

This thesis examines Recursive Markov Chains (RMCs), their natural extensions and connection to other models. RMCs can model in a natural way probabilistic procedural programs and other systems that involve recursion and probability. An RMC is a set of ordinary finite state Markov Chains that are allowed to call each other recursively and it describes a potentially infinite, but countable, state ordinary Markov Chain. RMCs generalize in a precise sense several well studied probabilistic models in other domains such as natural language processing (Stochastic Context-Free Grammars), population dynamics (Multi-Type Branching Processes) and in queueing theory (Quasi-Birth-Death processes (QBDs)). In addition, RMCs can be extended to a controlled version called Recursive Markov Decision Processes (RMDPs) and also a game version referred to as Recursive (Simple) Stochastic Games (RSSGs). For analyzing RMCs, RMDPs, RSSGs we devised highly optimized numerical algorithms and implemented them in a tool called PReMo (Probabilistic Recursive Models analyzer). PReMo allows computation of the termination probability and expected termination time of RMCs and QBDs, and a restricted subset of RMDPs and RSSGs. The input models are described by the user in specifically designed simple input languages. Furthermore, in order to analyze the worst and best expected running time of probabilistic recursive programs we study models of RMDPs and RSSGs with positive rewards assigned to each of their transitions and provide new complexity upper and lower bounds of their analysis. We also establish some new connections between our models and models studied in queueing theory. Specifically, we show that (discrete time) QBDs can be described as a special subclass of RMCs and Tree-like QBDs, which are a generalization of QBDs, are equivalent to RMCs in a precise sense. We also prove that for a given QBD we can compute (in the unit cost RAM model) an approximation of its termination probabilities within i bits of precision in time polynomial in the size of the QBD and linear in i. Specifically, we show that we can do this using a decomposed Newton's method.

Acknowledgements

First of all I would like to thank my supervisor, Kousha Etessami, for his enthusiasm and encouragement to work on hard problems, patience in correcting my (silly) grammar errors, tough lessons in polished writing, and simply for being a good friend. He proved to be an invaluable source of knowledge and I have learned a great deal from him. This thesis would not be what it is without his careful reading and comments on how to improve it.

I would like to thank Mihalis Yannakakis for fruitful collaboration, Amit Dubey and Frank Keller for providing us SCFGs from their NLP work which were later used as a case study for PReMo, Mary Cryan and my second supervisor Stephen Gilmore for being members of my progress panels, and finally Richard Mayr and Peter Bro Miltersen for agreeing to be my examiners. I would also like to thank Stephen and Michael Ummels for careful reading and commenting on parts of this thesis.

Furthermore, I simply thank all the people that I met in Edinburgh that made these three years in Edinburgh so enjoyable, especially the ones that I had the pleasure to share a flat or office with and who later became my good friends. Special thanks to Christophe, Jorge, Gaya for a constant dose of entertainment and Merilin for her great sense of humor, helping me to stay in good shape and just being there for me.

Last, but not least, I would like to thank my parents for their constant unconditional love and support.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Most of the material of this thesis was already published in [WE07], [EWY08i] and [EWY08q].

(*Dominik Wojtczak*)

Contents

List of Abbreviations

1-RSSG	1-exit Recursive Simple Stochastic Game, page 19.
1C-MDP	One-Counter Markov Decision Process, page 73.
DAG	Directed Acyclic Graph, page 13.
ETℝ	Existential Theory of the Reals, page 24.
LFP	Least Fixed Point, page 13.
MT-BP	Multi-Type Branching Process, page 28.
p1CA	probabilistic 1-Counter Automaton, page 30.
PLS	Polynomial Local Search, defined in [JPY88].
PPAD	"polynomial parity argument in a directed graph", defined in [Pap94].
pPBA	probabilistic Basic Process Algebra, page 29.
pPDS	probabilistic Pushdown System, page 28.
QBD	Quasi-Birth-Death process, page 31.
QBD-MDP	Quasi-Birth-Death Markov Decision Process, page 73.
RMC	Recursive Markov Chain, page 18.
RMDP	Recursive Markov Decision Process, page 18.
RSSG	Recursive Simple Stochastic Game, page 16.
SCC	Strongly Connected Component, page 13.
SCFG	Stochastic Context-free Grammar, page 26.
SM	stackless & memoryless strategy, page 21.

SQRT-SUM square-root sum problem, page 23.

TL-QBD Tree-Like Quasi-Birth-Death process, page 32.

TS-QBD Tree-structured Quasi-Birth-Death process, page 32.

List of Tables

List of Figures

Chapter 1

Introduction

The main goal of this thesis is the algorithmic analysis of certain classes of finitely-presentable infinite-state probabilistic models which combine probabilistic, recursive and in many cases also controlled/game behavior. We devise algorithms for analysis problems for several classes of such systems, study their complexity, and we describe a tool, called PReMo, which provides highly optimized implementations of analysis algorithms for these models. Such systems can be used as models of discrete-time stochastic reactive systems that naturally occur in various domains. Models are useful, since they help us to abstract away from reality and focus on things that really matter. Designing a model helps us understand the system before we actually build it, while deriving a model from a real system helps us to understand it. Systems that exhibit both probabilistic and controlled behavior are common in many domains. Unreliability is inherent to almost all physical systems, e.g., telecommunication networks, distributed systems, railway connections etc. Also, software systems, designed with robustness and efficiency in mind, often make explicit use of randomness, thus exhibiting probabilistic behavior. Even if a system or a program behaves deterministically, the environment were it operates is in many cases either unknown or very complex, and the best we can do is to model it either as a probabilistic interaction with our system (in order to examine the average behavior of our model) or as a malicious interaction (to examine its worst possible behavior).

It is very common to model purely probabilistic systems as finite-state Markov Chains or as Markov Decision Processes (MDPs) if the system is both probabilistic and controlled. However, finite-state probabilistic models differ significantly from infinite-state ones. (For instance, in finite-state systems no *null recurrent* states can occur, i.e., states that we revisit with probability 1 but the expected time of that happening being

equal to ∞.) Many (finitely-presentable) infinite-state models have been studied in various domains of probability theory and computer science. These domains include population dynamics and nuclear chain reactions (Multi-Type Branching Processes (MT-BPs)), natural language processing (Stochastic Context-Free Grammars (SCFGs)), biological sequence analysis (again SCFGs), queueing theory (Quasi-Birth-Death processes (QBDs)), modeling web surfing (random walks with "back buttons") and probabilistic model checking (Recursive Markov Chains (RMCs) and probabilistic Pushdown Systems (pPDSs)).

As an example, let us mention the classic model of QBDs that are widely used in performance evaluation to reason about efficiency of queueing systems. QBDs can model a single queue of clients at a server. Such a system can be in one of finitely-many control states. A client is served at a rate that depends on the current control state of the system. Similarly, a new client arrives at a rate that depends on the current control state of the system. During any such event the current control state can change. Although such a model assumes continuous time with exponential distributions on the time it takes for actions to happen, by just studying the embedded discrete-time QBD we can deduce many important properties of the original continuous-time model that do not depend on precise timing information. To reason about such models, we derive certain quantities or establish some properties that have some practical implications, e.g., for QBDs that could be whether almost surely the queue becomes empty and what is the expected maximum length of a queue before that happens. In order to compute such properties for a QBD one usually computes first its fundamental matrix, the so-called G matrix, consisting of the probabilities of reaching some internal state with an empty queue when starting at some internal state with a single client in the queue.

Most of the models studied in this thesis have an analog of the G matrix that is usually computed first in order to derive other properties of the system. These analogs usually correspond to the least fixed point (LFP) of a specific monotone polynomial equation system if the model is purely probabilistic or a min-max-polynomial equation system if it is both probabilistic and controlled. In order to find the LFP of such systems of equations efficiently we can use, e.g., numerical approximation methods described in [EY05s]. Although some tools are available for analyzing QBDs, no tool was able to analyze the more general model of RMCs and their controlled/game extensions: Recursive Markov Decision Processes (RMDPs) and Recursive Simple Stochastic Games (RSSGs). We filled this gap by developing PReMo, a tool that imple-

ments highly optimized versions of these approximation algorithms in Java. PReMo accepts as an input: RMCs, RSSGs, and probabilistic 1-Counter Automata (p1CAs) (equivalent of QBDs) specified in simple input languages tailored to each model and is able to perform all kind of analyses for them: computing the termination probability, the expected termination time, the standard deviation of that time, and the steady-state distribution for QBDs. Various other models like pPDSs, MTBPs and TL-QBDs can be analyzed indirectly since they can be translated by hand to the models supported by PReMo. PReMo is the first (and still only) software tool that is able to analyze all these models and its version for the most common operating systems can be downloaded from: groups.inf.ed.ac.uk/premo. In fact PReMo can also analyze much more general monotone systems of equations, not just polynomial ones.

In order to enable PReMo to analyze the best/worst expected time of termination for abstract models of probabilistic procedure programs we study an MDP extension of RMCs with rewards, Recursive Markov Decision Processes (RMDPs) (and more generally their game version with two players, Recursive Simple Stochastic Games (RSSGs)). We can imagine the controller that is trying to minimize the expected termination time in an RMDP as someone that is trying to fine-tune the program to make it run as fast as possible. At the same time a controller that is trying to maximize that time could be a malicious environment in which the program runs and if our system performs well against such a controller, it will be fine when run in any other environment. An uncontrolled model of pPDSs with rewards was already studied in [EKM05]. However, the questions about the general controlled/game model become undecidable and it is necessary to restrict ourselves to a natural 1-exit subclass of RMDPs (that can be equivalently reformulated as a game version of SCFGs). A crucial additional requirement is strict positivity of all rewards on all transitions, but this assumption is very natural for modeling the expected running time of probabilistic procedural programs since each step of a program takes some nonzero amount of time. We show that the one-player controlled versions (i.e., both minimizing and maximizing) of such a model can be solved in PTIME using Linear Programming.

Furthermore, we study the performance of a variant of the classical Newton's method, called the decomposed Newton's method, when used to approximate the G matrix of a given QBD within a given number of bits of precision m (i.e., within additive error $1/2^m$). Although many numerical methods were developed and their performance studied for computing the G matrix of a QBD, their analysis always followed the classical "linear/quadratic convergence" criterion of numerical analysis.

However, that kind of analysis does not shed light on the complexity of computing the G matrix, in the size of the model, since it studies the number of the iterations required only as a function of the number of bits of precision m, totally ignoring the size of the input QBD model. In fact, in practice almost always the desired m is just a small constant while it is the size of the input QBD that really varies. Furthermore, all these analyses assumed the input QBD to have several technical assumptions, including strong connectivity assumption, in order to make their proof techniques work. We fill these gaps at least for the decomposed Newton's method and show a strong polynomial bound on the number of iterations both in the size of the input as well as the number of bits of precision required to compute the G matrix of a general, possibly non-strongly connected, QBD.

The relationships between some of these models are already well established, but some of the connections were unknown until recently. Establishing how their expressive power compares helps those who build models to choose the right balance between the expressive power of a model and the efficiency of its analysis. In this thesis we show a new tight correspondence between several models used in probabilistic model checking and queueing theory. This leads to a number of methods and algorithms developed in queueing theory to be applied to models like pPDSs and RMCs, and complexity results established for the latter to be applied to the former. We also establish new results for QBDs that do not follow from any previous work.

The general domain of finitely-presented infinite-state probabilistic models is a very rich and fascinating field of study that is gaining more and more attention recently. This thesis addresses the computational complexity of some of the important questions about them as well as their practical aspect by developing a tool that is able to analyze them efficiently. However, many theoretical questions still remain open and we point out some of the them in the conclusion section of each chapter.

1.1 Outline of the thesis

The main goal of this thesis is to study the decidability and complexity of probabilistic infinite-state models described by various finitely-presentable specification formalisms. We survey some known results about these systems and provide a common framework of (infinite-state) Simple Stochastic Games as the underlying semantics for all of them.

In Chapter 2 we describe some preliminaries necessary to understand the rest

of the thesis and define what it means for two finitely-presentable models to be *essentially equivalent* and prove some equivalences according to that definition. The first part of that chapter contains the definition of (infinite-state) Simple Stochastic Games, and formally defines several finitely-presentable formalisms that define subclasses of Simple Stochastic Games and surveys some of the results already known for such subclasses. In the second part, we formally define what it means for two (finite-presentable) models to be essentially equivalent, and then prove equivalences between SCFGs games and 1-exit RSSGs (a natural subclass of RSSGs defined in Section 2.2.1 in Chapter 2; informally, it is a collection of RSSGs whose each component has just one exit); p1CAs and QBDs; pPDSs, RMCs, TL-QBDs and TS-QBDs. The second part of that chapter is partly based on our paper [EWY08q] and is a joint work with Kousha Etessami and Mihalis Yannakakis.

In Chapter 3 we examine the computational complexity of the fundamental questions for QBDs and related models. We start with stating the consequences of the equivalence between QBDs and p1CAs, and between Tree-structured (Tree-like) QBDs and pPDSs, and show that the SQRT-SUM problem reduces to a basic quantitative decision question for QBDs. We show important structural properties of p1CAs, and use them to prove a polynomial bound on the number of iterations of the decomposed Newton method needed to approximate the G matrix of a QBD within a given additive error ε. In the end of that chapter, we define a single controller MDP version of QBDs, Quasi-Birth-Death Markov Decision Processes (QBD-MDPs), and show that even the qualitative decision problems for them are DP-hard (both NP-hard and coNP-hard). In order to understand that chapter it is necessary to read the definitions of QBDs and p1CAs (if needed also TL-QBDs, TS-QBDs, and pPDSs), all contained in Section 2.2 in Chapter 2. Chapter 3 it is based on the extended version of our paper [EWY08q] and is a joint work with Kousha Etessami and Mihalis Yannakakis.

In Chapter 4 we study 1-exit Recursive Simple Stochastic Games (1-RSSGs) with positive rewards as a natural model of probabilistic procedural programs. We start by proving that 1-RSSGs with positive rewards are determined and their value is the least nonnegative solution of a min-max-linear equation system. Moreover, we show that both players have always stackless&memoryless optimal strategies which can be found via a simultaneous strategy improvement algorithm. Furthermore, we show that 1-RMDPs with positive rewards can be solved in PTIME, from which it easily follows that quantitative decision problems about 1-RSSGs with positive rewards are contained in NP$\cap$coNP and finding the optimal strategies is in Polynomial Local

Search (PLS; see [JPY88]). Finally, we show that qualitative questions about 1-RSSGs with positive rewards are as hard as quantitative questions about Condon's stochastic games, whilst qualitative questions about RSSGs with positive rewards are undecidable. In order to understand that chapter it is necessary to read the definition of RSSGs which is contained in Section 2.2 in Chapter 2. Chapter 4 is an extended version of our paper [EWY08i] and is a joint work with Kousha Etessami and Mihalis Yannakakis.

In Chapter 5 we describe our tool PReMo that we developed in Java. We discuss its internal workings, what kind of numerical algorithms are implemented and how. We present the experimental performance of PReMo on various case studies, including grammars derived from the Penn Treebank corpora, and finally we compare PReMo with SMCSolver, a solver for QBDs. That chapter, apart from reading some of the formal definitions from Chapter 2 if necessary, is self-contained. PReMo was already presented in a short tool paper [WE07] and some of the experimental results were already contained there as well.

1.2 Related work

In this section we point out a selection of the vast literature that is related to the results in this thesis. Each chapter of this thesis starts with an introduction that summarizes the tightly related work to the results contained in that chapter. Formal definitions, notions and results about the models discussed in this thesis are described in Chapter 2 and it might be beneficial to read them first before reading this section.

1.2.1 Stochastic games

(Concurrent discounted) Stochastic Games were studied by Lloyd Shapley in the early 1950s ([Sha53]). (Finite-state turn-based) Simple Stochastic Games (SSGs) were studied by Anne Condon ([Con92]), and she showed that their quantitative termination decision problem is in NP $\cap$ coNP; it is a well-known open problem whether it is in PTIME. In [CM94] strategy improvement algorithms for SSGs were studied and various single-node strategy improvements were shown to require at least an exponential number of strategy switches in the worst case. Also, based on variants of the classic Hoffman-Karp algorithm [HK66], a simultaneous strategy improvement was studied there for SSGs; an analog of our simultaneous strategy improvement for 1-RSSGs with positive rewards presented in Chapter 4. It remains open whether these simultaneous strategy improvement algorithms run in PTIME or not. Competitive Markov Decision

Processes ([FV97]) is yet another name for finite-state (concurrent) stochastic games used in Operation Research (OR) and economics.

See the introduction to Chapter 2 for more details.

1.2.2 RMCs and pPDSs

The two equivalent purely probabilistic recursive models, Recursive Markov Chains (RMCs) and probabilistic Pushdown Systems (pPDSs) were introduced independently in [EY05s] and [EKM04], as a natural abstract model of probabilistic procedural programs. Since that time, several papers appeared that study not only the complexity of their fundamental questions (e.g., [EY05s, EY07]), but also model checking (e.g., [EKM04, BKS05, EY05t, YE05]), and quantitative properties of their extended version with rewards on transitions (e.g., [EKM05]). See also Brázdil's thesis ([Bra07]) for an overview of most of these results.

1.2.3 RSSGs

RMDPs and RSSGs were studied in [EY05i, EY06s] in a setting without rewards, where the goal of the players was to maximize/minimize the probability of termination. 1-RMDPs were also studied in [BBFK06], where the goal of the single player was to maximize/minimize the probability of reaching a given control state regardless of its context. As we will see in Chapter 2, all these models can be efficiently embedded in a nonnegative reward model where the aim of the players is to maximize/minimize the total reward throughout the game. The RSSGs studied by us in Chapter 4 are reward-based, but crucially with all rewards being positive. See the "state of the art" part in Section 2.2.1 in Chapter 2 for detailed related work and known results about all these models.

1.2.4 QBDs and TL-QBDs

Quasi-Birth-Death Processes (QBDs) and more generally M/G/1-type and G/M/1-type Markov chains [1], have been studied for decades in queueing theory, performance evaluation, and related areas, both in discrete and continuous time, and so have numerical solution methods for them (see, e.g., the books [Neu81, Neu89, LR99, BLM05]).

[1] These chains also have the underlying transition structure of a 1-counter automaton, but one which can increase, or decrease, respectively, the counter by more than 1 in a single transition. These models need not in general be finitely-presented, because they do not a priori bound how many transitions (with distinct counter value changes) can exist from a given state. But of course typical instances are finitely-presented.

Latouche in [Lat94], studied the behavior of Newton's method on strongly connected QBDs, and showed (building on [OR70]) that under certain extra assumptions that method converges monotonically and "quadratically" to the matrix G. Remke et. al. in [RHC07] have studied numerical algorithms for model checking of continuous-time QBDs against properties expressed in the Continuous(-time) Stochastic Logic. Several other models, in particular, (discrete-time) stochastic Petri Nets restricted to markings where just one place can be unbounded, are already known to be equivalent to QBDs (see, e.g, [Ost01]).

Tree-Structured QBDs (TS-QBDs) are a generalization of QBDs, first studied in [YS94, TSY95, YA99]. Tree-Like QBDs (TL-QBDs) are a restriction of TS-QBDs, studied in, e.g., [LR99, BLM03, VHB06]. It was already observed in [HB03] that TL-QBDs and TS-QBDs are equivalent, under a tight notion of equivalence that we define in Section 2.3 in Chapter 2. Bini et. al. [BLM03] studied the performance of several numerical algorithms for TL-QBDs, including Newton's method. Building on [Lat94], they showed that under a similar set of assumptions, Newton's iterations are defined and converge monotonically and "quadratically" for various quantities such as the termination probabilities (the analog of the G matrix).

See the introduction to Chapter 3 for more details.

1.2.5 p1CAs

Probabilistic 1-Counter Automata have not yet been extensively studied in the literature on model checking and verification. However in a recent, and as yet unpublished, work ([BBE+09]) with Brázdil, Brožek, Etessami and Kučera we obtained a PTIME algorithm for deciding whether termination occurs almost surely (with probability 1) starting from a given control state, and counter value 1, for a probabilistic 1-counter automaton. This is related to the known conditions for the transience, positive recurrence, or null-recurrence of a QBD, under certain assumptions (e.g., irreducibility of the A matrix). However, there does not appear to be an analogous unconditional result about termination in the QBD literature. A result in [EY05s] on RMCs implies that the same problem for TL-QBDs is SQRT-SUM-hard (the definition of the SQRT-SUM problem can be found in the "state of the art" part of Section 2.2.1 in Chapter 2).

1.2.6 Non-probabilistic models

Pushdown automata are of course classic models that date back to the origins of automata theory (see, e.g., [HU79]). They have many applications, e.g., in parsing of languages. Pushdown systems (the transition graphs of pushdown automata), and the equivalent model of Recursive State Machines, have been studied extensively in the past decade for the analysis and model checking of procedural programs (see, e.g., [EHRS00, ABE$^+$05]). Boolean Programs ([BR00]) is yet another model to represent such programs and it has been successfully applied in a tool called SLAM ([BR02]), which is being used to verify Windows's device drivers. See the conclusions Section 5.5 to Chapter 5 for more details.

1-Counter Automata, which amount to Pushdown Systems with only one stack symbol, are a standard automata-theoretic model, and their relationship to other infinite-state models in automata theory has been well studied (see, e.g., [VP75, KJ02, Kuc03]).

1.2.7 Newton's method and systems of monotone polynomial equations

In [EY05s], a decomposed Newton's method was studied for approximation of termination probabilities, and it was shown that, after decomposition, Newton's method converges monotonically, starting from 0, for arbitrary monotone polynomial systems with nonnegative constants. These results built on the classic text [OR70].

Subsequently, Esparza, Kiefer, and Luttenberger, [KLE07, EKL08] studied in much greater detail the performance of (decomposed) Newton's method on such monotone systems of polynomial equations. Importantly for the results established in Chapter 3, not only was a worst-case linear convergence established as a function of the desired error, but in the case of strongly connected system of equations a constructive exponential upper bound was provided on the number of iterations required for Newton's method as a function of the encoding size of the polynomial system. No such constructive upper bounds is known so far for general (not-strongly connected) equation systems. It has to be noted that this kind of analysis differs significantly from the classical convergence rate analysis traditional to numerical analysis.

See the introduction to Chapter 3 for further details.

1.2.8 Other

An independent work by Gawlitza and Seidl [GS07] considers monotone min-max-linear equations with potentially negative constant terms (with entirely different mo-

tivation from abstract interpretation), and studies a different kind of strategy improvement algorithm for computing their least fixed point solution over the *full* extended reals. Their work is related to our work in Chapter 4, but in rather subtle ways. In particular their notion of Least Fixed Point(LFP) over the extended reals may yield negative values or even $-\infty$, and they assume that "strategies" (choices for the max and min operators) are memoryless, rather than proving a (memoryless) determinacy result. Moreover, their strategy improvement algorithm requires a particular initial strategy (otherwise it can fail) and thus is not directly formulable as a local search. Unlike our results, their results apparently do not yield [Gaw08] containment in NP$\cap$coNP, nor in PLS, for the relevant decision and search problems. Nevertheless, there are connections between their work and ours. In particular, Gawlitza [Gaw08] informs us that a modified version of their strategy improvement algorithm can also be used to obtain our PTIME upper bound for the LFP, over the non-negative extended reals, for the linear-min and linear-max equations that arise for 1-RMDPs.

Models related to 1-RMDPs have been studied in Operational Research (OR), under the name Branching Markov Decision Chains (a controlled version of multi-type Branching processes). These are close to the single-player SCFG model, with non-negative rewards, but simultaneous derivation law. They were studied by Pliska [Pli76], in a related form by Veinott [Vei69], and extensively by Rothblum and co-authors (e.g., [RW82]). Besides the restriction to simultaneous derivation, these models were restricted to the single-player MDP case, and to simplify their analysis they were typically assumed to be "transient" (i.e., the expected number of visits to a node was assumed to be finite under all strategies). None of these works yield a PTIME algorithm, that we present in Section 4.4 in Chapter 4, for computing the optimal expected rewards for 1-RMDPs with positive rewards. Furthermore, our results apply more generally to a 2-player setting.

Chapter 2

Probabilistic infinite state models and equivalences

In this chapter we define several formalisms that allow for specifying probabilistic countable-state models (with rewards). These models have been studied in many domains of theoretical computer science and mathematics such as population dynamics (Multi-Type Branching Processes [KS47, Har63]), natural language processing and biological sequence analysis (Stochastic Context-Free Grammars [DEKM99, MS99]), queueing theory (Quasi-Death-Birth processes [Neu81, Neu89, LR99] and Tree-like QBDs [LR99, BLM03, VHB06]), modeling web surfing ("random walks with back-buttons" [FKK$^+$00]) and probabilistic model checking (Recursive Markov Chains [EY05s] and probabilistic Pushdown Systems [EKM04]). Relationships between some of these models is already well established (e.g., see [EY05s]). In this chapter we establish some new connections between models used in queueing theory and models used in probabilistic procedural programs analysis. Although QBDs have been studied since the 1960s, new complexity results for them follow from this tight relationship.

Before we define all mentioned models formally, we first define a very general probabilistic game model that underlies all of them. Namely, (countable-state) Simple Stochastic Games (with rewards). (Discounted) Stochastic Games were already studied by Lloyd Shapley in the early 1950s ([Sha53]). These games are concurrent, i.e., the two players are simultaneously choosing actions independently of each other and depending on such a chosen pair of actions the game proceeds to a new state. However in this thesis, we are studying turn-based games only, i.e., at each step of the game only one player is choosing an action and he has perfect information about the whole game structure and what has happened until that moment. Simple Stochastic

Games were defined by Anne Condon ([Con92]), but they had only a finite number of states, all probabilities were equal to $1/2$ and the objective was (mainly) termination, i.e., the player receives reward 1 iff the game terminates at (reaches) a given node. As already noted by Condon, restricting the probabilities just to the $1/2$ value does not change their expressive power since any rational probabilities transitions can be constructed from such restricted transitions. On the other hand, allowing games to have an infinite number of states drastically changes their properties as we will see later. Competitive Markov Decision Processes ([FV97]) are yet another name for finite-state stochastic games used in Operation Research (OR) and economics. In order to simplify the notation we do not strictly follow the same formal notation as that used in prior work.

The rest of this chapter is organized as follows. In Section 2.1 we give a definition of (countable-state) Simple Stochastic Games and player strategies. In Section 2.2 we formally define Recursive Simple Stochastic Games (RSSGs), which are finite presentations of (countable-state) Simple Stochastic Games, and define memoryless strategies and memoryless&stackless strategies. After that we survey some of the existing results for RSSGs. In the same section, we also formally define other controlled and uncontrolled probabilistic finitely-presentable models: Stochastic Context-Free Grammar (SCFG), Multi-Type Branching Processes (MT-BPs), probabilistic Pushdown Systems (pPDSs), random walks with "back buttons", (probabilistic) 1-counter automata, Quasi-Birth-Death processes (QBDs), Tree-like QBDs and Tree-structured QBDs. In Section 2.3 we formally define what it means for two (finitely-presentable) models to be essentially equivalent, and prove such equivalences between 1-RSSGs and SCFG games; p1CAs and QBDs; pPDSs, RMCs, TL-QBDs and TS-QBDs. We conclude in Section 2.4 and present how all these models relate to each other in a simple diagram.

2.1 Preliminaries

By $\mathbb{R}_{>0} = (0,\infty)$ we denote the positive real numbers, $\mathbb{R}_{\geqslant 0} \doteq [0,\infty)$, $\overline{\mathbb{R}} \doteq [-\infty,\infty]$, $\mathbb{R}^{\infty}_{>0} \doteq (0,\infty]$, and $\mathbb{R}^{\infty}_{\geqslant 0} \doteq [0,\infty]$. The extended reals $\overline{\mathbb{R}}$ have the natural total order. We also assume the following usual arithmetic conventions on the non-negative extended reals $\mathbb{R}^{\infty}_{\geqslant 0}$: $a \cdot \infty = \infty$, for any $a \in \mathbb{R}^{\infty}_{>0}$; $0 \cdot \infty = 0$; $a + \infty = \infty$, for any $a \in \mathbb{R}^{\infty}_{\geqslant 0}$. We can naturally extend these basic arithmetic operations to matrix arithmetic operations over $\mathbb{R}^{\infty}_{\geqslant 0}$.

We use boldface letters $\mathbf{x},\mathbf{y},\mathbf{z},\ldots$ to denote variables that are vectors and we use

subscript index to refer to the value of a specific entry in such a vector. An index is either the name of a node, like in x_u, or the entry number, like in x_1. Two particular vectors (whose size differs depending on the context) are $\mathbf{1} = (1,1,\ldots)$, a vector of all ones, and $\mathbf{0} = (0,0,\ldots)$, a vector of all zeroes.

We say that a vector $\mathbf{x}^* \in (\mathbb{R}^{\infty}_{\geqslant 0})^n$ is the Least (non-negative) Fixed Point (LFP) of an equation system $\mathbf{x} = P(\mathbf{x})$ if $\mathbf{x}^*$ is a fixed point of that equation system (i.e., $\mathbf{x}^* = P(\mathbf{x}^*)$) and for any other fixed point $\mathbf{x}'$ we have $\mathbf{x}' \geqslant \mathbf{x}^*$, where $\geqslant$ is a partial entry-wise order on vectors in $(\mathbb{R}^{\infty}_{\geqslant 0})^n$.

Directed Acyclic Graph (DAG) is a directed graph in which there is no path from any node to itself. Strongly Connected Component (SCC) of a directed graph is any (inclusion-wise) maximal subset of nodes such that there is a path between any two nodes from that subset. It is a well-know fact that any directed graph can be decomposed in linear-time into SCCs that then form a DAG. The set of nodes of that DAG is the set of SCCs and there is an edge from one SCC to another if there exists an edge from any node in the former to any node in the latter.

In all the probabilistic models we define, we assume for computational purposes that all probabilities on transitions are rational, and that they are encoded in the standard way by providing numerator and denominator in binary.

Definition 2.1.1. A (countable-state zero-sum turn-based) ***Simple Stochastic Game*** $\mathcal{G}$ (with finite branching and nonnegative rewards) is a pair (S, Δ), where S is a countable set of states partitioned into three disjoint sets S_0, S_1, and S_2, and Δ is a transition relation $\subseteq S \times ((0,1] \cup \{\bot\}) \times S \times \mathbb{R}_{\geqslant 0}$. Nodes in the set S_0 are probabilistic (or controlled by the so-called "nature" player), while nodes in S_1 and S_2 belong to Player 1 and Player 2 respectively. We require the set of transitions $(u, p_{uv}, v, c_{uv}) \in \Delta$ to be finite (and allow it to be empty) for any $u \in S$. Moreover, for $(u, x, v, c) \in \Delta$ we have $x \in (0,1]$ if $u \in S_0$ and $x = \bot$ if $u \in S_1 \cup S_2$. We also require that if $u \in S_0$ then p_{uv} is a probability distribution over the possible next states, i.e., $\sum_v p_{uv} = 1$.

A Simple Stochastic Game $\mathcal{G} = (S, \Delta)$ induces the following transition system with rewards. A tuple $(u, p_{uv}, v, c_{uv}) \in \Delta$, where $p_{uv} \in (0,1]$, states that from the state $u \in S_0$ we move to the state $v \in S$ with probability p_{uv} and such a transition yields reward c_{uv} to Player 1 (and the same cost to Player 2). On the other hand, a tuple $(u, \bot, v, c_{uv}) \in \Delta$ states that from the state $u \in S_1 \cup S_2$, Player 1 or 2 (depending on who is the owner of u) can choose to move to the state v which gives Player 1 reward c_{uv} (and incurs cost c_{uv} to Player 2). Notice that, if we fix $S_1 = S_2 = \emptyset$ and forget about the players, we will get a classical model of (countable-state) Markov Chains with (nonnegative) rewards.

A ***play*** is a (finite or infinite) path in the transition system of $\mathcal{G}$. It starts at one of the states and proceeds by taking one of the transitions available from the current state. A ***strategy*** σ_i for player i, $i \in \{1,2\}$, is a function $\sigma_i : S^*S_i \mapsto S$, where, given the history $ws \in S^*S_i$ of the play so far, with $s \in S_i$ (i.e., it is player i's turn to make a move), $\sigma_i(ws) = s'$, where $(s, \bot, s', c) \in \Delta$ for some c, determines the next move of player i. (We could also allow randomized strategies, but as we will see, this will not be necessary for the games we are considering in this thesis.) Let Ψ_i denote the set of all strategies for player i. A pair of strategies σ_1, σ_2, were $\sigma_i \in \Psi_i$, induces in a straightforward way a Markov chain with rewards $M^{\sigma_1,\sigma_2} = (S^*, \Delta')$, whose set of states is the set of histories S^* and the transition relation Δ' is a subset of $S^* \times (0,1] \times S^* \times \mathbb{R}_{\geqslant 0}$. Moreover, $(ws, p, wst, c) \in \Delta'$ (where $w \in S^*$, $s,t \in S$) iff $(s,p,t,c) \in \Delta$ and $s \in S_0$ or $(s,1,t,c) \in \Delta$ and $(\sigma_1 \cup \sigma_2)(ws) = t$.

If we impose that each play of the game $\mathcal{G}$ starts from the same initial state u, then for strategies σ_1 and σ_2 we obtain a Markov Chain M^{u,σ_1,σ_2}. For M^{u,σ_1,σ_2} we can define in a straightforward way a random variable $X_u^{k,\sigma_1,\sigma_2}$ whose value is the reward at the k-th step (when starting from u; of course if all the paths starting in u are shorter than k, then its value is 0 by default).[1] There are several different ways to define the outcome of M^{u,σ_1,σ_2} (i.e., the payoff to Player 1), $r_u^{\sigma_1,\sigma_2}$, based on these random variables X^{k,σ_1,σ_2} (see [Put94, NS03]):

- ***total expected reward*** simply defines
$$r_u^{\sigma_1,\sigma_2} = E\left(\sum_{k=1}^{\infty} X_u^{k,\sigma_1,\sigma_2}\right)$$
- ***the discounted total reward*** with the discount factor $\lambda < 1$ sets
$$r_u^{\sigma_1,\sigma_2} = E\left(\sum_{k=1}^{\infty} \lambda^{k-1} X_u^{k,\sigma_1,\sigma_2}\right)$$
- and finally, ***the limit-average reward*** assigns
$$r_u^{\sigma_1,\sigma_2} = \liminf_{m\to\infty} E\left(\frac{\sum_{k=1}^{m} X_u^{k,\sigma_1,\sigma_2}}{m}\right)$$

where EX denotes the expected value of random variable X. Notice that, since the value of $X_u^{k,\sigma_1,\sigma_2}$ is always positive, the order in which we sum these random variables does not matter, and in fact by the linearity of expectations we can just sum the

[1] We will not define this random variable formally. We could define a probability space with a σ-field over the subsets of plays, but such a formal treatment will not be necessary for us as these concepts are not used explicitly anywhere in this thesis. For their formal definition see, e.g., [Bil79].

expected values of these variables instead and obtain the same outcome value. Also, when the objective of the game is total expected reward, the outcome can easily be equal to ∞, while for the other two objectives the outcome can always be bounded by a function in the size of the input and λ. In this thesis we study only games with the total expected reward objective, since all the objectives we are going to define later can easily be encoded with such a criterion.

Recall that Simple Stochastic Games are zero-sum games, i.e., if Player 1 earns x then Player 2 loses x. Player 1 will be trying to maximize the outcome of the game while Player 2 will be trying to minimize it in order to minimize his loss. Henceforth, we will be referring to Player 1 as the ***maximizer*** and Player 2 as the ***minimizer***.

We say that the game $\mathcal{G}$ that starts in u has a ***value*** (or is ***determined***) if the following holds: $\sup_{\sigma_1 \in \Psi_1} \inf_{\sigma_2 \in \Psi_2} r_u^{\sigma_1,\sigma_2} = \inf_{\sigma_2 \in \Psi_2} \sup_{\sigma_1 \in \Psi_1} r_u^{\sigma_1,\sigma_2}$, and denote such a value by r_u^*. If a game has a value, then player i's strategy is said to be *optimal* if it guarantees player i the optimal reward r_u^* no matter what the other player does. Formally, a strategy σ_1^* is said to be optimal for Player 1 if $\inf_{\sigma_2 \in \Psi_2} r_u^{\sigma_1^*,\sigma_2}$ is equal to r_u^* and similarly, a strategy σ_2^* is said to be optimal for Player 2 if $\sup_{\sigma_1 \in \Psi_1} r_u^{\sigma_1^*,\sigma_2}$ is equal to r_u^*. A strategy is said to be ε-optimal if its value guarantees a reward at most ε away from the optimal value no matter what the other player does, i.e., for the maximizer's strategy σ_1^*: $\inf_{\sigma_2 \in \Psi_2} r_u^{\sigma_1^*,\sigma_2} \geqslant r_u^* - \varepsilon$ and for the minimizers's strategy σ_2^*: $\sup_{\sigma_1 \in \Psi_1} r_u^{\sigma_1,\sigma_2^*} \leqslant r_u^* + \varepsilon$.

From a well-known result due to Martin ([Mar98]) we know that all *Blackwell games* are determined and both players have ε-optimal (randomized) strategies. That class of games subsumes denumerable stochastic games with bounded Borel payoff functions ([MS98]). We instantly know from this that any Simple Stochastic Game $\mathcal{G}$ is determined when the objective of the game is the discounted total reward or the limit-average reward criterion, but we cannot apply that result directly to games with total expected reward criterion unless we somehow show that the payoff is bounded by some finite value. For this reason, we have to prove the determinacy of the games studied in Chapter 4 ourselves, since their total reward cannot be bounded by any finite value.

2.2 Definitions

In this section we define several formalisms that are finitely-presentable and yet are able to describe subclasses of infinite-state Markov Chains or Simple Stochastic Games. It order to simplify the definitions, only the first two described models, Recursive Sim-

ple Stochastic Games (RSSGs) and Stochastic Context-Free Grammar (SCFG) games, incorporate controllers and rewards on transitions. The rest of the models are just finite representations of potentially infinite-state Markov Chains (without rewards) that can easily be seen as a special case of Simple Stochastic Games. These models could easily be extended to include rewards and players, but since their controlled versions will not be studied later in this thesis (apart from the controlled version of p1CAs without rewards in Chapter 3), we do not define them formally in order to simplify the definitions.

2.2.1 Recursive Simple Stochastic Games and related models

Let us firstly define Recursive Simple Stochastic Games (RSSGs) informally. A small example RSSG is depicted in Figure 2.1. Intuitively, an RSSG is a set of components (named A and B in our example) consisting of a finite number of entries, exits, internal nodes and boxes which have their own entries and exits. All these nodes, entries and exits are connected by transitions. Just like for Simple Stochastic Games, some of these nodes are controlled by one of two players, while some others are stochastic and all the transitions out of them are probabilistic. In fact, a component without any box inside of it is essentially a finite-state Simple Stochastic Game. Each box is mapped to one of the components so that every time we reach an entry of a box, we jump to the corresponding entry of the component it is mapped to. When/if we finally reach an exit node of an component, we jump back to the respective exit of the box that we used to enter this component. This process models, in an obvious way, function invocation in probabilistic procedural programs. Every potential function call is represented by a box. Entry nodes represent parameter values passed to the function, while exit nodes represent returned values. We can view the players as if one of them is trying to optimize the program while the other one acts as a malicious environment in which such a program executes.

Now formally, a ***Recursive Simple Stochastic Game*** (RSSG) is a tuple $A = (A_1, \ldots, A_k)$, where each *component* $A_i = (N_i, B_i, Y_i, En_i, Ex_i, pl_i, \delta_i, \xi_i)$ consists of:

- A set N_i of ***nodes***, with a distinguished subset En_i of *entry* nodes and a (disjoint) subset Ex_i of *exit* nodes.
- A set B_i of ***boxes***, and a mapping $Y_i : B_i \mapsto \{1, \ldots, k\}$ that assigns to every box (the index of) a component. To each box $b \in B_i$, we associate a set of *call ports*, $Call_b = \{(b, en) \mid en \in En_{Y(b)}\}$, and a set of ***return ports***, $Ret_b = \{(b, ex) \mid ex \in$

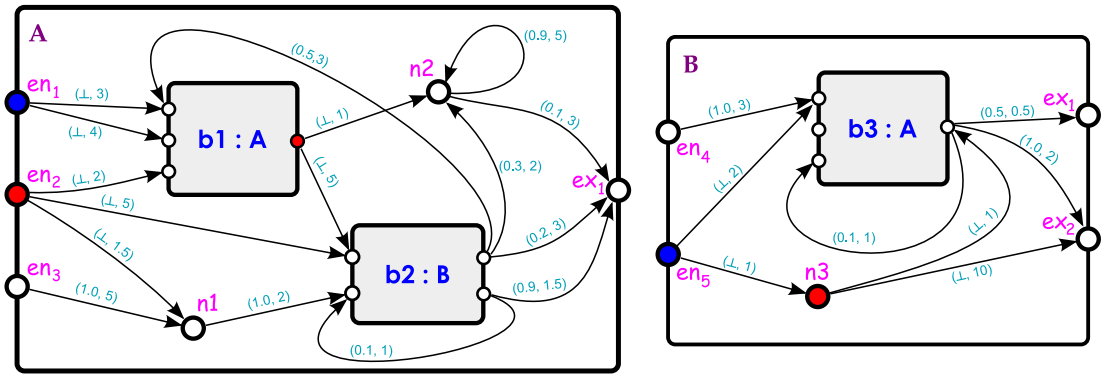

Figure 2.1: An example (multi-exit) Recursive Simple Stochastic Game (RSSG) consisting of two components, A and B. Red vertices belong to player 1, blue ones to player 2, and white ones to "nature" (i.e., are random). Each box (labeled, e.g., b1:A) has a name (here b1) and is mapped to one of the components (here A). Each edge has a label whose first entry represents the probability of using that edge or is equal to $\perp$ if the origin of that edge is a controlled node, while the second entry represents the reward assigned to that edge.

$\mathrm{Ex}_{Y(b)}\}$. Let $Call^i = \cup_{b\in B_i} Call_b$, $Ret^i = \cup_{b\in B_i} Ret_b$, and let $Q_i = N_i \cup Call^i \cup Ret^i$ be the set of all nodes, call ports and return ports; we refer to these as the ***vertices*** of component A_i.

- A mapping $pl_i : Q_i \mapsto \{0,1,2\}$ that assigns to every vertex a ***player*** (Player 0 represents "chance" or "nature"). We assume $pl_i(ex) = 0$ for all $ex \in Ex_i$.

- A transition relation $\delta_i \subseteq Q_i \times ((0,1] \cup \{\perp\}) \times Q_i \times \mathbb{R}_{\geqslant 0}$, where for each tuple $(u,x,v,c_{uv}) \in \delta_i$ we have: $u \in (N_i \setminus Ex_i) \cup Ret^i$ is the source control state, $v \in (N_i \setminus En_i) \cup Call^i$ is the destination control state, x is either (i) the transition probability, $p_{uv} \in (0,1]$, if $pl_i(u) = 0$, or (ii) $x = \perp$ if $pl_i(u) = 1$ or 2, and finally $c_{uv} \in \mathbb{R}_{\geqslant 0}$ is the nonnegative reward associated with this transition. We assume there is at most one transition from u to v in δ for any pair of control states $u,v \in V$. Also, for computational purposes we assume that the given probabilities p_{uv} and rewards c_{uv} are rational. Probabilities must also satisfy consistency: for every $u \in pl_i^{-1}(0)$, $\sum_{\{v' | (u,p_{uv'},v',c_{uv'})\in\delta_i\}} p_{uv'} = 1$, unless u is a call port or an exit node, neither of which have outgoing transitions, in which case by default $\sum_{v'} p_{uv'} = 0$.

- Finally, the mapping $\xi_i : Call_i \cup Ex_i \mapsto \mathbb{R}_{\geqslant 0}$ maps each call port of a box inside component A_i and each exit of A_i to a nonnegative rational value.

We use the symbols (N, B, Q, δ, etc.) without a subscript, to denote the union over all components. Thus, e.g., $N = \cup_{i=1}^{k} N_i$ is the set of all nodes of A, $\delta = \cup_{i=1}^{k} \delta_i$ the set of all transitions, etc. An RSSG A defines a global denumerable Simple Stochastic Game with rewards, $\mathcal{G}_A = (V, \Delta)$ as follows. The ***global states*** V are either a special terminal state ✠ or a pair of the form $\langle \beta, u \rangle \subseteq B^* \times Q$, where $\beta \in B^*$ is a (possibly empty) sequence of boxes and $u \in Q$ is a vertex of A. The states V and transitions Δ are defined inductively as follows:

1. $\langle \varepsilon, u \rangle \in V$, for $u \in Q$, where ε denotes the empty string.
2. if $\langle \beta, u \rangle \in V$ & $(u, x, v, c) \in \delta$, then $\langle \beta, v \rangle \in V$ and $(\langle \beta, u \rangle, x, \langle \beta, v \rangle, c) \in \Delta$.
3. if $\langle \beta, (b, en) \rangle \in V$ & $(b, en) \in Call_b$, then $\langle \beta b, en \rangle \in V$ & $(\langle \beta, (b, en) \rangle, 1, \langle \beta b, en \rangle, \xi((b, en))) \in \Delta$.
4. if $\langle \beta b, ex \rangle \in V$ & $(b, ex) \in Ret_b$, then $\langle \beta, (b, ex) \rangle \in V$ & $(\langle \beta b, ex \rangle, 1, \langle \beta, (b, ex) \rangle, 0) \in \Delta$.
5. $(\langle \varepsilon, ex \rangle, 1, ✠, \xi(ex)) \in \Delta$, for each $ex \in Ex$.

The partition of V into $V_0 \cup V_1 \cup V_2$ is given as follows: ✠ and nodes of the form $\langle \beta, u \rangle$, where $u \in Call \cup Ex$, belong to the set of probabilistic nodes V_0. For all the other nodes of V we have $\langle \beta, u \rangle \in V_{pl(u)}$. We consider $\mathcal{G}_A$ with various ***initial states*** of the form $\langle \varepsilon, u \rangle$, denoting it by $\mathcal{G}_A^u$. The only ***terminating state*** of $\mathcal{G}_A$, i.e., a state without any outgoing transitions, is ✠.[2] In $\mathcal{G}_A$ Player 1 (the maximizer) has a set of strategies Ψ_1 and Player 2 (the minimizer) has a set of strategies Ψ_2. Conventionally, we will denote the maximizer's strategy by σ and the minimizer's strategy by τ. An RSSG A and two strategies σ and τ induce a Markov chain with rewards $M_A^{\sigma,\tau} = (V^*, \Delta')$.

A maximizing (minimizing) Recursive Markov Decision Process (RMDP) is an RSSG where $V_2 = \emptyset$ ($V_1 = \emptyset$, respectively). Furthermore, a Recursive Markov Chain (RMC) is an RSSG where both V_1 and V_2 are empty.[3] Notice that, the game $\mathcal{G}_A$ is already a Markov Chain if A is an RMC, since it then contains only probabilistic states. Several special subclasses of these models were distinguished in [EY05s]. The most important subclass is the class that restricts each component (and thus also all the boxes) to have just one exit. Such a subclass of our models will be referred to by adding prefix '1-exit' (thus 1-exit RMCs/RMDPs/RSSGs or 1-RMCs/1-RMDPs/1-RSSGs for

[2] In fact in [EY05i, EY06s] where RSSGs were defined for the first time and Chapter 4, all nodes of the form $\langle \varepsilon, ex \rangle$ for $ex \in Ex$ are terminal. We added an extra state ✠ and a transition from each node $\langle \varepsilon, ex \rangle$ to ✠ in order to unify the expositions of all the different objectives we are going to consider. As we will see later, this does not change much for their analysis.

[3] RMCs were firstly defined in [EY05s]. They did not contain rewards on transitions, but of course they can easily be described using our richer definition.

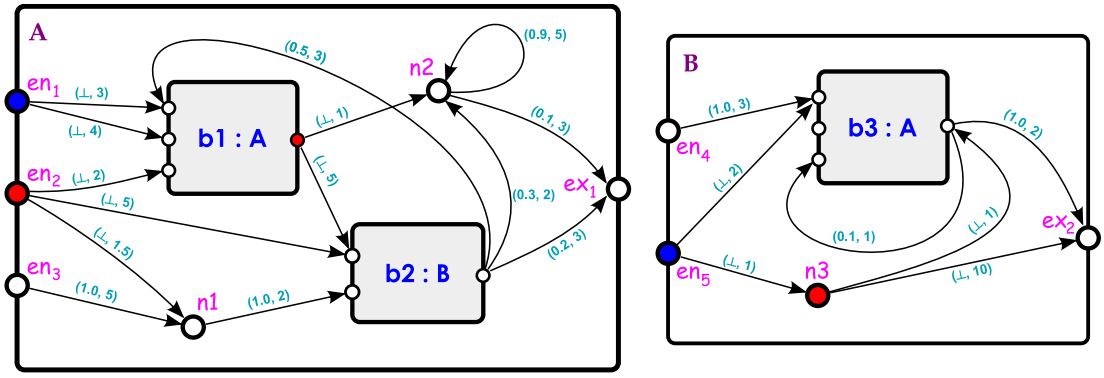

Figure 2.2: An example 1-exit Recursive Simple Stochastic Game (1-RSSG) consisting of two components, A and B. Red vertices belong to player 1, blue ones to player 2, and white ones to "nature" (are random). Each box (labelled, e.g., b1:A) has a name (here b1) and is mapped to a component (here A). Each edge has a label whose first entry is equal to the probability of using that edge or is $\perp$ if the origin of that edge is a controlled node, while the second entry represents the reward assigned to that edge.

short). We can see an example 1-RSSG in Figure 2.2 (compare it with Figure 2.1). Intuitively, the "1-exit" restriction essentially restricts these finite-state subroutines so they do not return a value, unlike for general RSSGs in which they can return distinct values. Many problems that are hard or undecidable for the general class of RSSGs become a lot easier and tractable once the model is restricted to the 1-exit subclass. If we want to emphasize that we are talking about the general unrestricted model, we will refer to it as a multi-exit RMC/RMDP/RSSG.

Now we will describe three types of objectives that were studied so far in the work on RSSGs and phrase them as RSSGs with a total expected reward criterion.

- The ***termination probability*** — optimizing the probability of reaching a subset H of the set of states $\{\langle \varepsilon, ex\rangle | ex \in Ex\}$ when starting at a given state $\langle \varepsilon, u\rangle$ where $u \in V$. This objective was studied for RMCs without rewards in [EY05s, EKM04] and for RMDPs/RSSGs without rewards in [EY05i]. For a given RSSG A without rewards, we construct an RSSG A' with rewards by simply setting $\xi(ex) = 1$ for all $\langle \varepsilon, ex\rangle \in H$ and setting all other rewards to zero. The optimal expected reward in A' is equal to the optimal probability of reaching ✠ in A, but that value is equal to the optimal probability of reaching H since ✠ is reached with probability 1 from the set of states H.

- The ***reachability objective*** — optimizing the probability of reaching a given

node $u \in Q$ in an RSSG A without rewards or equivalently reaching any of the states $\{\langle \beta, u\rangle | \beta \in B^*\}$ during a play that starts at a given state $\langle \varepsilon, v\rangle$ ($v \in V$) of the underlying Simple Stochastic Game $\mathcal{G}_A$. This objective was studied in [BBFK06].[4] We can encode it as an RSSG with rewards A' by slightly modifying A and adding rewards to it. We replace all the transitions out of the node u with a single self-loop transition that has probability one and reward 0. All the transitions that lead to u in A are assigned reward 1, and reward 0 is assigned to all the other transitions. It is easy to see that the optimal total reward in A' is equal to the optimal probability of reaching any of the states $\{\langle \beta, u\rangle | \beta \in B^*\}$ in $\mathcal{G}_{A'}$.

- The ***total reward in a positive reward setting*** — we are optimizing the undiscounted sum of all the rewards accumulated throughout a play of a given RSSG with positive rewards. More precisely, all the rewards on transitions (apart from rewards that are assigned by ξ) are required to be positive (not just nonnegative). Notice that such an optimal value can easily be equal to ∞. This model is studied in Chapter 4 (and in [EWY08i] which is a shorter version of that chapter).

We will be referring to the first class of games as termination RSSGs, the second one as reachability RSSGs and the last one as RSSGs with positive rewards. The determinacy of the first two classes follows from Martin's theorem ([Mar98, MS98]), because the total payoff is always $\leqslant 1$. The payoffs in RSSGs with positive rewards can be unbounded and we prove their determinacy in Chapter 4.

Once we know an optimal value of a game exists, several different questions can be posed about it. The ***decision problem*** is the following: Is the value of a given game $\triangle p$, where $p \in \mathbb{Q}^{\infty}_{\geqslant 0}$ and $\triangle \in \{=, \leqslant, <, \geqslant, >\}$? Furthermore, the decision problem can be ***quantitative*** when we do not impose any restrictions on the value of p or ***qualitative*** when p is the minimal or the maximal value such an objective can achieve, e.g., 0 or 1 for probabilities. On the other hand, ***the approximation problem*** asks to compute the optimal value within a given absolute or relative error $\varepsilon > 0$. Notice that once we solve the decision problem, we can also approximate the optimal values if they are finite. We simply perform a binary search in the interval of all possible reward values $[0, M]$ (e.g., $M = 1$ for probabilities). If the maximum reward cannot be a priori bounded, e.g., for RSSGs with positive rewards, we can just start with $r = 1$ and repeatedly ask if the value is $\leqslant r$ and if it is not double the value of r. If the optimal value is finite,

[4] In fact, slightly more general objectives were studied in [BBFK06], but it is irrelevant to our study. Also, they were formulated in terms of controlled probabilistic Pushdown Systems (pPDSs) which is an equivalent model to RSSGs (see Section 2.2.4 for a definition of uncontrolled pPDSs without rewards).

this procedure will eventually terminate finding some upper bound on the optimal value of the game. Once we have the upper bound M, we can proceed further exactly as before. As we will soon see, these questions have (very) different computational complexity depending on the class of objectives being considered.

On the other hand, notice that even knowing the exact value of the game, does not imply that we can compute an optimal strategy that achieves it. Martin's theorem shows the existence of ε-optimal (randomized) strategies in a nonconstructive way, so optimal strategies may not exist and even if they do, they may not be computable. Indeed, we can see in Figure 2.3 an example reachability 1-RSSG that has no optimal strategy, only ε-optimal ones. Moreover, in a general setting of RSSGs with nonnegative rewards, they may not even be an ε-optimal strategy for any $\varepsilon > 0$, even if the game has a value. We can see such an example in Figure 2.4; the maximizer does not have an ε-optimal strategy to achieve the value of the game, ∞, for any $\varepsilon > 0$. In conclusion, another important sort of questions is whether in a given class of games optimal strategies always exist, how fast can they be computed (if at all), and what kind of strategies suffices to achieve the optimal values. Notice that a strategy is a function whose domain is the set of all possible histories of a play. Such a set is infinite even if the game has a finite number of states. Thus, we need to find some succinct finite representation for such strategies. An important class of strategies is the class of ***memoryless strategies***, strategies that apart from the current state do not depend on the history of the play. Formally, a strategy σ is memoryless if $\sigma(ws) = \sigma(s)$ for any $w \in S^*$ and $s \in S$ (where S is the set of states of the game). This gives a finite representation for finite state stochastic games since the set of states S is finite and it suffices to define such a strategy just for each element of S. However, strategies of that sort do not have a finite representation in general (and may not even be computable) for the infinite state stochastic games that we mainly consider in this thesis. A more restricted class of strategies specific to RSSGs is the class of ***stackless & memoryless strategies*** (SM strategy), strategies that not only do not depend on the history of the play, but also do not depend on the current stack content (hence only on the current control state). Formally, a strategy σ is SM if it is memoryless and $\sigma(\langle \beta, u \rangle) = \sigma(\langle \varepsilon, u \rangle)$ for any $\beta \in B^*$ & $u \in Q$. Hence, we need to specify such an SM strategy only for each control state, the set of which is finite.

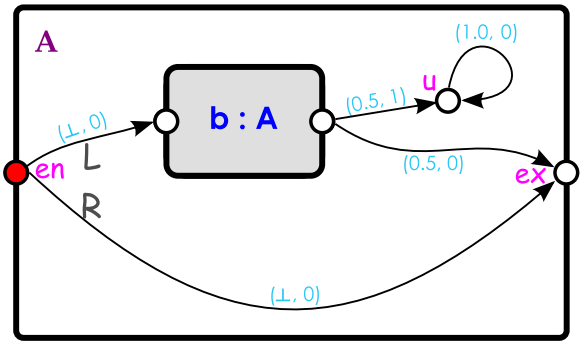

Figure 2.3: An example reachability (maximizing) 1-RMDP where the maximizer has no optimal strategy. A strategy that picks Left k times and then picks Right reaches node u with probability $(1-2^{-k})$, but no strategy achieves 1, which is the supremum value of this game. Obviously, for $k = \log_2 \frac{1}{\varepsilon}$, the strategy just described is an ε-optimal strategy. Notice that the two SM strategies are the worst possible, reaching u with probability 0.

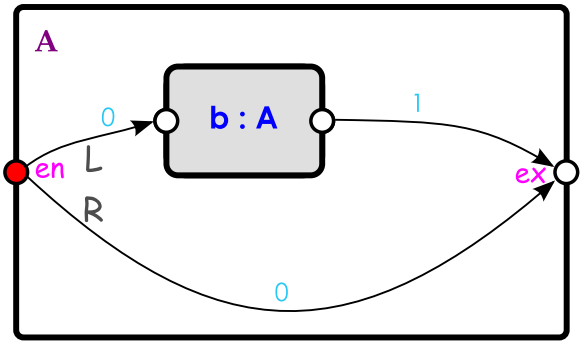

Figure 2.4: An example maximizing 1-RMDP with nonnegative rewards where the maximizer has no ε-optimal strategy. A strategy that picks Left k times and then picks Right grants him a payoff of k, but no strategy achieves ∞, which is the optimal value of this game. Obviously no strategy can obtain payoff that is ε close to ∞ for any finite ε. Notice that the two SM strategies are the worst possible obtaining the total reward 0.

The state of the art

We now present some known upper and lower complexity bounds for answering the just posed problems for these three classes of games, but first we will define two other computational problems, which have an intriguing complexity status, and that can be reduced to some of the questions we will be dealing with.

The ***square-root sum problem*** (SQRT-SUM problem) asks, given natural numbers $(d_1,\ldots,d_n) \in \mathbb{N}^n$ and $k \in \mathbb{N}$, whether $(\sum_i \sqrt{d_i}) \leqslant k$. This problem can be solved in PSPACE, but it is a major open problem since the 1970s ([GGJ76]) whether it can be placed in NP or even polynomial time hierarchy. Many problems in computational geometry are at least as hard as this problem, e.g., the Traveling Salesmen problem in Euclidian space.

The other problem, which is as hard as SQRT-SUM via a PTIME Turing reduction, is the Positive Straight-Line-Program problem (PosSLP) studied in [ABKM06]. The PosSLP problem asks whether a given arithmetic circuit with integer inputs, and gates $\{+,*,-\}$, outputs a positive number or not. It was shown there that PosSLP is hard under PTIME Turing reductions for the entire class of decision problems that can be decided in polynomial time in the discrete Blum-Shub-Smale (BSS) model of computation over the reals using rational constants ([BCSS98]). The discrete BSS model is basically equivalent to the unit-cost algebraic RAM model of computation, i.e., all arithmetic operations are performed in a single time unit no matter how big the operands are. Intuitively, we are just counting the number of arithmetic operations necessary to perform the task. (In Chapter 3 we show a PTIME algorithm for approximating the fundamental G matrix of a given Quasi-Birth-Death process in the unit-cost algebraic RAM model of computation.) Notice that the standard Turing machine model can simulate the unit-cost algebraic RAM model, but with the cost of each operation being polynomial in the size of the operands. It has to be noted that the size of the operands can grow exponentially in the number of arithmetic operations performed. It was shown in [Tiw92] that SQRT-SUM problem can be solved in PTIME in the BSS model, while we still do not even know how to solve it in NP in the standard Turing model. Moreover, it was shown in [ABKM06] that PosSLP, and thus also SQRT-SUM, can be decided in the 4th level of the Counting Hierarchy (an analog of the polynomial time hierarchy for counting classes like #P (a class defined in [Val79])) which is a slight improvement over PSPACE.

Termination RSSGs. The decision problem for termination RMCs (computing the termination probabilities) was shown in [EY05s] to be expressible in Existential The-

ory of the Reals (ET$\mathbb{R}$). Deep results in [GV88, Can88, Ren92] shows that ET$\mathbb{R}$ can be decided in PSPACE. As described before, an approximation of the termination probabilities in PSPACE easily follows from the solution to the decision problem. As for the lower bounds, the decision problem was shown to be as hard as the SQRT-SUM and PosSLP problems via many-one reductions ([EY05s, EY07]). Furthermore, any non-trivial approximation, to within $\varepsilon < 1/2$ additive error, of the termination probability is in fact SQRT-SUM-hard, and so is the problem of checking whether the termination probability is equal to 1 ([EY05s, EY07]). However, for 1-exit RMCs, checking if the termination probability is equal to 1 was shown to be solvable in PTIME ([EY05s]). In the controlled setting, a quantitative decision question about termination multi-exit RMDPs (and RSSGs) was shown to be undecidable ([EY05i]). However, when RMDPs are restricted to the 1-exit subclass, their qualitative decision problems can be solved in PTIME ([EY05i, EY06s]). It follows that the qualitative decision problem about 1-RSSGs can be solved in NP$\cap$co-NP. Quantitative decision questions about 1-RMDPs/1-RSSGs can be expressed in ET$\mathbb{R}$, and hence they can be solved in PSPACE. Furthermore, in termination 1-RSSGs both players have always SM optimal strategies that can be computed in PSPACE.

Reachability RSSGs. Although this setting looks similar to the previous one (since we are trying to reach a given set of nodes and it does not matter how the game proceeds after that) the complexity of computing the values and the kind of strategies needed to achieve them in reachability RMDPs/RSSGs are strikingly different. Again, the decision problems for multi-exit reachability RMDPs (and RSSGs) are undecidable[5]. As already mentioned before (see Figure 2.3), in reachability maximizing 1-RMDPs there may not exist any optimal strategy, we only know the existence of ε-optimal strategies. (On the other hand, for reachability minimizing 1-RMDPs an optimal strategy always exists as a consequence of well-known results for countable-state MDPs (see, e.g., [Put94])). The quantitative decision questions about the optimal values (and even their approximation) are wide open as well is checking if the supremum probability over all strategies is equal to 1 or not. On the other hand, it was shown in [BBFK06], building on [EY06s], that we can decide in PTIME whether there exists an (optimal) strategy in a maximizing 1-RMDP that reaches the desired control state with probability 1. Notice that the already mentioned example (Figure 2.3) shows that such an optimal strategy may not exist even if the optimal value is indeed

[5]This fact has not been formally stated anywhere, but it trivially follows from the undecidability proof for multi-exit termination RMDPs; one can just create a new component with a single box, a copy of the gadget used in [EY05i], and a single internal node reachable only from the n-th exit of that gadget.

1. Moreover, even if an optimal strategy exists, there may be no optimal strategy that is memoryless and stackless ([BBFK06]).

Total reward in a positive reward setting. Although this setting seems to be more general than the two previous ones, this class of games has a lot nicer properties than both of them. Again, multi-exit RSSGs with positive rewards are undecidable (Theorem 4.4.13 in Chapter 4). On the other hand, unlike terminating 1-RSSGs and reachability 1-RSSGs, the optimal value of an 1-RSSGs with positive rewards is always a rational number (or is equal to ∞). Moreover, such a value can be computed exactly for both maximizing and minimizing 1-RMDPs with positive rewards in PTIME. This means that we can answer their quantitative questions, not just qualitative ones, in PTIME. Optimal SM strategies always exist for both players in 1-RSSGs with positive rewards and it easily follows that their optimal value can be computed in NP $\cap$ coNP. The optimal strategies for both players can be found via a simultaneous strategy improvement algorithm. A corollary is that computing the game value and optimal strategies for these games is contained in the class PLS of polynomial local search problems ([JPY88]). Whether this strategy improvement algorithm runs in the worst-case in PTIME is open, just like its version for finite-state SSGs. We know that in 1-exit RMDPs with nonnegative rewards, there may not even be ε-optimal strategies (see Figure 2.4), hence the assumption of all rewards being positive is crucial. This assumption essentially causes all nonterminating plays to have total reward ∞ and hence the optimal reward of a game to be ∞ if the game does not terminate with probability 1. Without that assumption a game may have a finite total reward even if it terminates with probability 0.

2.2.2 Stochastic Context-free Grammar games

We define here a natural probabilistic game model that, as will see later in Section 2.3.1, has exactly the same expressive power as 1-exit RSSGs. This model is essentially an extension of the well-known context-free grammars to a probabilistic and controlled setting. Terminal symbols in our grammars are completely ignored, since the rewards are associated with grammar rules only. The nonterminals in a Stochastic Context-Free Grammar (SCFG) game are split into three disjoint sets of nonterminals: *random* nonterminals, *player-1* nonterminals and *player-2* nonterminals, controlled by Player 1 and Player 2, respectively. For each *random* nonterminal X, we are given a probability distribution on the rules $(X \mapsto \alpha)$ where X appears on the left hand side. Starting from some single nonterminal, a play of the game builds a derivation of the grammar.

The derivation proceeds in left-most manner, by choosing a remaining ***left-most*** nonterminal, S, and expanding it. The precise derivation law (left-most, right-most, etc.) does not effect the game value in the strictly positive reward setting, but as we will see later it does if we allow 0 rewards. If S belongs to *random*, it is expanded randomly by choosing a rule $S \to \alpha$, according to a given probability distribution over the rules whose left hand side is S. If S belongs to *player-i*, then player i chooses which grammar rule to use to expand this S. The play continues, either forever, or until the empty string is derived. Player 1's goal is to maximize the total (possibly infinite) expected reward gained during the entire derivation, while Player 2's goal is to minimize that reward.

A Stochastic Context-Free Grammar (SCFG) is an SCFG game with no players, i.e., all nonterminals are random. Let us now present an example SCFG with rewards given by the following grammar rules: $\{X \overset{(1/3,3)}{\longmapsto} XX\ ;\ X \overset{(2/3,2)}{\longmapsto} \varepsilon\}$. Here X is the only nonterminal (and there are no terminal symbols). The pair (p,c) of quantities labeling a rule denotes the probability, p, of that rule firing, and the reward, c, accumulated for each use of that rule during a derivation. Consider now a random derivation of this grammar, starting from the nonterminal X. The derivation terminates when it reaches the empty string ε. What is the expected total reward accumulated during the entire derivation? It is not hard to see that if we let x denote the total expected reward, then x must satisfy the following equation: $x = (1/3 * (3 + (x + x))) + (2/3 * 2) = (2/3)x + (7/3)$. Therefore, the total expected reward is the unique solution to this equation, namely $x = 7$. Note that, in general, such a derivation may not terminate with probability 1, and that the expected reward need not be finite (consider the same grammar with modified probabilities: $\{X \overset{(2/3,3)}{\longmapsto} XX\ ;\ X \overset{(1/3,2)}{\longmapsto} \varepsilon\}$).

Now formally, a ***Stochastic Context-Free Grammar (SCFG) game*** (with left-most derivation), is given by $G = (V, R)$, where $V = V_0 \cup V_1 \cup V_2$ is a set of nonterminals, which is partitioned into three disjoint sets: V_0 are the probabilistic nonterminals (controlled by nature); V_1 and V_2, the nonterminals controlled by Player 1 and Player 2, respectively. Set R is a set of rules, where each rule $r \in R$ has the form $r = (X, p_r, c_r, Z_r)$, where $X \in V$, $p_r \in (0,1]$ is a (rational) probability if $X \in V_0$ and $p_r = \perp$ otherwise; $c_r \in \mathbb{Q}_{>0}$ is a rational reward, and $Z_r \in V^*$ is a (possibly empty) string of nonterminals. For each nonterminal, X, let $R_X \subseteq R$ denote the set of rules that have X on the left hand side. For each $X \in V_0$ we have $\sum_{r=(X,p_r,c_r,Z_r)\in R_X} p_r = 1$ (i.e., a probability distribution is assigned to each R_X). This game defines a (countable) Simple Stochastic Game $\mathcal{G} = (S, \Delta)$. The set of states of the game S is a subset of V^*, i.e., strings of

nonterminals. A game starts in a state consisting of a single terminal $X \in V$. In each round, if the current state is $\mathcal{S} = X_1 \dots X_k$, we proceed by a left-most derivation law as follows: choose a rule $r = (X_1, p_r, c_r, Z_r) \in R_{X_1}$. If $X_1 \in V_0$ then the rule r is chosen probabilistically among the rules in R_{X_1}, according to the probabilities p_r. If $X_1 \in V_i$, $i \in \{1,2\}$, then the rule r is chosen by player i. After the choice is made, the play moves to the new state $Z_r X_2 \dots X_k$. The reward gained in that round by Player 1 is c_r. The game continues until (and unless) we reach the empty-string state $\mathcal{S} = \varepsilon$.

It is easy to modify this definition in order to define SCFG games with a right-most derivation law. We could also define SCFG games with a simultaneous derivation law, i.e., all nonterminals in the current state are expanded by simultaneously applying to them one of their associated grammar rules. Formally, for each X_i, $i = 1, \dots, k$, we choose a rule $r(i) \in R_{X_i}$. For $X_i \in V_0$ the rule $r(i)$ is chosen probabilistically, with probability $p_{r(i)}$ and for $X_i \in V_i$, $i \in \{1,2\}$, player i chooses the rule $r(i)$. After the choices have been made, the play moves to new state $\mathcal{S}' = Z_{r(1)} Z_{r(2)} \dots Z_{r(k)}$. The reward gained in that round by Player 1 is $\sum_{(i,j)} c_{r(i,j)}$. These games are related to Multi-Type Branching Processes (MT-BPs) that are presented in the next section.

To see the difference between left-most, right-most and simultaneous derivation, let us present some simple examples for which the game value wildly differs (between 0 and ∞) depending on the derivation law used. Indeed, consider the purely deterministic context-free grammar given by the rules: $\{X \overset{(\perp,0)}{\mapsto} XY \; ; \; X \overset{(\perp,0)}{\mapsto} \varepsilon \; ; \; Y \overset{(\perp,7)}{\mapsto} \varepsilon \}$, where X and Y are nonterminals belonging to the maximizing player, Player 1. Suppose the initial nonterminal is X. If the deterministic game proceeds by left-most derivation, it is easy to see that there is no optimal strategy for maximizing Player 1's total payoff. Indeed, there aren't even any ε-optimal strategies, because the supremum is ∞. In fact, if Player 1 uses n times the rule $X \overset{(\perp,0)}{\mapsto} XY$ to expand the left-most X in the derivation, next uses $X \overset{(\perp,0)}{\mapsto} \varepsilon$, and finally uses n times $Y \overset{(\perp,7)}{\mapsto} \varepsilon$ to expand all n remaining Y nonterminals, the total reward is $7 * n$. But no single strategy will gain ∞ reward. Note in particular that any "stackless and memoryless" strategy, which always picks one fixed rule for each nonterminal, regardless of the history of play and the remaining nonterminals (the "stack" in this context), is the worst strategy possible; its total reward is 0. By contrast, if we require simultaneous expansion of all remaining nonterminals in each round, then there is a single "stackless and memoryless" strategy that gains infinite reward, namely: in each round expand every copy of X using $X \overset{(\perp,0)}{\mapsto} XY$, and (simultaneously) expand every copy of Y using its unique rule. Clearly, after $n \geqslant 1$ rounds we accumulate reward $7 * (n-1)$ by doing this. Thus the total re-

ward will be ∞. Similarly, consider the simple grammar $\{X \overset{(\perp,0)}{\mapsto} XY\,;\, Y \overset{(\perp,1)}{\mapsto} Y\}$, where, again, both nonterminals X, Y are controlled by the maximizing player, and X is the start nonterminal. Under the left-most derivation law, clearly the maximum reward is 0, whereas under the right-most or simultaneous derivation law, the total reward is ∞. So, the supremum total (expected) reward is not robust and can wildly differ, depending on the derivation law. On the other hand, if all the rewards on grammar rules are positive then it can be shown that the total reward is robust and does not depend on whether the left-most, right-most or simultaneous derivation law is used.

2.2.3 Multi-Type Branching Processes

Multi-Type Branching Processes (MT-BPs) are an important class of stochastic processes, first studied by Kolmogorov and Sevastyanov beginning in the 1940s ([KS47]). They have many applications, e.g., molecular biology ([KA02]) and nuclear chain reactions ([EU48]). Formally, the set of states of an MT-BP G is a subset of $\mathbb{N}^n$, where n is the number of types in G. The i-th entry of a state vector v represents the number of entities of type i in the state v. Each type i has an associated set of rules of the form $i \overset{p}{\longmapsto} v$, where $v \in \mathbb{N}^n$ is a vector that a single entity of type i can be replaced with and $p \in (0,1]$ is the probability assigned to such a rule. For each type, the probabilities, p, on its associated rules constitute a probability distribution. The Markov Chain with rewards generated by an MT-BP G can be described intuitively as follows: we start with some given vector $v \in \mathbb{N}^n$; at each single step we simultaneously apply to each entity one of the rules, r, associated to its type, with that rule's probability p_r, replacing it with the vector v_r (the rhs of the rule r); we then sum all such vectors v_r and obtain the next state vector v; we terminate once we reach $v = \mathbf{0}$ (the all-zero vector). For more background on this model see, e.g., [Har63]. Each MT-BP can be formulated as an SCFG with a simultaneous derivation law. The only difference between these two models is how the state is represented: as a vector of natural numbers for MT-BPs or as a string for SCFGs. For detailed discussion about MT-BPs and their relationships to other models, see [EY05s].

2.2.4 Probabilistic Pushdown Systems

There are a number of equivalent variations on the definition of (probabilistic) Pushdown Systems. We use a standard definition which is convenient for our analysis. A ***probabilistic Pushdown System*** (pPDS) $\mathcal{P} = (Q_P, \Gamma, \Delta)$ consists of a set of con-

trol states Q_P, a stack alphabet Γ, and a probabilistic transition relation $\Delta \subseteq (Q_P \times \Gamma) \times (0,1] \times Q_P \times \{swap(\Gamma), swap\&push(\Gamma \times \Gamma), pop\}$. That is, a transition has the form $((s,\gamma),p_{(s,\gamma),(s',\mathcal{C})},(s',\mathcal{C}))$, where based on the control state s and the symbol on top of the stack, γ, with probability $p_{(s,\gamma),(s',\mathcal{C})}$, the transition updates the control state to s', and performs action $\mathcal{C}$ on the stack. Specifically, if $\mathcal{C} = swap(\gamma')$ then the action swaps the top-of-stack symbol, γ, with symbol γ'. If $\mathcal{C} = swap\&push(\gamma',\gamma'')$, then the action both swaps γ with γ' and then pushes γ'' on top of the stack.[6] Lastly, if $\mathcal{C} = pop$, then the action pops the top-stack-symbol γ off the stack. Each such transition has an associated probability $p_{(s,\gamma),(s',\mathcal{C})}$, and we assume that for each pair of control state and top of stack symbol, (s,γ), we have $\sum_{(s',\mathcal{C})} p_{(s,\gamma),(s',\mathcal{C})} = 1$. We assume there is a special stack symbol $\perp \in \Gamma$ that marks the bottom of the stack. Accordingly, $\perp$ is never overwritten with a different stack symbol, nor popped off the stack, and is never pushed onto the stack or overwrites a different stack symbol. A stack with letter γ at the top and remaining content $\omega \in \Gamma^*$ will be written as $\omega\gamma$ (note that the leftmost symbol in $\omega\gamma$ is $\perp$). A pPDS $\mathcal{P}$ defines a countable-state Markov chain $M(\mathcal{P}) = (V',\Delta')$ in an obvious way. Namely, the states of $M(\mathcal{P})$ are $V' = \{(w,s) \mid s \in Q_P,\ w \in \perp\Gamma^*\}$, and the probabilistic transitions of $M(\mathcal{P})$ are $\Delta' = \{((w,s),p,(w',s')) \mid ((s,\gamma),p,(s',\mathcal{C})) \in \Delta$ & applying action $\mathcal{C}$ to w yields $w'\}$.

A special subclass of pPDSs are probabilistic Basic Process Algebras (pBPAs) which are the same as SCFGs with left-most derivation (without players and rewards).

2.2.5 Random walks with "Back Buttons"

In [FKK+00] a probabilistic model of web surfing with "Back Button" was studied. This model extends a finite Markov chain by allowing not only for normal forward probabilistic transitions, but also for "pressing the back button" with some probability and returning to the previous state from which the current state was entered. Such a model in fact can be easily described by restricting pPDSs: only one control state is allowed, there are *pop* transitions, but there are no *swap* transitions and *swap*&*push* transitions cannot change the top stack symbol, i.e., they are *push* transitions. Such a model can also easily be described as a subclass of SCFGs and 1-RMCs (see [EY05s] for details).

[6] Note that the standard push transition $((s,\gamma),p_{(s,\gamma),(s',push(\gamma'))},(s',push(\gamma')))$ can be trivially encoded as $((s,\gamma),p_{(s,\gamma),(s',swap\&push(\gamma,\gamma'))},(s',swap\&push(\gamma,\gamma')))$.

2.2.6 (Probabilistic) 1-Counter Automata

A ***probabilistic 1-counter automaton*** (p1CA), $\mathcal{A}$, is just a pPDS with only one stack symbol γ (other than the special bottom symbol $\perp$). In other words, it is a pPDS with $\Gamma = \{\perp, \gamma\}$. This is not the usual definition: they are typically defined as having a finite number of control states and an additional non-negative counter which can be incremented or decremented during transitions, and such transitions can be enabled/disabled depending on whether the counter is equal to 0 or not. However, this can easily be seen to be equivalent to a pPDS with one stack symbol, γ. The stack acts precisely as a (unary) counter, and the counter is equal to 0 precisely when the top stack symbol is $\perp$.

Formally, a p1CA is usually defined in the following form, which we will find convenient. A p1CA, $\mathcal{A}$, is 3-tuple $\mathcal{A} = (S, \delta, \delta_0)$ where S is a finite set of ***control states*** and $\delta \subseteq S \times \mathbb{R}_{>0} \times \{-1, 0, 1\} \times S$ and $\delta_0 \subseteq S \times \mathbb{R}_{>0} \times \{0, 1\} \times S$ are ***transition relations***. The transition relation δ is enabled when the counter is nonzero, and the transition relation δ_0 is enabled when it is zero. We use $p_{uv}^{(c)}$ to denote the unique probability such that there is a transition $(u, p_{uv}^{(c)}, c, v) \in \delta$, and likewise we use $q_{u,v}^{(c)}$ to denote the unique probability such that there is a transition $(u, q_{u,v}^{(c)}, c, v) \in \delta_0$. If such a transition exists, it is unique, and thus $p_{uv}^{(c)} > 0$ (or $q_{u,v}^{(c)} > 0$) is uniquely determined. If such a transition doesn't exist, we may sometimes assume for convenience that $p_{uv}^{(c)} = 0$ (or $q_{u,v}^{(c)} = 0$), even though there are no explicit 0-probability transitions provided in the input which describes $\mathcal{A}$. The transition probabilities out of each control state u define a probability distribution, i.e., $\sum_{c=-1}^{1} \sum_v p_{uv}^{(c)} = 1$, and $\sum_{c=0}^{1} \sum_v q_{u,v}^{(c)} = 1$. A p1CA, $\mathcal{A}$, generates a denumerable-state Markov chain $M(\mathcal{A}) = (V', \Delta')$ with state set $V' = \{(s, d) \mid s \in S,\ d \in \mathbb{N}\}$, and probabilistic transition relation $\Delta' = \{((s,0), p, (s', j)) \mid (s, p, j, s') \in \delta_0\} \cup \{((s, i), p, (s', j)) \mid i > 0,\ \&\ (s, p, c, s') \in \delta,\ \&\ j = i + c\}$.

A single controller MDP version of p1CAs is defined in Section 3.5 of Chapter 3 where we show that the qualitative decision problem about their termination probability is DP-hard (both NP-hard and coNP-hard). By contrast, the same question for 1-RMDPs can be solved in PTIME. No algorithm is yet known for answering the decision and even approximation problems for controlled p1CAs.

Now we define a non-probabilistic non-controlled version of p1CAs, 1-counter automata (1CAs). They will be useful when reasoning about the structure of the transition system of p1CAs, while ignoring the exact probability values. Formally, a ***1-counter automaton*** (1CA) is just a p1CA without probabilities, i.e., the transition relation is non-deterministic. So, a 1CA $\mathcal{A} = (S, \delta, \delta_0)$, has transition relations

$\delta \subseteq S \times \{-1,0,1\} \times S$, and $\delta_0 \subseteq S \times \{0,1\} \times S$. To each p1CA, $\mathcal{A} = (S, \delta, \delta_0)$, we can associate an ***underlying 1CA***, $\mathcal{A}' = (S, \delta', \delta_0')$, which ignores probabilities of transitions and treats them non-deterministically. Specifically, a transition $(u,c,v) \in \delta'$ ($\in \delta_0'$) iff $p_{uv}^{(c)} > 0$, ($q_{u,v}^{(c)} > 0$, respectively). For a 1CA, $\mathcal{A} = (S, \delta, \delta_0)$, a ***path*** starting at state (s_1, n_1) is a sequence of states $(s_1, n_1), (s_2, n_2), ..., (s_r, n_r)$, such that, for all $i \in \{1, \ldots, r-1\}$, either $n_i > 0$ and $(s_i, n_{i+1} - n_i, s_{i+1}) \in \delta$ or $n_i = 0$ and $(s_i, n_{i+1} - n_i, s_{i+1}) \in \delta_0$. It is called a ***nonzero path*** if $n_i > 0$ for all $i \in \{1, \ldots, r-1\}$. (Note that we allow $n_r = 0$ in nonzero paths.) Such a (nonzero) path is called a ***(nonzero) terminating path*** if $n_r = 0$, and if so it is said to terminate in state $(s_r, 0)$. For p1CAs, $\mathcal{A}$, we define paths, nonzero paths, etc., as simply the paths, nonzero paths, etc., in the underlying 1CA. Note that for a p1CA the probability that a particular nonzero path $(s_1, n_1), (s_2, n_2), ..., (s_r, n_r)$ occurs, in a random walk starting at state (s_1, n_1) of the Markov chain $M(\mathcal{A})$, is precisely equal to $\prod_{1 \leqslant i < r} p_{s_i s_{i+1}}^{(n_{i+1} - n_i)}$.

2.2.7 Quasi-Birth-Death Processes (QBDs)

We consider discrete-time QBDs only. Of course, many analyses for continuous-time QBDs can be reduced to analyses of their respective embedded discrete-time chains.

A ***Quasi-Birth-Death process*** (QBD) is a countable state Markov chain whose transition matrix has the following block structure:[7]

$$\begin{bmatrix} B_0 & B_1 & 0 & 0 & 0 & \ldots \\ A_{-1} & A_0 & A_1 & 0 & 0 & \ldots \\ 0 & A_{-1} & A_0 & A_1 & 0 & \ldots \\ 0 & 0 & A_{-1} & A_0 & A_1 & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \end{bmatrix}$$

where $B_0, B_1, A_{-1}, A_0, A_1 \in \mathbb{R}_{\geqslant 0}^{m \times m}$. Thus, the finite input which describes a QBD consists of the five $m \times m$ matrices: B_0, B_1, A_{-1}, A_0, and A_1. We can represent each state of a QBD by a pair (i,j), where $1 \leqslant i \leqslant m$ is the index of the state within its block and $j \in \mathbb{N}$ is the index of the block. Central to many analyses for QBDs is the computation of the associated G matrix, which we will call the ***termination probability matrix***. This is a $m \times m$ matrix, whose (i,i') entry $G_{i,i'}$ denotes the probability that, starting in state $(i,1)$, the Markov chain will eventually visit a state in block 0, and such that the first

[7] In fact, various slightly different definitions of QBDs are given in the literature, typically differing slightly on the structure of transition probabilities in the boundary cases, i.e., for the first few blocks. These differences are immaterial and these variants can be efficiently embedded in the transition structure described here, as many authors have already observed.

such state it visits is $(i',0)$. As is well known (e.g., [Neu81]), G is the least non-negative solution to the matrix equation $X = A_{-1} + A_0X + A_1X^2$, i.e., for any non-negative solution matrix G', we have $G \leqslant G'$ (entry-wise inequality). Other key matrices, which are also central to computations for QBDs, can be derived from the matrix G. Specifically, the R matrix, has $R_{i,i'}$ equal to the expected number of visits to state $(i',n+1)$, starting from state (i,n), before returning to a state in a block $\leqslant n$. The matrix U (the "taboo probability" matrix) has $U_{i,i'}$ equal to the probability that starting from state $(i,1)$ the chain does not visit a state in block 0 until it eventually revisits a state in block 1, and it does so in state $(i',1)$. The matrices U and R can be obtained from G: $U = A_0 + A_1G$, and $R = A_{-1}(I-U)^{-1}$. (Of course, an approximate solution of G will introduce errors in the solutions for U and R.) If the QBD is positive recurrent, these matrices can be used to compute the steady state probabilities for being in any given state (i,j) (see, e.g., [LR99]). Specifically, if for $n \geqslant 0$ we let $\boldsymbol{\pi}_n$ denote the m-vector whose i'th entry is the steady-state probability of being in (i,n), then $\boldsymbol{\pi}_{n+1} = \boldsymbol{\pi}_1 R^n$, for $n \geqslant 1$, and $\boldsymbol{\pi}_0, \boldsymbol{\pi}_1$ are the unique solution to the following system of equations:

$$\boldsymbol{\pi}_0 = \boldsymbol{\pi}_0 B_0 + \boldsymbol{\pi}_1 A_{-1}$$
$$\boldsymbol{\pi}_1 = \boldsymbol{\pi}_1 B_1 + \boldsymbol{\pi}_1 A_0 + \boldsymbol{\pi}_1 R A_{-1}$$

with the normalization condition $\boldsymbol{\pi}_0\mathbf{1} + \boldsymbol{\pi}_1(I-R)^{-1}\mathbf{1} = 1$ (provided that $\sum_{i \geqslant 0} R^i$ converges).

2.2.8 Tree-Like and Tree-Structured QBDs

Tree-Like Quasi-Birth-Death processes (TL-QBDs) and ***Tree-structured Quasi-Birth-Death processes*** (TS-QBDs) are generalizations of QBDs and several slight variants of their formal defintions have appeared in the literature. We use the most restrictive definition of TL-QBDs (as in [HB03]), in order to have the strongest results about the equivalence of all these models. Consider the infinite rooted d-ary tree T_d, label every edge with a symbol in $\Gamma = \{1,\ldots,d\}$, and label every node with the string $w \in \Gamma^*$ corresponding to the path from the root; the root is labeled with the empty string ε. The states of TS-QBDs and TL-QBDs consist of pairs (w,i), where $w \in \Gamma^*$ is (the label of) a node of the tree T_d and $i \in \{1,\ldots,m\}$ acts as a "control state". The transitions of a TS-QBD are as follows. From a state (ε,i), $i \in \{1,\ldots,m\}$, there is a transition to state:

1. (ε,j) with probability $f^{i,j}$, where $j \in \{1,\ldots,m\}$.
2. (s,j) with probability $u_s^{i,j}$, where $s \in \Gamma$, and $j \in \{1,\ldots,m\}$.

From any state (wk,i), where $w \in \Gamma^*$ and $k \in \Gamma$, and $i \in \{1,\ldots,m\}$, there is a transition to state:

3. (w,j) with probability $d_k^{i,j}$.
4. (ws,j), where $s \in \Gamma$, with probability $a_{k,s}^{i,j}$.
5. (wks,j), where $s \in \Gamma$, with probability $u_s^{i,j}$.

A TS-QBD can thus be described by a finite collection of $m \times m$ matrices (specifically, d^2+2d+1 such matrices) with rational entries, namely the matrices D_k, $A_{k,s}$, U_s, and F, where $k,s \in \Gamma$, and where their (i,j) entry is $d_k^{i,j}$, $a_{k,s}^{i,j}$, $u_s^{i,j}$, and $f^{i,j}$, respectively.

TL-QBDs are defined by restricting TS-QBDs: TL-QBDs are TS-QBDs with the additional requirement that if $k \neq s$, then $A_{k,s} = 0$ (i.e., the zero matrix), and secondly that $A_{k,k} = A_{s,s}$ for all $k,s \in \Gamma$. Thus, in a TL-QBD, there are no direct transitions from a state (wk,i) to a state (ws,j), where $k \neq s$, and if there is a direct transition from state (wk,i) to state (wk,j), with probability p, then there is a direct transition from state (ws,i) to (ws,j) with the same probability p. In other words, the probability of transition from control state i to control state j, while not changing the "stack", does not depend on the topmost (rightmost) symbol on the "stack".

2.3 Efficient embeddings and equivalences

We show that various probabilistic models with total accumulated payoff function are "essentially equivalent". To make the notion of "essentially equivalent" precise, we use the following definitions.[8]

Definition 2.3.1. For a (countable-state) Simple Stochastic Game $\mathcal{G}$ with states t and t', we write $t \overset{\bar{t},p,c}{\rightsquigarrow} t'$ for $p \in \{\perp\} \cup \mathbb{R}_{\geqslant 0}$ and a sequence of states $\bar{t} = t_1,\ldots,t_{k-1}$ (can be empty if $k = 1$), to denote that the following transitions exist in $\mathcal{G}$ (denoting $t_0 = t$, $t_k = t'$): (t_0,p,c_0,t_1) and $(t_i,1,c_i,t_{i+1})$ for $1 \leqslant i < k$ and furthermore $\sum_{0 \leqslant i < k} c_i = c$. (Note that if $k = 1$, this just says that (t,p,c,t') is a transition of $\mathcal{G}$.)

We shall say that one (countable state) Simple Stochastic Game $\mathcal{G}$ ***embeds efficiently*** in another Simple Stochastic Game $\mathcal{G}'$, if there exist two polynomial-time computable

[8] All the definitions in this section are intended to capture "an equivalence" between probabilistic controlled/game models with the total undiscounted reward objective. They do not apply to games where the payoff is computed via a discounted or average reward method and it is nontrivial how one could extend it to such settings. On the other hand, as we have already shown, we can encode all the objectives that we discuss in this thesis, e.g., the termination and reachability, and others, just by using the undiscounted total reward objective.

mappings, f, g, where f is a one-to-one mapping from states of $\mathcal{G}$ to states of $\mathcal{G}'$, and g is a one-to-one mapping that maps a transition (t, p, c, t') of $\mathcal{G}$ to a sequence, $\bar{t} = t_1 \dots t_{k-1}$ of states in $\mathcal{G}'$, such that $f(t) \overset{\bar{t},p,c}{\rightsquigarrow} f(t')$ holds in $\mathcal{G}'$, and furthermore such that none of the auxiliary states $t_1, \dots, t_{k-1}$ are in the range of the mapping f.

Intuitively, this is essentially a monomorphic embedding of one Simple Stochastic Game inside another, except that a transition (t, p, c, t') can be "stretched" into a sequence of transitions, using intermediate auxiliary states, and with probability 1 transitions out of these auxiliary states leading to the target, $f(t')$, and the total reward of such a sequence of transitions being equal to the original reward c. Notice that if $\mathcal{G}$ embeds efficiently into $\mathcal{G}'$ (via mappings f and g) and $\mathcal{G}'$ embeds efficiently into $\mathcal{G}''$ (via mappings f' and g') then $\mathcal{G}$ embeds efficiently into $\mathcal{G}''$ (via $f' \circ f$ and slightly modified $\overline{f'} \circ g$; see below for the definition of $\overline{f'}$). In other words, this relation is transitive.

Furthermore, notice that we can extend the function f to a monomorphic mapping of histories of a play (sequences of states) in $\mathcal{G}$ to histories of a play (sequences of states) in $\mathcal{G}'$. For a history of a play $\bar{u} = u_0 u_1 \dots u_k$ we define $\bar{f}(\bar{u}) = f(u_0) g(u_0, u_1) f(u_1) g(u_1, u_2) \dots f(u_k)$, where the value of $g(u, v)$ is the sequence of states the function g maps the unique transition from u to v. Notice also that function f^{-1} is well defined for all non-auxiliary states in $\mathcal{G}'$ and can be extended to a function $\overline{f^{-1}}$ from histories of a play in $\mathcal{G}'$ to histories of a play in $\mathcal{G}$. A history $\bar{u} = u_0 u_1 \dots u_k$ would be mapped to a sequence of states in $\mathcal{G}$ derived from concatenating states $f^{-1}(u_i)$ one after another for which u_i is not an auxiliary state. It is not hard to see that any strategy σ in Simple Stochastic Game $\mathcal{G}$ has a corresponding strategy for the same player in $\mathcal{G}'$ defined by $\sigma' = f \circ \sigma \circ \overline{f^{-1}}$. Moreover, if the strategy σ was optimal in $\mathcal{G}$ then it is also optimal in $\mathcal{G}'$. It is because, when starting with any initial history $\bar{u}$ in $(\mathcal{G}')^{\sigma,\tau}$, the total expected reward is the same as when starting with an initial history $\overline{f^{-1}}(\bar{u})$ in $\mathcal{G}^{\sigma',\tau'}$. It follows that if a Simple Stochastic Game $\mathcal{G}$ embeds efficiently into a Simple Stochastic Game $\mathcal{G}'$, then the game $\mathcal{G}$ that starts at u has a value iff $\mathcal{G}'$ that starts at $f(u)$ has a value; furthermore, these values are equal if they exist.

In Section 2.2, we defined various finite descriptions of potentially infinite (but countable-state) Simple Stochastic Games (or Markov Chains). For a family $\mathcal{F}$ of finite presentations of Simple Stochastic Games, each $\mathcal{A} \in \mathcal{F}$, describes a potentially infinite-state underlying stochastic game $\mathcal{G}(\mathcal{A})$. We now define what it means for different classes of finitely-presented Simple Stochastic Games to be "essentially equivalent" (called M-equivalent).

Definition 2.3.2. If $\mathcal{F}$ and $\mathcal{F}'$ are two classes of finitely-presented Simple Stochastic Games, we say that $\mathcal{F}$ is ***efficiently subsumed*** by $\mathcal{F}'$ iff: there is a polynomial-time computable mapping $h : \mathcal{F} \mapsto \mathcal{F}'$, which maps a model $\mathcal{A} \in \mathcal{F}$ to a $h(\mathcal{A}) \in \mathcal{F}'$, and such that there exists a pair of functions $f_{\mathcal{A}}$ and $g_{\mathcal{A}}$, which can themselves be efficiently computed (as Turing machines) from $\mathcal{A}$, and such that $f_{\mathcal{A}}$ and $g_{\mathcal{A}}$ constitute an efficient embedding of $\mathcal{G}(\mathcal{A})$ into $\mathcal{G}(h(\mathcal{A}))$. Finally, we say that two classes $\mathcal{F}$ and $\mathcal{F}'$ of finitely-presented stochastic games are ***M-equivalent*** if both of them are efficiently subsumed by the other.

It is not hard to see that if one family $\mathcal{F}$ of finitely-presented Simple Stochastic Games is efficiently subsumed by another family $\mathcal{F}'$, via a mapping h, then a variety of computational problems for $\mathcal{G}(A)$, where $A \in \mathcal{F}$, can efficiently be reduced to basically the same analyses for $\mathcal{G}(h(A))$, where $h(A) \in \mathcal{F}'$. Furthermore, since the relation of being efficiently subsumed is transitive, M-equivalence is also transitive (and obviously symmetric), hence it is an equivalence relation.

So far we formally defined an equivalence relation between classes of finitely-presented Simple Stochastic Games, models that are controlled and involve rewards. However, in Section 2.2 we defined many classes that are purely probabilistic with no rewards assigned to transitions (e.g., QBDs and p1CAs). The underlying model of these classes are countable-state Markov Chains as oppose to Simple Stochastic Games. The definition of being "efficiently embedded" can easily be adopted to such a setting. The only change needed is a redefinition of the relation $t \overset{\bar{t},p,c}{\rightsquigarrow} t'$. For a Markov Chain $\mathcal{M} = (V, \Delta)$ we write $t \overset{\bar{t},p}{\rightsquigarrow} t'$, where $t, t' \in V$, $\bar{t} = t_1, \ldots, t_{k-1}$ is a sequence of states, and $p \in (0, 1]$, to denote that the following transitions exist in $\mathcal{M}$ (denoting again $t_0 = t$, $t_k = t'$): $(t_0, p, t_1) \in \Delta$ and $(t_i, 1, t_{i+1}) \in \Delta$ for $1 \leqslant i < k$. The definition of M-equivalence remains the same, but the efficient embedding notion uses the just defined $t \overset{\bar{t},p}{\rightsquigarrow} t'$ relation. If two probabilistic non-controlled models are M-equivalent then the computational problems for one of them can efficiently be reduced to the corresponding problem for the other model. These include both transient analyses (such as reachability or hitting probability) as well as limit distributions[9].

Obviously, pPDSs with only one stack symbol γ (other than $\perp$) and p1CAs are M-equivalent. (Under the insignificant technical assumption that counter values in states of a p1CA are encoded in unary as otherwise the mapping f would not be a

[9] In some cases, an aperiodic irreducible chain may be turn into a periodic one, or vise-versa, by the embedding (because the embedding can convert a transition into a sequence of two or more transitions), but this is a minor issue and the original steady-state distribution, if it exists, can be recovered from the uniquely determined stationary distribution of the embedded chain.

polynomial-time computable function.)

Furthermore, it follows from [EY05s] that RMCs are M-equivalent to pPDSs, while 1-RMCs are M-equivalent to SCFGs with left-most derivation, that in turn are the same as pBPAs (probabilistic Basic Process Algebras). We will now establish some new M-equivalences.[10]

2.3.1 Equivalence of SCFG games and 1-exit RSSGs

We prove that SCFG games (with left-most derivation) and 1-RSSGs are M-equivalent. Furthermore, we show that 1-RSSGs with positive rewards are M-equivalent to SCFG games (with left-most derivation) with positive rewards, i.e., the rewards assigned to grammar rules are all strictly positive.

We will do it in several steps by showing M-equivalence between several slight variations of our models. First, we show that we can safely assume that in our games $\xi(ex) = 0$ for any $ex \in Ex$. This will allow us to ignore the state ✠ in the analysis of RSSGs with positive rewards in Chapter 4 and safely assume that the states (ε, ex) for $ex \in Ex$ are the terminal states instead. Of course 1-RSSGs (with positive rewards) for which $\xi(ex) = 0$ for any $ex \in Ex$ can be efficiently embedded into 1-RSSGs (with positive rewards). Conversely, for 1-RSSG (with positive rewards) A with mapping ξ, for each of its components, X, whose exit, ex, satisfies $\xi(ex) > 0$, we create a new component X' which is almost the exact copy of X apart from its exit ex becoming an inner node of X' and the new exit of X' being denoted by ex'. The "new" node ex is probabilistic and has only one transition, with probability 1 and reward $\xi(ex)$, to the new exit ex'. A new 1-RSSG (with positive rewards) A' is created from A by keeping all its components, boxes, mapping, rewards, transitions intact, but adding the new components X'. The new mapping ξ' for A' is equal to 0 for all $ex \in Ex'$ and for $u \in \mathit{Call}'$ it is equal to $\xi(u)$. It is quite easy to see that A embeds efficiently into A' and for A' we have $\xi'(ex) = 0$ for all $ex \in Ex'$.

The next step in the simplification of 1-RSSGs with positive rewards is dropping the requirement of $\xi(u)$ being positive for all $u \in \mathit{Call}$. Unfortunately in this case we cannot show full M-equivalence, but a slight relaxation of M-equivalence. For a 1-RSSG with positive rewards A for which $\xi(u) = 0$ for some $u \in \mathtt{Call}$ we create a new 1-RSSG with positive rewards A' for which all $\xi'(u) > 0$ and such that $\mathcal{G}(A)$ "almost efficiently embeds" into $\mathcal{G}(A')$. Let $c_{\mathtt{min}}$ be the lowest reward on any transition in A (of course $c_{\mathtt{min}} > 0$). The new 1-RSSG A' is almost the exact copy of A apart from

[10] All of them apart from the first one where already presented in [EWY08q].

the reward on any transition that leads to $u \in \mathit{Call}$, for which $\xi(u) = 0$, decreased by $c_{min}/2$ and the new mapping ξ' being equal to $c_{min}/2$ for all such call ports u. From the definition of c_{min}, it follows that all the transitions in A' have positive rewards, even the rewards assigned by ξ'. The problem of mapping Simple Stochastic Game $\mathcal{G}(A)$ into $\mathcal{G}(A')$ lays in the states corresponding to the call ports. The total reward for such nodes is actually $c_{min}/2$ higher in $\mathcal{G}(A')$ than in $\mathcal{G}(A)$. However, if we relax the definition of efficient embedding, allowing the mapping f not to map probabilistic nodes that have just a single outgoing edge with probability 1 and reward 0, then an identity function for all other states of $\mathcal{G}(A)$ suffices as the mapping function f. It is not hard to see that a probabilistic state u of $\mathcal{G}$, that has a single probability 1 and reward 0 transition connecting it to some other state v, can be safely ignored during the analysis of $\mathcal{G}$, since without any practical change in the behavior of the game $\mathcal{G}$, any node connected to u can be connected directly to v instead. As we can see, the analysis of 1-RSSGs with positive rewards where $\xi(u) = 0$ for some $u \in \mathit{Call}$ can be reduced to the analysis of games with all rewards on call ports being strictly positive and so in Chapter 4 we can safely assume that $\xi(u) > 0$ for all $u \in \mathit{Call}$.

Now, we will simplify SCFG games, by showing that they are M-equivalent to SCFG games with grammar rules restricted to having at most two nonterminals on their right-hand side. Specifically, only three kinds of rules are allowed in such grammars, either of the form (1) $X \overset{(p,c)}{\longmapsto} \varepsilon$, or (2) $X \overset{(p,c)}{\longmapsto} Y$, or (3) $X \overset{(p,c)}{\longmapsto} YZ$. Furthermore, all rules of the form $X \overset{(p,c)}{\longmapsto} YZ$ are the unique rules associated with the nonterminal X, i.e., X is a probabilistic nonterminal, and the unique rule has the form $X \overset{(1,c)}{\longmapsto} YZ$ for some reward c. The transformation is really simple and it is related to the fact that any CFG has an equivalent CFG in Chomsky Normal Form. The only rules that need to be changed are of the form $X \overset{(p,c)}{\longmapsto} X_1X_2\ldots X_k$ for $k \geqslant 3$. Each such rule is replaced by the following set of rules $X \overset{(p,c/k)}{\longmapsto} Z_k$; $\{Z_i \overset{(1,c/k)}{\mapsto} Z_{i-1}X_i \mid i = 2,\ldots,k\}$, where we define $Z_1 \equiv X_1$, and for $i \in \{2,\ldots,k\}$, Z_i-s are new nonterminals added to the grammar. Notice that if c was positive then c/k stays positive and so any SCFG game with positive rewards is translated into a SCFG game with positive rewards and such that each grammar rule has at most two nonterminals on its right-hand side.

We can finally show an M-equivalence between 1-RSSGs (with positive rewards) and SCFG games (with positive rewards). Our reduction is a specialized version of the reduction given in [EY05s] to just one control state models, but at the same time extended to the "games with rewards" setting. The M-equivalence will be shown for 1-RSSGs with $\xi(ex) = 0$ for all $ex \in Ex$ and SCFG games with grammar rules restricted

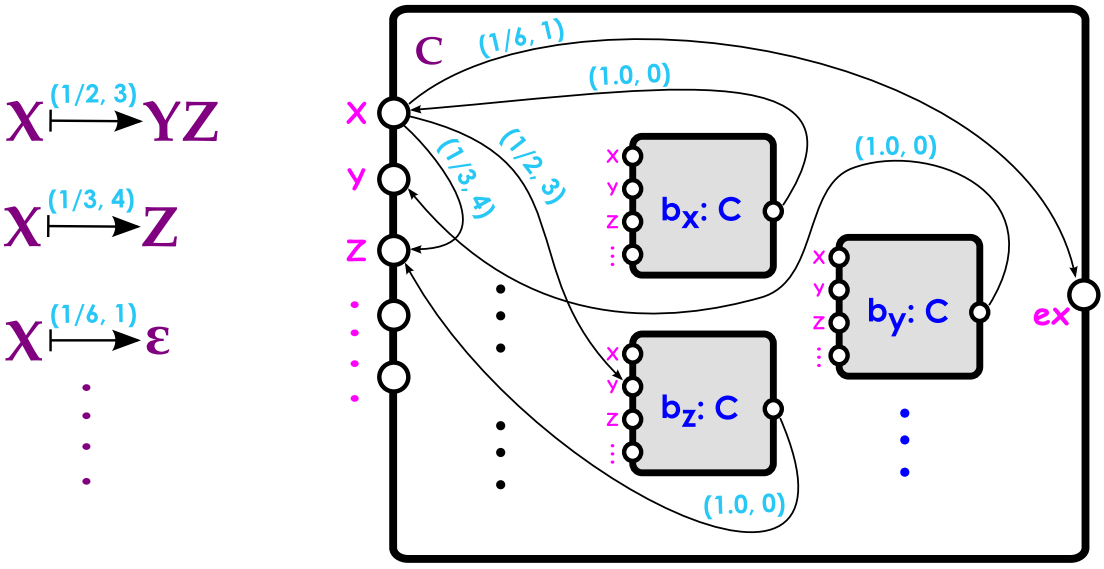

Figure 2.5: An example translation from an SCFG with rewards to a 1-RMC with rewards. The three rules on the left are mapped to the transitions on the right in the 1-RMCs with rewards. In order to map a positive reward transition of an SCFG to a series of positive reward transitions, only a simple manipulation of this mapping is required.

to up to two nonterminals on the rhs, which both, as we have shown, are M-equivalent to the original models. A simple example of a translation of an SCFG with rewards to a 1-RMC with rewards is presented in Figure 2.5.

First we show how to efficiently embed a grammar $G = (V = V_0 \cup V_1 \cup V_2, R)$ (with positive rewards) into a 1-RSSG A (with positive rewards) with $\xi(u) > 0$ for all $u \in \mathit{Call} \cup \mathrm{Ex}$. Let c_{min} be the minimum reward assigned to any grammar rule of G. The 1-RSSG A has a single component named C, with a single exit ex, and multiple entries: en_X for each nonterminal $X \in V$. Component C also contains boxes b_X for each nonterminal $X \in V$ (which of course are mapped to C). For each grammar rule of type (1) $X \overset{(p,c)}{\longmapsto} \varepsilon$ we add $(en_X, p, c - c_{min}/2, ex)$ to the transition relation δ of A, (2) $X \overset{(p,c)}{\longmapsto} Y$ we add (en_X, p, c, en_Y), and (3) $X \overset{(p,c)}{\longmapsto} YZ$ we add $(en_X, p, c - c_{min}/2, (b_Z, en_Y))$. Furthermore, we add the following transitions to δ: $((b_X, ex), 1.0, c_{min}/2, en_X)$ for each $X \in V$ and we set $\xi(u) = c_{min}/2$ for all $u \in \mathit{Call} \cup \mathrm{Ex}$. The mapping pl of vertices of A is defined as follows: if $X \in V_i$ then $\mathrm{pl}(en_X) = i$ and for all other vertices u of A we have $\mathrm{pl}(u) = 0$. The mapping f between states of $\mathcal{G}(G)$ and $\mathcal{G}(A)$ is defined as follows: $f(X\alpha) = \langle \beta, en_X \rangle$, where $\alpha = X_1 X_2 \ldots X_k$ is a sequence of nonterminals and $\beta \in B^*$ is equal to $b_{X_k} \ldots b_{X_2} b_{X_1}$. We are now done since we know that any 1-RSSG A with

$\xi(u) > 0$ for $u \in \mathit{Call} \cup \mathit{Ex}$ can be efficiently embedded into a 1-RSSG A' with $\xi'(u) = 0$ for all $u \in \mathit{Ex}$.

The other way around, for a 1-RSSG $A = (N, B, Y, En, Ex, pl, \delta, \xi)$ (with positive rewards) with the set of vertices $Q = N \cup \mathit{Call} \cup \mathit{Ret}$ we define the following SCFG game $G = (V, R)$ (with positive rewards):

- for each $u \in Q \setminus \mathit{Ex}$ we add a nonterminal X_u to V
- for a call port $u = (b, en)$ of box b (whose only exit is (b, ex)) we add the following rule to R: $X_u \overset{p,\xi(u)}{\longmapsto} X_{en} X_{(b,ex)}$.
- for all other nodes u and each of their transitions $(u, p, c, v) \in \delta$: if v is an exit of a component we add a rule $X_u \overset{p,c}{\longmapsto} \varepsilon$ and otherwise we add $X_u \overset{p,c}{\longmapsto} X_v$

We split nonterminals X_u into sets V_0, V_1, V_2 as follows: $X_u \in V_{pl(u)}$. The mapping f between states of $\mathcal{G}(A)$ and $\mathcal{G}(G)$ can be defined as follows: $f(\langle \beta, u \rangle) = X_u \alpha$, where $\beta = b_1 b_2 \dots b_k$ is a sequence of boxes of A and $\alpha = X_{(b_k,ex)} \dots X_{(b_2,ex)} X_{(b_1,ex)}$.

It can easily be seen that SCFG games (with positive rewards) with right-most derivation are M-equivalent to SCFG games (with positive rewards) with left-most derivation (just reverse the order of all the sequences of nonterminals) and so they are also M-equivalent to 1-RSSGs (with positive rewards).

2.3.2 Equivalence of QBDs and p1CAs

Recall that p1CAs can be defined as pPDSs with just one stack symbol.

Proposition 2.3.3. *QBDs and p1CAs are M-equivalent. Specifically:*

1. *For every QBD $\mathcal{Q}$, there is an easily (linear time) computable pPDS $\mathcal{P}$, with only one stack symbol, such that $\mathcal{Q}$ efficiently embeds in $M(\mathcal{P})$. Moreover, $|P| = O(|\mathcal{Q}|)$, where the size of $\mathcal{Q}$ is measured in terms of the size of the input matrices $B_{i,j}$, $i, j \in \{0, 1\}$ and A_b, $b \in \{-1, 0, 1\}$.*
2. *For every pPDS $\mathcal{P}$ with one stack symbol we can compute (in linear time) matrices $B_{i,j}$, $i, j \in \{0, 1\}$ and A_b, $b \in \{-1, 0, 1\}$, yielding a QBD, $\mathcal{Q}$, such that $M(P)$ efficiently embeds in $\mathcal{Q}$. Moreover, $|\mathcal{Q}| = O(|P|)$.*

Proof.
1. Given a QBD, $\mathcal{A}$, with underlying $k \times k$ matrices $B_0, B_1, A_{-1}, A_0, A_1$, the states of the corresponding PDS, $h(\mathcal{A})$, shall have the structure $\mathcal{P} = (Q_P, \Gamma, \Delta)$, where $\Gamma = \{\bot, \gamma\}$, and

$Q_P = \{1, \ldots k\}$. The transition relation Δ is defined to contain precisely the following transitions for $1 \leqslant i, j \leqslant k$:

- $((i, \perp), (B_0)_{i,j}, (j, \mathit{swap}(\perp))) \in \Delta$.
- $((i, \perp), (B_1)_{i,j}, (j, \mathit{swap}\&\mathit{push}(\perp, \gamma))) \in \Delta$.
- $((i, \gamma), (A_{-1})_{i,j}, (j, \mathit{pop})) \in \Delta$.
- $((i, \gamma), (A_0)_{i,j}, (j, \mathit{swap}(\gamma))) \in \Delta$.
- $((i, \gamma), (A_1)_{i,j}, (j, \mathit{swap}\&\mathit{push}(\gamma, \gamma))) \in \Delta$.

Clearly, $\mathcal{P}$ defines a pPDS with the property that it has one stack symbol γ other than $\perp$, and the stack is always of the form $\perp\gamma^r$, for some $r \geqslant 0$. It is not hard to see that this translation yields an efficient embedding.
2. Any pPDS with only one stack symbol can be viewed as a QBD. Indeed, this is fairly easy to see. Given such a pPDS, the *swap* transitions out of pairs of the form $(q, \perp)$, where, recall, we must swap $(q, \perp)$ with $(q', \perp)$ in order to maintain $\perp$ at the bottom of the stack, can be viewed as giving the matrix B_0, and any $\mathit{swap}\&\mathit{push}(\perp, \gamma)$ transitions out of $(q, \perp)$ can be viewed as giving the matrix B_1. Furthermore, for the transitions out of pairs of the form (q, γ), we can view the *pop*, $\mathit{swap}(\gamma)$ and $\mathit{swap}\&\mathit{push}(\gamma, \gamma)$ transitions as giving the matrices A_{-1}, A_0, and A_1, respectively. □

2.3.3 Equivalence of TL-QBDs, pPDSs and RMCs

Obviously TL-QBDs are a special case of TS-QBDs. Furthermore, TS-QBDs are themselves a special case of pPDSs (equivalently, RMCs [EY05s]), where transitions are constrained as follows:

- For every transition of the form $((s, \gamma), p_{(s,\gamma),(s',\mathcal{C})}, (s', \mathcal{C})) \in \Delta$, where $\mathcal{C} = \mathit{swap}\&\mathit{push}(\gamma', \gamma'')$, we must have $\gamma = \gamma'$. In other words, every "swap and push" operation must be just a "push" operation.
- Furthermore, for all $\gamma, \gamma' \in \Gamma$, we must have:

$$p_{(s,\gamma),(s',\mathit{swap}\&\mathit{push}(\gamma,\gamma''))} = p_{(s,\gamma'),(s',\mathit{swap}\&\mathit{push}(\gamma',\gamma''))}$$

  In other words, the probability of the "push" does not depend on the top stack symbol.

It should be clear that pPDSs with the above restriction are isomorphic to TS-QBDs, under the mapping that maps a state (w,i) of a TS-QBD to the state $(\bot w,i)$ of the corresponding pPDS. We shall show that all pPDSs can be efficiently embedded in TL-QBDs, and thus that all three models are M-equivalent.

Theorem 2.3.4. *pPDSs, RMCs, TL-QBDs, and TS-QBDs are all M-equivalent. Specifically:*

1. *Every TL-QBD as well as every TS-QBD, $\mathcal{Q}$, is a (special form of) pPDS.*

2. *For every pPDS $\mathcal{P}$ we can compute (in quadratic time) a TL-QBD (and thus a TS-QBD), $\mathcal{A}$, such that $M(\mathcal{P})$ efficiently embeds in $\mathcal{A}$, Moreover, $|\mathcal{A}| = O(|\mathcal{P}|)$.*

Proof. It is easy to see from the definitions that pPDSs are the most general model and TL-QBDs the least general. To prove all equivalences, we show that the *swap&push* operation of a pPDS can be encoded using a sequence of 3 transitions of a TL-QBD, using new auxiliary states. First, notice that the *pop* operation of a pPDS effectively already exists in TL-QBDs, and the *swap* operation of a pPDS can then also be encoded once we have *swap&push*: we can simply add a new symbol, ζ, to Γ and instead of having a transition from state $(w\gamma,i)$ to state $(w\gamma',j)$ with probability p, we add a transition that changes state $(w\gamma,i)$ to $(w\gamma'\zeta,j)$ with probability p, and another transition that changes any state $(w'\zeta,j)$ to (w',j) with probability 1. Note that the two transitions together take us from state $(w\gamma,i)$ to state $(w\gamma',j)$ with probability p. Note that we do have available, in a TL-QBD, the ability to do a "pop" with probability 1, as in the second transition described here, which can depend on the top stack symbol, in this case ζ, and we do not need to change the control state.

Now we describe how to implement *swap&push*. If the original control states of the pPDS are $\{1,\ldots,n\}$, then the new control states of the TL-QBD will be of the form $\{1,\ldots,n\} \times \Gamma^{\leqslant 2} \times \{1,2,3\}$. The swap operations of the pPDS shall be mimicked by swap operations (as described above) on control states of the form $(q,\emptyset,1)$. The only place control states of the form $(q,\gamma,2)$ and $(q,\gamma,3)$ shall be used is as follows: a transition of the form: $((q,\gamma),p_{(q,\gamma),(q',\mathcal{C})},(q',\mathcal{C}))$ of the pPDS, where $\mathcal{C} = \textit{swap\&push}(\gamma',\gamma'')$, shall be mimicked by using the following three transitions of the TL-QBD:

Starting at state $(w\gamma,(q,\emptyset,1))$ of the TL-QBD, there is a transition with probability $p_{(q,\gamma),(q',\mathcal{C})}$ $(= d_{\gamma}^{(q,\emptyset,1),(q',\gamma'\gamma'',2)})$ to state $(w,(q',\gamma'\gamma'',2))$, followed by a probability 1 $(= u_{\gamma'}^{(q',\gamma'\gamma'',2),(q',\gamma'',3)})$ transition from state $(w,(q',\gamma'\gamma'',2))$ to state $(w\gamma',(q',\gamma'',3))$, and then finally a probability 1 $(= u_{\gamma''}^{((q',\gamma'',3),(q',\emptyset,1)})$ transition from $(w\gamma',(q',\gamma'',3))$ to $(w\gamma'\gamma'',(q',\emptyset,1))$.

The given transformation constitutes an efficient embedding of the Markov chain $M(\mathcal{P})$, for the given pPDS, $\mathcal{P}$, into the Markov chain $M(\mathcal{A}_{\mathcal{P}})$ for a corresponding TL-QBD, $\mathcal{A}_{\mathcal{P}}$. In particular, the number of control states of $\mathcal{A}_{\mathcal{P}}$ is at most $3|Q_{\mathcal{P}}| \cdot |\Gamma_{\mathcal{P}}|^2$, and the size of the stack alphabet for $\mathcal{A}_{\mathcal{P}}$ is the same as that of $\mathcal{P}$. This mapping thus defines an efficient embedding, and establishes the equivalence. □

2.4 Conclusions

We presented several classes of finitely-presentable infinite state probabilistic models that are used in probability theory and computer science, and have many applications beyond that. We defined what it means for two such formalisms to be essentially equivalent and established some new connections between QBDs and p1CAs; TL-QBDs, TS-QBDs and pPDSs; 1-RSSGs and SCFG games. These equivalences allows the results developed for one model to become immediately applicable to the other. This fact is most evident for QBDs for which many new complexity results are presented in Section 3.2 of Chapter 3 as a corollary of results developed for RMCs and pPDSs. Furthermore, establishing the relationship between the expressive power of these models clarifies their landscape and allows us to represented them as a simple diagram in Figure 2.6. Of course the established equivalences do not mean that we should immediately replace all these model by just one representative of their equivalence class. Different domains of computer science and probability theory find different models more natural and easier to apply to the problems specific to such a domain.

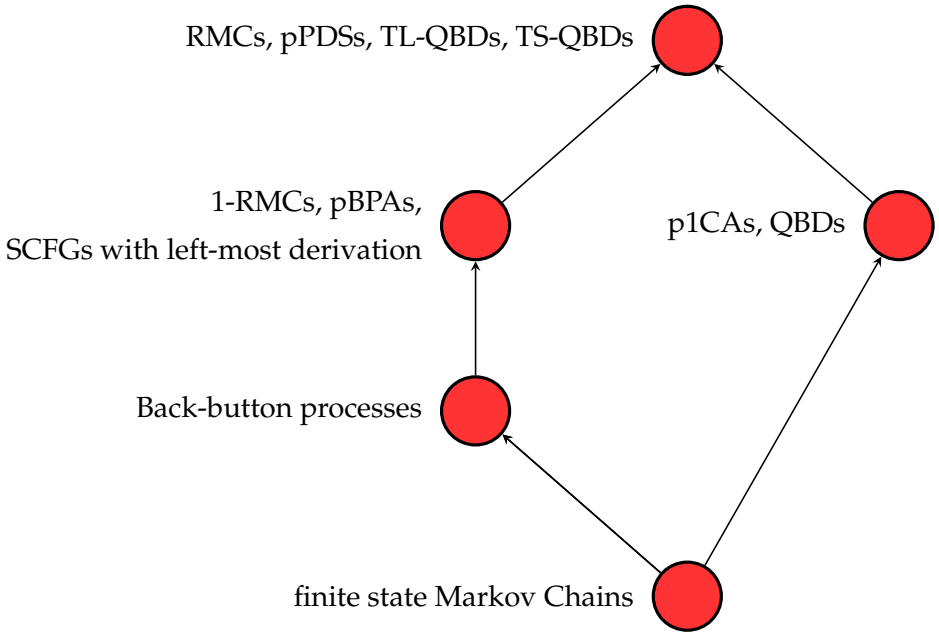

Figure 2.6: Diagram of relative expressive power of all the models discussed in this chapter. The least expressive model is at the bottom while the most expressive ones are at the top. Notice that 1-RMCs and QBDs are incomparable. The fact that QBDs do not subsume 1-RMCs follows from the result in Corollary 3.3.17 and the example in Figure 3.5. On the other hand, the fact that 1-RMCs do not subsume QBDs follows (indirectly) from Theorem 3.5.1 since the problem stated there for 1-RMDPs is in PTIME (see [EY06s]). Notice that MT-BPs are not essentially equivalent to 1-RMCs (since intuitively the transitions are taken in parallel not sequentially), but many computational problems for them can be reduced to the same problems for 1-RMCs.

Chapter 3

Computational complexity of QBDs and their extensions

3.1 Introduction

Quasi-Birth-Death Processes (QBDs) have been studied for decades in queueing theory, performance evaluation, and related areas, both in discrete and continuous time. Tree-Structured QBDs (TS-QBDs) are a generalization of QBDs, and Tree-Like QBDs (TL-QBDs) are a restriction of TS-QBDs. As it was shown in Chapter 2 there is a close correspondence between discrete-time QBDs and probabilistic 1-Counter Automata (p1CAs), and also between TS-QBDs, TL-QBDs, Recursive Markov Chains (RMCs) and probabilistic Pushdown Systems (pPDSs). In this chapter, we exploit these equivalences to obtain several new algorithmic results about these models. A number of results follow immediately from these equivalences and existing results about these various models. (All these models are formally defined in Section 2.2 of Chapter 2.)

The fundamental quantities associated with all such stochastic models are the termination probabilities, called in the QBD setting "the G matrix" which consists of the probabilities of reaching one control state from another while for the first time moving down one level from the original starting level. (This definition will be made formal later.) It is well known that the G matrix of a given QBD is the least nonnegative solution to the following quadratic matrix equation system: $X = A_{-1} + A_0 X + A_1 X^2$, where A_i is the matrix containing the probabilities of transitions of that QBD that modify the counter by i. Many numerical solution methods were developed for computing the G matrix (see, e.g., the books [Neu81, Neu89, LR99, BLM05]) and the quantities that can be derived from it. The analysis of these methods followed the classical numerical

analysis approach by establishing under what circumstances a given method "converges linearly/quadratically" (for their formal definition see, e.g., [OR70]). In particular, Latouche in [Lat94], studied the behavior of Newton's method on strongly connected QBDs, and showed (building on [OR70]) that under certain assumptions that method converges monotonically and "quadratically" to the G matrix. In [BLM03] the performance of Newton's method on strongly connected TL-QBDs was studied and building on [Lat94] it was shown that under a similar set of assumptions, Newton's method again converges monotonically and "quadratically" to an analog of the G matrix in the TL-QBD setting. Several other methods were shown to be "quadratically convergent", e.g., ***logarithmic reduction*** ([LR93]) and ***cyclic reduction*** ([BLM05]). However, it has to be noted that these kinds of analyses do not directly shed light on the complexity of computing the G matrix as a function of the size of the model. In order to see this, let us recall the definition of "linear convergence": A sequence $\{c_m\}_{m=1,2,\ldots}$ (of numbers, vectors or matrices) is said to converge to c^* "linearly" if there exists $f, g > 0$, such that

$$\left\| c_{(f+g\cdot i)} - c^* \right\| \leqslant 1/2^i$$

where $\|\cdot\|$ is some suitable (absolute value, vector, matrix) norm. In other words after f initial iterations we gain one bit of precision every g iterations. Notice that this analysis completely ignores the size of the model, focusing instead on the number of bits i of precision we would like to achieve. However, it is hard to imagine that in practice we need more than, e.g., one hundred bits of precision. (Moreover, if we are using the standard floating point representation of real numbers, the number of bits we can in general even represent using them is smaller than that.) Hence, a proper analysis should really focus on the quantity that really varies which is the size of the model. Let us then denote the size of a model by n, i.e., the number of bits necessary to encode this model in some efficient representation. Now of course f and g will depend on n, so in fact they are functions that map the size of the model n to the maximum value of, respectively, f and g for models of size n. The number of iterations required to obtain i bits of precisions becomes now an expression of the form: $f(n) + g(n) \cdot i$. In order to see what can be misleading in "linear convergence", suppose that $f(n)$ and $g(n)$ turns out to be triply-exponential functions in n. Then although the expression $f(n) + g(n) \cdot i$ is still technically linear in i, for any non-trivial models we would get a bound that is completely useless from any practical point of view. In fact, it might be the case that before we reach the point where the convergence becomes linear, i.e., we get one bit of precision every $g(n)$ iterations, we would need to compute

more than enough bits of the actual solution already. Henceforth, we be focusing on estimating the functions f and g in terms of the size of the model n, and show that $f(n)$ is polynomial and $g(n) = 1$ for the sequence of approximation we get when Newton's method is applied to the system of equations of an arbitrary QBD (even null-recurrent one). In fact, we observe using recent results for pPDSs ([KLE07]) that this polynomial upper bound for QBDs fails badly for TL-QBDs, even though Newton's method still "converges linearly" on these examples. This shows a vast difference between our analysis and the convergence rate analysis traditional to numerical analysis.

We in fact show that a polynomial bound on $f(n)$ holds even for non-strongly connected QBDs when the decomposed Newton's method is used. The decomposed Newton's method was studied for arbitrary monotone systems of polynomial equations in [EY05s] and it was shown there to monotonically converge to their least non-negative solution when starting from initial all-zero values after the equation system was decomposed into strongly connected components (SCCs). (For details on how the decomposed Newton's method works, see Section 3.4). Importantly for our results in this chapter, in [KLE07, EKL08] the performance of that method was studied in greater detail and not only a worst-case linear convergence was established, but in the case of a strongly connected system of equations a constructive upper bound was provided on the number of iterations required by the Newton's method as a function of the encoding size of the polynomial system. No constructive upper bound is known so far for general (not-strongly connected) equation systems. More precisely, it was shown there that for strongly connected monotone polynomial equations the decomposed Newton's method needs at most $f(n) = \mathcal{O}(2^n)$ initial steps after which it obtains one bit of precision per $g(n) = 1$ iteration. On the other hand, for arbitrary equation systems no bound on $f(n)$ is known and the best estimate of $g(n)$ is $\mathcal{O}(2^n)$. As we can see, the previous results do not imply any constructive bound at all for non-strongly connected QBDs nor any immediate polynomial bound on $f(n)$ in the strongly connected QBD setting.

However, just by using the results for the strongly connected case and by studying the special structure of the equation systems for QBDs once decomposed into SCCs, we show that polynomially (in the size of the input) many iterations of Newton's method suffices in order to converge to within any constant additive factor of the termination probabilities. More precisely, we show that, given a QBD, its G matrix can be approximated to within i bits of precision in time polynomial in both the encoding size of the QBD and in i, using polynomially many arithmetic operations. (More

formally, in the unit-cost RAM model or discrete Blum-Shub-Smale (BSS) model of computation.) We summarize all these results in Table 3.1.

model	s-c QBDs (p1CAs)	general QBDs	s-c TL-QBDs (RMCs)	general TL-QBDs
lower bounds	???	???	???	$f(n) = \Omega(2^n)$ $g(n) = \Omega(1)$ [KLE07]
upper bounds	$f(n) = \mathcal{O}(n^k)$ $g(n) = 1$ ([this thesis])	$f(n) = \mathcal{O}(n^k)$ $g(n) = 1$ ([this thesis])	$f(n) = \mathcal{O}(2^n)$ $g(n) = 1$ ([KLE07, EKL08])	$f(n) = ?$ $g(n) = \mathcal{O}(2^n)$ ([KLE07, EKL08])

Table 3.1: A summary of the results on performance analysis of the decomposed Newton's method for subclasses of monotone systems of polynomial equations (after all variables with zero value were removed). The constant k is $\leqslant 8$ (a very rough estimate) and s-c stands for "strongly connected". For all places with ???, a trivial lower bound $f(n) = 0, g(n) = 1$ holds.

In order to derive this result we establish an upper bound on the length of the shortest terminating path between two control states in 1-counter automata and the structure of dependencies among variables in the nonlinear equation for the computation of the G matrix. The former allows us to derive immediately a polynomial upper bound on $f(n)$ from the results of [KLE07, EKL08], and the later allows us to use a classic result on how perturbations in the coefficients of linear equation systems affect its solution in order to deal with a possibly nested series of linear SCCs "above" nonlinear ones in the decomposition of the equation system for QBDs into SCCs. See Figure 3.6 for an example of a QBD, the corresponding polynomial equation system and its decomposition into SCCs.

A tool called PReMo described in Chapter 5 implements optimized versions of the decomposed Newton's method and other methods for the analysis of general min-max polynomial equation systems occurring for various stochastic models discussed in this thesis and has been augmented with an input format for QBDs. We have conducted a comparison of PReMo's performance on QBDs with an existing tool for QBDs: SMCSolver [BMSH06]. The result of this comparison can be found in Section 5.4 of Chapter 5.

The rest of this chapter is organized as follows: In Section 3.2 we state the consequences of the equivalence between QBDs and p1CAs, and between Tree-structured and Tree-like QBDs and pPDSs, and show that the SQRT-SUM problem reduces to the decision problem for QBDs. In Section 3.3 we prove important structural properties of p1CAs, and in Section 3.4 we use them to analyze the decomposed Newton method for QBDs and prove a polynomial bound on the number of iterations necessary to ob-

tain an ε-approximation of the G matrix. In Section 3.5 we define a single controller extension of QBDs, Quasi-Birth-Death Markov Decision Processes (QBD-MDPs), and show that even the qualitative decision problems for them are DP-hard (both NP-hard and coNP-hard). We conclude and present some open problems in Section 3.6.

3.2 Lower bounds on decision procedures for QBDs and TL-QBDs

As it was shown in Section 2.3.3 of Chapter 2, TL-QBDs (which are generalization of QBDs) are equivalent in a formal sense to RMCs and pPDSs, thus all results for pPDSs and RMCs apply to TL-QBDs, and vice versa. The following corollary highlights a few results for TL-QBDs (and TS-QBDs) that follow from work on pPDSs and RMCs. Let us recall the definition of the SQRT-SUM problem, which asks: given natural numbers $(d_1, \ldots, d_n) \in \mathbb{N}^n$ and $k \in \mathbb{N}$, decide whether $(\sum_i \sqrt{d_i}) \leqslant k$ holds. Whether this problem can be solved in NP is open since the 1970s (see Section 2.2.1 of Chapter 2 for more details about the complexity of this problem).

Corollary 3.2.1. *1. ([EY05t, YE05]) The quantitative model checking problem for QBDs and TL-QBDs, against a linear-time (ω-regular or Linear Time Logic (LTL)) property, is decidable in PSPACE in the size of the model.*[1]

2. ([EY05s, EY07]) The SQRT-SUM *problem is polynomial time reducible to the problem of approximating the termination probabilities (the analog of the* G *matrix entries) for TL-QBDs, even to within any constant additive factor $c < 1/2$. Furthermore, even deciding whether a termination probability for a TL-QBD is* 1 *is* SQRT-SUM*-hard.*

3. ([KLE07]) There are TL-QBDs for which at least exponentially many iterations of the (decomposed) Newton's method ([EY05s]), applied to the nonlinear equations for termination probabilities are needed as a function of the TL-QBD's encoding size, to even converge to within just one bit of precision of a termination probability.

The following is not a corollary of earlier results.

Theorem 3.2.2. *The* SQRT-SUM *problem is polynomial time reducible to the following problem: given a p1CA (QBD) with control states u and v, and given a rational value p decide whether $G_{u,v} \leqslant p$.*

[1] For background on model checking, ω-regular languages, Linear Time Logic, see, e.g., [Eme90, Tho97, CGP99].

Proof. This proof is very similar to the proof in [EY05s] that 1-exit RMCs are SQRT-SUM-hard.

Given numbers $(d_1,\ldots,d_n)$ and k, we will construct a p1CA as follows. The p1CA has control state u and n other control states, t_i, corresponding to the given numbers, d_i, $i=1,\ldots,n$. It also has one other control state, v. Let $m=\max_i d_i$. Let $c_i=(1/2)(1-(d_i/m^2))$, for $i=1,\ldots,n$. The transitions of the p1CA are as follows, for $i=1,\ldots,n$: $(u,1/n,0,t_i)\in\delta$ $(t_i,1/2,+1,t_i)\in\delta$ and $(t_i,c_i,-1,t_i)\in\delta$ and $(t_i,1/2-c_i,0,v)\in\delta$, also $(v,1,-1,v)\in\delta$.

We claim that $G_{u,v}=(1/(nm))\cdot\sum_{i=1}^{n}\sqrt{d_i}$, and thus that $G_{u,v}\leqslant(k/(nm))$ if and only if $\sum_{i=1}^{n}\sqrt{d_i}\leqslant k$. To see the claim, note that for each i, we have G_{t_i,t_i} is the least non-negative solution to equation $x=(1/2)x^2+c_i$, and thus that $G_{t_i,t_i}=(1-\sqrt{(1-2c_i)})=(1-\sqrt{d_i}/m)$. Next note that the probability of terminating (in any state) starting from each t_i is 1, because it satisfies the equation $x=(1/2)x^2+(1/2)$. Thus, $G_{t_i,t_i}+G_{t_i,v}=1$ and therefore $G_{t_i,v}=\sqrt{d_i}/m$. Thus, $G_{u,v}=\sum_i(1/n)\sqrt{d_i}/m=1/(nm)\sum_i\sqrt{d_i}$. □

3.3 Structural properties of QBDs (p1CAs)

This section develops crucial structural properties of (probabilistic) 1-Counter Automata, used in section 3.4 to establish strong results on the performance of (decomposed) Newton's method for QBDs. Let $mp(s,s')$ ($mp_{n\text{-}z}(s,s')$) denote the length of the shortest (nonzero, respectively) terminating path starting at state $(s,1)$ and terminating at state $(s',0)$. If there is no such (nonzero) terminating path, then by definition $mp(s,s')=\infty$ ($mp_{n\text{-}z}(s,s')=\infty$, respectively). By convention, a path with a single state has length 0. The next lemma shows that in 1CAs whenever a terminating path exists, a "short" (polynomial length) such path also exists.

Lemma 3.3.1. *Suppose $\mathcal{A}=(S,\delta,\delta_0)$ is a 1CA where $|S|=k$. For any pair of control states $s,s'\in S$, either $mp_{n\text{-}z}(s,s')=\infty$ or else $mp_{n\text{-}z}(s,s')\leqslant k^3$. Likewise, either $mp(s,s')=\infty$, or else $mp(s,s')\leqslant k^4$.*

Proof. We first prove the k^3 upper bound for the length of nonzero terminating paths, and we then show why a k^4 upper bound follows for the length of arbitrary terminating paths. Let $(s_1,n_1),(s_2,n_2),(s_3,n_3),....,(s_r,n_r)$ be the shortest nonzero terminating path starting from $(s,1)$ and terminating in $(s',0)$. (In particular, $(s_1,n_1)=(s,1)$, $(s_r,n_r)=(s',0)$.) We can see an example path represented as a simple plot of the value of the counter along this path in Figure 3.1.

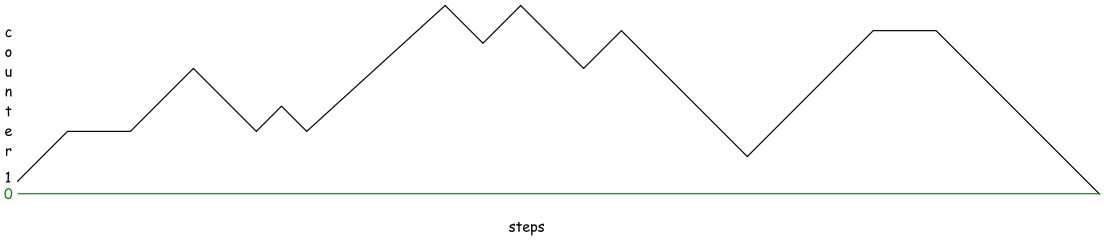

Figure 3.1: A plot of the counter value at each step of the shortest path starting at $(s,1)$ and terminating at $(s',0)$. (The difference in controls states along this path is not represented.)

Let $c_{max} = \max_{i=1}^{r} n_r$ be the maximum value of the counter along this path. There exists some state (s_j, c_{max}) on this path that achieves the highest counter value. (c_{max} may occur more than once, but let's just pick one, say the earliest occurrence, see Figure 3.2 for an example.)

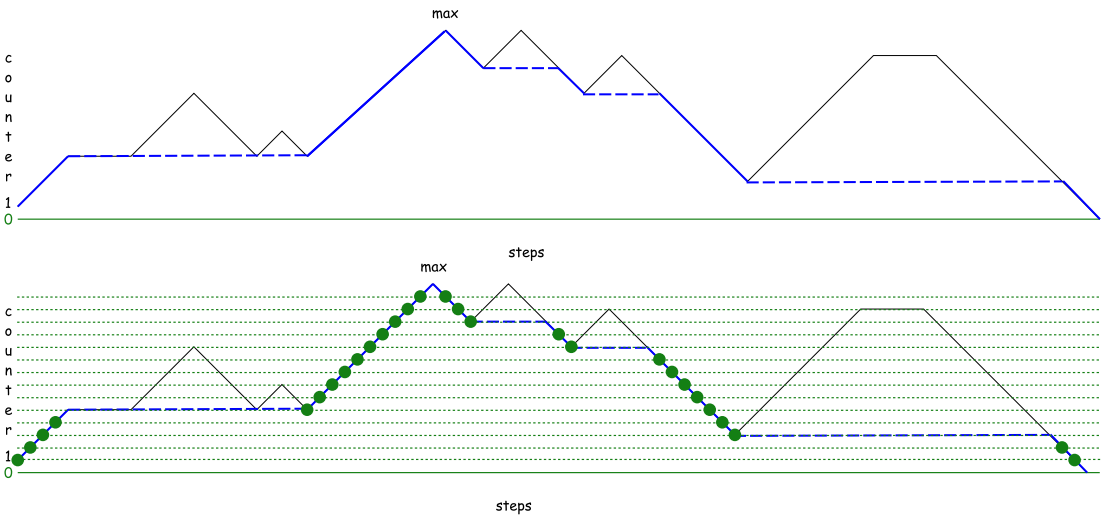

Figure 3.2: The shortest path starting at $(s,1)$ and terminating at $(s',0)$. The left-most state with the maximal value of the counter is marked. The blue line represents the lowest value of the counter encountered when coming from the maximal counter state. Below that, the states that are on the blue line are marked in green and matched into pairs with states with the same counter value on the opposite side of the maximal counter state.

For every counter value $c = 1,\ldots,c_{max}$, we define the pairs (s_{i_c}, c) and $(s_{i'_c}, c)$ as follows: i_c is the largest index $i \leqslant j$ in the path such that the i'th state is (s_i, c), and such that for all $i \leqslant j' \leqslant j$, the j''th state on the path is $(s_{j'}, c')$ where $c' \geqslant c$. (In other words, in the segment from (s_i, c) to (s_j, c_{max}) the count doesn't go below c.) Likewise i'_c is

the smallest index $i \geqslant j$ such that (s_i, c) is on the path and such that on the subpath from (s_j, c_{max}) to (s_i, c) the counter doesn't go below c. Note that $i_{c_{max}} = i'_{c_{max}} = j$. We illustrate it in Figure 3.2 on the same example as before. Clearly such pairs of indices i_c and i'_c are uniquely defined for each $c = 1, \ldots, c_{max}$, and we have $i_1 < i_2 < \ldots < i_{c_{max}} = i'_{c_{max}} < \ldots < i'_2 < i'_1$.

Now the key observation: if $c_{max} > k^2$ then by the pigeon-hole principle there must exist a pair of control states s^a and s^b such that for two distinct values $1 \leqslant c' < c'' \leqslant c_{max}$ of the counter, we have $s^a = s_{i_{c'}} = s_{i_{c''}}$ and $s^b = s_{i'_{c'}} = s_{i'_{c''}}$. Therefore, since we must have $i_{c'} < i_{c''} \leqslant i'_{c''} < i'_{c'}$, we can remove the following two, positive length, segments from the above shortest path and still get a valid nonzero terminating path from $(s_1, 1)$ to $(s_r, 0)$, which would be a contradiction. Namely, we can remove segments: $(s_{i_{c'}}, n_{i_{c'}}) \ldots (s_{i_{c''}-1}, n_{i_{c''}-1})$ and $(s_{i'_{c''}+1}, n_{i'_{c''}+1}) \ldots (s_{i'_{c'}}, n_{i'_{c'}})$. In Figure 3.3 we can see an example of such a cut operation. The resulting path is guaranteed by its construction to be a shorter nonzero terminating path, starting at $(s, 1)$ and terminating at $(s', 0)$, contradicting the fact that the original path was the shortest such path. Therefore, by contradiction, it must be the case that $c_{max} \leqslant k^2$.

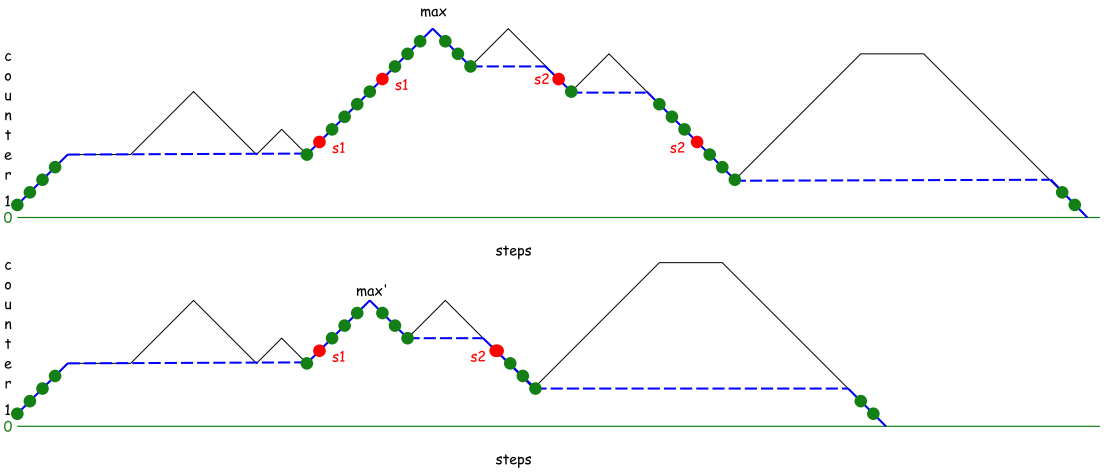

Figure 3.3: The shortest path starting at $(s, 1)$ and terminating at $(s', 0)$ with a repeating pair of control states for two different counter values marked. Below a shorter path starting at $(s, 1)$ and terminating in $(s', 0)$ resulting from a removal of the parts of the path in between the matching control states on the left and on the right.

Therefore, the path $(s_1, 1) \ldots (s_{r-1}, n_{r-1})$ can contain at most $k(k^2) = k^3$ distinct states (not counting repetitions). However, note that in fact no state can repeat along this shortest nonzero terminating because otherwise it would not be the shortest nonzero

terminating path. Therefore the length of the shortest nonzero terminating path from $(s,1)$ to $(s',0)$ is $\mathrm{mp}_{\text{n-z}}(s,s') \leqslant k^3$.

Next we show why it follows that unless $\mathrm{mp}(s,s') = \infty$, then $\mathrm{mp}(s,s') \leqslant k^4$. Consider a shortest terminating path $\pi = (s,1)\ldots(s',0)$, which may include intermediate states with 0 counter values. Note that such a shortest path can only hit the counter value 0 at most k times, because otherwise a 0-counter state would be repeated, and this would then not constitute a shortest path. By the established k^3 upper bound on the length of shortest nonzero terminating paths, we know that the subpath between every pair of 0-counter states in the shortest path π can have at most length k^3. Since there are at most k 0-counter states along the path, the total length of the path is $|\pi| \leqslant k^4$. □

Let us now show two examples for which such a shortest terminating path between two control states has length $\Theta(k^2)$:

Example 3.3.2. Let us consider 1CA $A = (S,\delta,\delta_0)$, where $S = \{s_1,s_2,\ldots,s_{2k}\}$, we have $(s_{2k},-1,s_{k+1}) \in \delta$, and for $i \leqslant k$ we have $(s_i,1,s_{i+1}) \in \delta$, and for $k+1 \leqslant i \leqslant 2k-1$ we have $(s_i,0,s_{i+1}) \in \delta$. (Transitions in δ_0 are irrelevant to our analysis.) The shortest path from $(s_1,1)$ terminating at $(s_{k+1},0)$ has length k^2+k. The length of this path in relation to the number of control states k' (equal to $2k$) is $\frac{1}{4}k'^2 + \mathcal{O}(k')$.

Example 3.3.3. Let us consider 1CA $A = (S,\delta,\delta_0)$, where $S = \{s_1,s_2,\ldots,s_k,s'_1,s'_2,\ldots,s'_{k+1}\}$, $(s_k,1,s_1) \in \delta$, $(s_k,0,s'_1) \in \delta$, $(s'_{k+1},-1,s'_1) \in \delta$, and for $i \leqslant k-1$ we have $(s_i,1,s_{i+1}) \in \delta$, and for $i \leqslant k$ we have $(s'_i,-1,s'_{i+1}) \in \delta$. In other words:

$$\begin{aligned}\delta = \{&(s_1,1,s_2),(s_2,1,s_3),\ldots,(s_k,1,s_1),(s_k,0,s'_1),\\ &(s'_1,-1,s'_2),(s'_2,-1,s'_3),\ldots,(s'_{k+1},-1,s'_1)\}\end{aligned}$$

We would like to find the shortest path between $(s_1,1)$ and $(s'_1,0)$. Notice that each such path visits only control states s_i-s until it reaches for the first time s'_1 from s_k and from that point onwards it visits only control states s'_i-s. For a particular path from $(s_1,1)$ to $(s'_1,0)$, let x denote the number of occurrences of states with control state s_1 and by y the number of occurrences of states with s'_1. In order for a path to terminate at s'_1, we have to have $1+x\cdot k-1 = y\cdot(k+1)$, since otherwise the path would not finish with a counter value 0. (Moreover, the left hand side of this equation is the value of the counter where the transition from s_k to s'_1 takes place.) Since k and $k+1$ are relatively prime for any $k \geqslant 1$, the smallest solution to this equation is $x = k+1$ and $y = k$. This means that the shortest path between $(s_1,1)$ and $(s'_1,0)$ has length $2k(k+1)$ which in terms of the number of states k' (equal to $2k+1$) is $\frac{1}{2}k'^2 - o(1)$. Moreover, the highest

value of the counter along that path is equal to $k(k+1) = \Theta(k'^2)$, which shows that the analysis in Lemma 3.3.1 of the highest possible value of a counter along any shortest terminating path is tight (up to a multiplicative constant).

We conjecture that the previous example has the longest shortest terminating path in relation to the number of control states.

Conjecture 3.3.4. For any p1CA with k control states the length of the shortest terminating path between any two control states is always lower than $\mathcal{O}(k^2)$.

If this conjecture is correct, the upper bound, that we derive for $f(n)$ for the (decomposed) Newton's method for QBDs, would be improved.

For a p1CA, $A = (S, \delta, \delta_0)$, and a pair of states $s, s' \in S$, let us by $G_{s,s'}$ denote the probability that, starting from state $(s,1)$, a random walk on the chain $M(A)$ will traverse a nonzero path that eventually visits and terminates in state $(s',0)$. Given the equivalence of p1CAs and QBDs, the probabilities $G_{s,s'}$ yield precisely the G matrix associated with the QBD, whose size is $|S| \times |S|$. We now use Lemma 3.3.1 to give a "polynomial size" lower bound on positive termination probabilities $G_{s,s'}$, associated with a p1CA (QBD).

Corollary 3.3.5. *Let $A = (S, \delta, \delta_0)$ be a p1CA where $|S| = k$, and let $p_{min} > 0$ be the smallest positive probability on any transition of A.* [2] *For any pair of states $s, s' \in S$, either $G_{s,s'} = 0$ or $G_{s,s'} \geqslant p_{min}^{k^3}$.*

Proof. Indeed, $G_{s,s'} > 0$ iff there is a nonzero terminating path starting at $(s,1)$ and terminating at $(s',0)$. By Lemma 3.3.1, the length of the shortest such path is $\leqslant k^3$. Therefore its probability is at least $p_{min}^{k^3}$. □

Notice that Corollary 3.3.5 and Lemma 3.3.1 do not hold for multi-exit RMCs as it can be seen from the example in Figure 3.4. The path starting at entry en of component A_{k+1} ($k \geqslant 0$) and terminating at ex_1 has to pass through two copies of component A_k, in each of them terminating at the exit ex_1. This means that the length of the shortest path starting at en in A_k and terminating at ex_1 at least doubles if we increment k by 1. Hence, if we fix some n and form an RMC with components $A_0, \ldots, A_n$ then the shortest path terminating at ex_1 in A_n has length $\Theta(2^n)$ while the size of such an RMC is $\Theta(n)$. Furthermore, we terminate at the exit ex_1 in A_0 with probability $1/2$ and the number of A_0 components we have to go through in order to reach the exit ex_1 in A_k

[2] In other words, we have $(u, p_{min}, c, v) \in \delta$ for some u, v, c, and $p_{min} > 0$, and for any transition $(u', p', c', v') \in \delta$, with $p' > 0$, we have $p_{min} \leqslant p'$.

grows exponentially in k. In fact the probability of termination at ex_1 in A_n is as small as $1/2^{2^n}$.

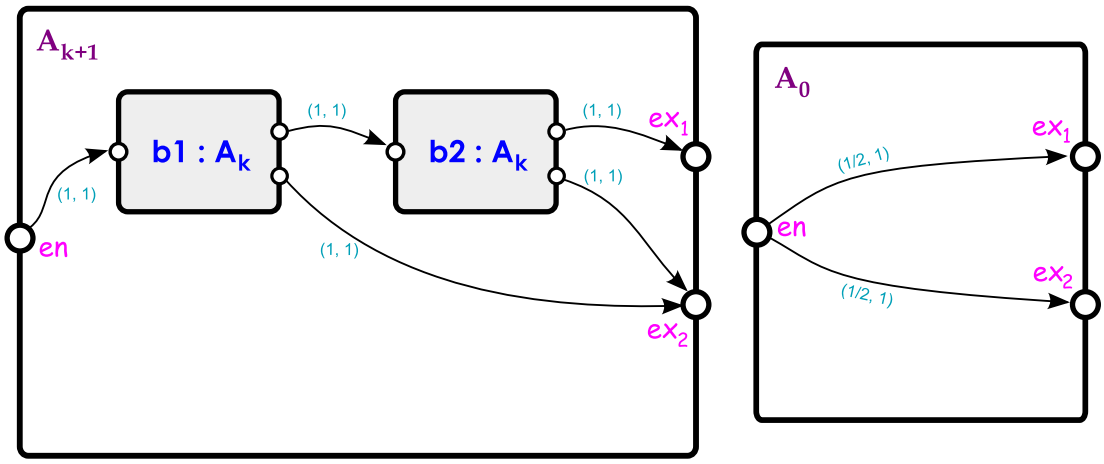

Figure 3.4: An example family of multi-exit RMCs ($k = 0, 1, \ldots, n-1$, for some fixed $n \geqslant 1$) with a very long shortest terminating path at ex_1 in A_n and a very low probability of termination at ex_1. This example is taken from [EY05s].

For a pair of states $u, v \in S$, let x_{uv} be a variable denoting the (unknown) probability, $G_{u,v}$. It is well know (e.g., [Neu81]) that the termination probability matrix G is the least non-negative solution of the matrix equation $X = A_{-1} + A_0X + A_1X^2$ which after expansion can be written down as follows:

$$x_{uv} = p_{uv}^{(-1)} + \Big(\sum_{w \in S} p_{uw}^{(0)} x_{wv}\Big) + \sum_{y \in S} p_{uy}^{(1)} \sum_{z \in S} x_{yz} x_{zv} \tag{3.1}$$

We can clean up this system of equations by removing the variables x_{uv} for which $G_{u,v} = 0$. This can be done in polynomial-time, even for more general fixed point equations associated with pPDSs and RMCs (see [EY05s]). (After clean-up, the equations may no longer have the simple matrix form.) Henceforth, we consider only cleaned-up equation systems, where only nonzero variables remain.

Based on this equation system we can build a dependency graph, $D = (\tilde{X}, E)$, whose nodes are all nonzero variables $\tilde{X} = \{x_{uv} : u, v \in S \text{ and } G_{u,v} \neq 0\}$ and there is an edge $(x_{uv}, x_{st}) \in E$ iff x_{st} occurs on the rhs of the equation $x_{uv} = \alpha$ corresponding to x_{uv}. We decompose this graph into strongly connected components (SCCs) and sort them topologically. As a result we obtain a sequence of SCCs $X_1, X_2, \ldots, X_m$ such that there can exist a path in graph D from variable $x \in X_i$ to variable $x' \in X_j$ only if $i \geqslant j$. We will write $x_{st} \equiv x_{uv}$ iff $s = u$ and $t = v$. We say a variable x_{uv} ***depends on*** the value of a variable x_{st} iff either $x_{st} \equiv x_{uv}$, or there is a path from x_{uv} to x_{st} in the graph D. Of

course this relation is transitive. We say that an equation $x_{uv} = \alpha$ is ***nonlinear in a set*** X' ***of variables*** if, by removing all variables that are not in X' from monomials in α, we are left with an expression α' that is nonlinear. We say that SCC X_i is nonlinear if the equation $x_{uv} = \alpha$ of some variable $x_{uv} \in X_i$ is nonlinear in X_i.

We introduce some additional notation. For a 1CA, $A = (S, \delta, \delta_0)$, we write $u \xrightarrow{+} v$ iff $(u,1,v) \in \delta$; we write $u \to v$ iff $(u,0,v) \in \delta$, and $u \xrightarrow{-} v$ iff $(u,-1,v) \in \delta$. We use the same notation for p1CAs, to denote positive probability transitions, i.e., such transitions existing in the underlying 1CA. For a (p)1CA, and for $k < 0$, we write $s \xrightarrow{k} t$ iff there exists a nonzero terminating path starting at $(s,|k|)$ and terminating at $(t,0)$. For $k \geqslant 0$ we write $s \xrightarrow{k} t$ iff there exists a nonzero path starting at $(s,1)$ and ending at $(t,k+1)$. Note that all states along this path have counter value $\geqslant 1$. In the special case $k = 0$ we have $u \xrightarrow{0} u$ for all $u \in S$, since we allow paths to have length 0. Also note that $s \xrightarrow{+} t$ implies $s \xrightarrow{1} t$, and $s \to t$ implies $s \xrightarrow{0} t$ and finally $s \xrightarrow{-} t$ implies $s \xrightarrow{-1} t$.

Suppose that for some k, $s \xrightarrow{k} t$ holds, and that $(s,n_1)\dots(t,n_l)$ is a nonzero path that witnesses this. Then note that, for any $d > 0$, $(s,n_1+d)\dots(t,n_l+d)$ is also a nonzero path in the same (p)1CA. We will exploit this fact repeatedly.

Proposition 3.3.6. *If* $u \xrightarrow{k_1} v \xrightarrow{k_2} w$ *for some* $u,v,w \in S$*, and either* $k_1 \geqslant 0$ *or* $k_1, k_2 \leqslant 0$*, then* $u \xrightarrow{k_1+k_2} w$*.*

Proof. We join the two paths: from u to v satisfying $\xrightarrow{k_1}$ and from v to w satisfying $\xrightarrow{k_2}$. The resulting path will fulfil the $\xrightarrow{k_1+k_2}$ requirements. For instance if $k_1 \geqslant 0$ and $k_1 + k_2 \geqslant 0$ then the first part of the joined path from u to v starting at $(u,1)$ will reach (v,k_1+1) without encountering a 0-counter state, since it fulfils $\xrightarrow{k_1}$. The second part from v to w will have the counter shifted up by k_1, thus it starts at (v,k_1+1) and finishes at (w,k_1+k_2+1), but does not hit counter 0 in between, since it fulfils $\xrightarrow{k_2}$. □

Example 3.3.7. Note that it might be the case that $u \xrightarrow{k_1} v \xrightarrow{k_2} w$, but $u \xrightarrow{k_1+k_2} w$ does not hold. This can only happen if $k_1 < 0$ and $k_2 \geqslant 0$. For instance, if $\delta = \{(u,1.0,-1,v),\ (v,1.0,1,w)\}$, we have $u \xrightarrow{-1} v \xrightarrow{1} w$, but not $u \xrightarrow{0} w$.

Proposition 3.3.8. *If* $u \xrightarrow{k} v$ *for some* $u,v \in S$*, then:*

- *if* $k < -1$*:* $u \xrightarrow{-1} w \xrightarrow{k+1} v$*, for some* $w \in S$
- *if* $k > 1$*:* $u \xrightarrow{k-1} w \xrightarrow{1} v$*, for some* $w \in S$*,*
- *if* $k = 1$*:* $u \xrightarrow{0} w \xrightarrow{+} z \xrightarrow{0} v$*, for some* $w,z \in S$*,*

(in the last case z *might be equal to* v *and* u *might be equal to* w*).*

Proof. For $k \leqslant -1$ pick as w the first control state on the $u \xrightarrow{k} v$ path from $(u,|k|)$ to $(v,0)$ that has counter value $|k|-1$. For $k \geqslant 1$ pick as w the last state on the $u \xrightarrow{k} v$ path from $(u,1)$ to $(v,k+1)$ that has counter value k. For $k=1$, the transition after state $(w,1)$ has to increase the counter since otherwise it would not be the last state on the nonzero path with counter value 1. So let the next state be $(z,2)$. From that state the nonzero path must reach the end state $(s,2)$ without encountering a state with counter value 1. □

Remark 3.3.9. After cleanup, if a variable x_{st} is on the rhs of a clean equation $x_{uv} = \alpha$, there are 3 (not mutually exclusive) possibilities for how x_{st} occurs in α:

1. as $p^{(0)}_{us} x_{st}$, so $u \to s \xrightarrow{-1} t = v$
2. as $p^{(1)}_{us} x_{st} x_{tv}$, so $u \xrightarrow{+} s \xrightarrow{-1} t \xrightarrow{-1} v$
3. as $p^{(1)}_{uw} x_{ws} x_{st}$, so $u \xrightarrow{+} w \xrightarrow{-1} s \xrightarrow{-1} t = v$

Note that in cases (1.) and (3.) we have $u \xrightarrow{0} s \xrightarrow{-1} t = v$ and in case (2.) we have $u \xrightarrow{1} s \xrightarrow{-1} t \xrightarrow{-1} v$.

Theorem 3.3.10. *If the clean equation $x_{uv} = \alpha$, for a variable $x_{uv} \in X_i$ is nonlinear in the variables belonging to X_i, and if the clean equation for a variable $x_{st} \in X_j$ is nonlinear in the variables belonging to X_j, and there is a path from x_{uv} to x_{st} in dependency graph D, then there is a path from x_{st} to x_{uv} in D.*

Proof. We will first prove a few lemmas. For control states $u,v \in S$, let δ_{uv} denote the usual Kronecker δ: $\delta_{uv} = 1$ if $u = v$ and $\delta_{uv} = 0$ if $u \neq v$.

Lemma 3.3.11. *In dependency graph D, if the shortest path from x_{uv} to x_{st} has a length $k < \infty$ then for some k', $1 - \delta_{vt} \leqslant k' \leqslant k$, we have $u \xrightarrow{k'} s \xrightarrow{-1} t \xrightarrow{-k'} v$.*

Proof. Proof by induction on k. The case $k = 1$ follows from Remark 3.3.9 and the fact that if $t = v$ (in other words $\delta_{vt} = 1$) then $t \xrightarrow{0} v$ holds by default. Assume the statement is true for k and consider some shortest path of length $k+1$ between two variables x_{uv} and x_{st}. Let us consider the variable that is just before x_{st} on this shortest path and assume it is x_{wz} for some $w,z \in S$. Obviously the shortest path in D from x_{uv} to x_{wz} has a length k. We know from the induction assumption that for some $1-\delta_{vz} \leqslant k' \leqslant k$ we have $u \xrightarrow{k'} w \xrightarrow{-1} z \xrightarrow{-k'} v$. On the other hand we know that from x_{wz} we can reach x_{st} in one step, thus from Remark 3.3.9 we get that $w \xrightarrow{1} s \xrightarrow{-1} t \xrightarrow{-1} z$ or $w \xrightarrow{0} s \xrightarrow{-1} t = z$ (both of these form a $w \xrightarrow{-1} z$ path). Considering these two facts

together we get that either $u \xrightarrow{k'} w \xrightarrow{1} s \xrightarrow{-1} t \xrightarrow{-1} z \xrightarrow{-k'} v$ or $u \xrightarrow{k'} w \xrightarrow{0} s \xrightarrow{-1} t = z \xrightarrow{-k'} v$. Now using Proposition 3.3.6 we get that either $u \xrightarrow{k'+1} s \xrightarrow{-1} t \xrightarrow{-(k'+1)} v$ or $u \xrightarrow{k'} s \xrightarrow{-1} t \xrightarrow{-k'} v$. Hence the statement for $k+1$ is true as well. □

Lemma 3.3.12. *If x_{wv} is a nonzero variable and $u \xrightarrow{0} w$ then x_{uv} is also nonzero and depends on x_{wv}.*

Proof. First of all, notice that if $u = w$ then the statement is trivial. Secondly the variable x_{uv} is nonzero since a path $u \xrightarrow{0} w \xrightarrow{-1} v$ forms a $u \xrightarrow{-1} v$ path.

Now if $u \neq w$ then take a path from $(u,1)$ to $(w,1)$ that fulfils $u \xrightarrow{0} w$. Take all the states along that path that have the counter equal to 1: $(s_0,1),(s_1,1),\ldots,(s_n,1)$ where $s_0 = u$ and $s_n = w$ (we know that $n \geqslant 1$ since $u \neq w$). Notice that for all $i \leqslant n$ the variables $x_{s_i v}$ are nonzero because path $s_i \xrightarrow{-1} v$ exists (just take a subpath of the $u \xrightarrow{0} w \xrightarrow{-1} v$ path). Now consider the state $(s_{n-1},1)$. From this state the path cannot take transition reducing the counter to 0 since then the path would finish before reaching $(w,1)$. If the path takes a transition that leaves the counter unchanged then the next state on this path has to be $(s_n,1)$. It is because $(s_n,1)$ was supposed to be the next state after $(s_{n-1},1)$ to have the counter equal to 1. This means that on the rhs of the equation for the variable $x_{s_{n-1}v}$ there is an expression $p^{(0)}_{s_{n-1}s_n} x_{s_n v}$ and as a result variable $x_{s_{n-1}v}$ depends on $x_{s_n v}$. Finally, if the path from $(s_{n-1},1)$ takes a transition $s_{n-1} \xrightarrow{+} z$ then on the rhs of the equation for the variable $x_{s_{n-1}v}$ there is an expression $p^{(1)}_{s_{n-1}z} x_{z s_n} x_{s_n v}$. This is because s_n is the first state after $(z,2)$ that has the value of the counter equal to 1 and so the path $z \xrightarrow{-1} s_n$ exists. Therefore $x_{z s_n} \neq 0$ and similarly $x_{s_n v} \neq 0$ thus after the cleaning step this expression will remain on the rhs of the equation for $x_{s_{n-1}v}$. Hence again $x_{s_{n-1}v}$ depends on $x_{s_n v}$. By an easy induction we can prove that for all $0 \leqslant i < n$ the variable $x_{s_i v}$ depends on $x_{s_{i+1}v}$. Now finally, from the transitivity of this relation we can deduce that variable $x_{s_0 v} (\equiv x_{uv})$ depends on $x_{s_n v} (\equiv x_{wv})$. □

Example 3.3.13. Notice that the assumption about the value of x_{wv} being nonzero is crucial even if we know that x_{uv} is nonzero. For instance in the following example: $\delta = \{(u,0.5,0,w),\ (u,0.5,-1,v),\ (w,1.0,1,w)\}$ we have that $x_{uv} = 0.5 > 0$ and $u \xrightarrow{0} w$, but x_{uv} does not depend on x_{wv} since its value is zero.

Lemma 3.3.14. *A nonzero variable x_{uv} depends on the value of a nonzero variable x_{st} iff for some $k \geqslant 1 - \delta_{vt}$ we have $u \xrightarrow{k} s \xrightarrow{-1} t \xrightarrow{-k} v$.*

Proof. ($\Rightarrow$) Note that if $x_{uv} \equiv x_{st}$ then $u = s$ and $v = t$, so $1 - \delta_{vt} = 0$ and $s \xrightarrow{-1} t$ (since $x_{st} > 0$) thus we have $u \xrightarrow{0} u = s \xrightarrow{-1} t \xrightarrow{0} t = v$.

If $x_{uv} \not\equiv x_{st}$ then there is a path in D from x_{uv} to x_{st} and so there is also the shortest one. Let us denote its length by k'. From Lemma 3.3.11 for some k, such that $1-\delta_{vt} \leqslant k \leqslant k'$, we have $u \xrightarrow{k} s \xrightarrow{-1} t \xrightarrow{-k} v$.

($\Leftarrow$) Of course x_{uv} and x_{st} are both nonzero since from $u \xrightarrow{k} s \xrightarrow{-1} t \xrightarrow{-k} v$ we know that $s \xrightarrow{-1} t$ and $u \xrightarrow{-1} v$ holds.

If it happens that $k = 0$ then necessarily $v = t$. In other words we know that $u \xrightarrow{0} s \xrightarrow{-1} t = v$ which means that $u \xrightarrow{0} s$ and $x_{st} > 0$. Now from Lemma 3.3.12 we get that x_{ut} ($\equiv x_{uv}$) is nonzero and depends on x_{st}.

The rest of the proof is by induction on k. If $k = 1$ then $u \xrightarrow{1} s \xrightarrow{-1} t \xrightarrow{-1} v$. Of course we instantly have that x_{uv}, x_{st}, x_{tv} are nonzero. From Proposition 3.3.8 we know that we can decompose the $u \xrightarrow{1} s$ part into $u \xrightarrow{0} w \xrightarrow{+} z \xrightarrow{0} s$ for some $w, z \in S$ and the whole path would look as follows: $u \xrightarrow{0} w \xrightarrow{+} z \xrightarrow{0} s \xrightarrow{-1} t \xrightarrow{-1} v$. Furthermore, $z \xrightarrow{-1} t$ and $w \xrightarrow{-1} v$, so x_{zt} and x_{wv} are nonzero. From this we can deduce that on the rhs of the equation for x_{wv} we will have an expression $p^{(1)}_{wz} x_{zt} x_{tv}$. This means that x_{wv} depends on variable x_{zt}. In addition from the facts $u \xrightarrow{0} w$, $z \xrightarrow{0} s$ and Lemma 3.3.12 we get that x_{uv} depends on x_{wv}, and x_{zt} depends on x_{st}. Finally, from the transitivity of this relation we obtain that x_{uv} depends on x_{st}.

Now assume that the statement is true for some k' and let us consider a $u \xrightarrow{k'+1} s \xrightarrow{-1} t \xrightarrow{-(k'+1)} v$ path. From Proposition 3.3.8 we know that for some $w, z \in S$ we can decompose this path into a $u \xrightarrow{k'} w \xrightarrow{1} s \xrightarrow{-1} t \xrightarrow{-1} z \xrightarrow{-k'} v$ path. It follows that $w \xrightarrow{1} s \xrightarrow{-1} t \xrightarrow{-1} z$ and $u \xrightarrow{k'} w \xrightarrow{-1} z \xrightarrow{-k'} v$. Now from the induction assumption for $k = 1$ we get that x_{wz} is nonzero and depends on x_{st} and from the induction assumption for $k = k'$ we get that x_{uv} is nonzero and depends on x_{wz}. This means that x_{uv} also depends on x_{st}. □

Example 3.3.15. It might be the case that $u \xrightarrow{0} s \xrightarrow{-1} t \xrightarrow{0} v$ where $t \neq v$, but x_{uv} does not depend on x_{st} like in the following example: $\delta = \{(u, 1.0, 0, s),\ (s, 1.0, -1, t),\ (t, 1.0, 0, v)\}$.

Lemma 3.3.16. *If the clean equation for a variable $x_{uv} \in X_i$ is nonlinear in the variables belonging to X_i then for some $k_0 \geqslant 1, k_1 \geqslant 0$ and some $w \in S$ we have $u \xrightarrow{k_0} u \xrightarrow{-1} v \xrightarrow{1-k_0} w \xrightarrow{k_1} u \xrightarrow{-1} v \xrightarrow{-k_1} v$.*

Proof. Since x_{uv} is nonlinear in the variables belonging to X_i then from Remark 3.3.9 we can deduce that for some $s, t \in S$ we have $x_{st}, x_{tv} \in X_i$ and the clean equation for x_{uv} has on the rhs an expression $p^{(1)}_{us} x_{st} x_{tv}$. It follows that $u \xrightarrow{+} s \xrightarrow{-1} t \xrightarrow{-1} v$. Since x_{st} is in the same SCC as x_{uv} then there has to be a path from x_{st} to x_{uv} in the graph D and using Lemma 3.3.11 we get that for some $k \geqslant 1 - \delta_{vt}$ we have $s \xrightarrow{k} u \xrightarrow{-1} v \xrightarrow{-k} t$.

From the same argument we get that for some $k' \geqslant 0$ we have $t \xrightarrow{k'} u \xrightarrow{-1} v \xrightarrow{-k'} v$. Now joining these paths together we get $u \xrightarrow{+} s \xrightarrow{k} u \xrightarrow{-1} v \xrightarrow{-k} t \xrightarrow{k'} u \xrightarrow{-1} v \xrightarrow{-k'} v$. Finally, using Proposition 3.3.6 we have $u \xrightarrow{k+1} u \xrightarrow{-1} v \xrightarrow{-k} t \xrightarrow{k'} u \xrightarrow{-1} v \xrightarrow{-k'} v$. □

We can now finish the proof of Theorem 3.3.10. Using Lemma 3.3.16 we get that for some $k_0, l_0 \geqslant 1$, $k_1, l_1 \geqslant 0$ and $w, z \in S$ we have $u \xrightarrow{k_0} u \xrightarrow{-1} v \xrightarrow{1-k_0} w \xrightarrow{k_1} u \xrightarrow{-1} v \xrightarrow{-k_1} v$ and $s \xrightarrow{l_0} s \xrightarrow{-1} t \xrightarrow{1-l_0} z \xrightarrow{l_1} s \xrightarrow{-1} t \xrightarrow{-l_1} t$. We can simplify the later to $s \xrightarrow{l_0} s \xrightarrow{-1} t \xrightarrow{-l_0} t$ for some $l_0 \geqslant 1$ using Proposition 3.3.6.

Since there is a path from x_{uv} to x_{st} then from Lemma 3.3.11 we have $u \xrightarrow{k} s \xrightarrow{-1} t \xrightarrow{-k} v$ for some $k \geqslant 1 - \delta_{vt}$. Now we will show that $s \xrightarrow{k'} u \xrightarrow{-1} v \xrightarrow{-k'} t$ holds for some $k' \geqslant 1$ and using Lemma 3.3.14 we will get that the variable x_{st} depends on the variable x_{uv}. We start the $s \xrightarrow{k'} u \xrightarrow{-1} v \xrightarrow{-k'} t$ path by iterating the $s \xrightarrow{l_0} s$ path n times for sufficiently large n obtaining a $s \xrightarrow{n \cdot l_0} s$ path: $\underbrace{s \xrightarrow{l_0} s \xrightarrow{l_0} s \xrightarrow{l_0} \ldots \xrightarrow{l_0} s}_{n \text{ times}}$. We will see how big n should be later. Now from the last s we do: $s \xrightarrow{-1} t \xrightarrow{-k} v \xrightarrow{1-k_0} w \xrightarrow{k_1} u \xrightarrow{k_0} u \xrightarrow{-1} v \xrightarrow{-k_1} v \xrightarrow{-k_1} v \xrightarrow{1-k_0} w \xrightarrow{k_1} u \xrightarrow{k_0} u \xrightarrow{k} s \xrightarrow{-1} t$ and after that we iterate n times the $t \xrightarrow{-l_0} t$ path. Along the whole path the value of the counter is changed by: $nl_0 - 1 - k + 1 - k_0 + k_1 + k_0 - 1 - k_1 - k_1 + 1 - k_0 + k_1 + k_0 + k - 1 - nl_0 = -1$. Now if $nl_0 > k + k_0 + k_1$ (this can be done since $l_0 \geqslant 1$) then using Proposition 3.3.6 we can rewrite it as $s \xrightarrow{nl_0 - k + k_1} u \xrightarrow{-1} v \xrightarrow{-nl_0 + k - k_1} t$. Essentially, we make the value of the counter sufficiently high at the beginning of the path in order to prevent it from reaching counter 0 before it reaches the final t state (with $(w, nl_0 - k - k_0 - k_1)$ being the state with the lowest value of the counter before that point). Now finally, since $nl_0 - k + k_1 \geqslant 1$ it follows from Lemma 3.3.14 that x_{st} depends on x_{uv}. □

Corollary 3.3.17. *In the DAG, H, along any directed path $X_{i_1} X_{i_2} \ldots X_{i_r}$ of SCCs there is at most one nonlinear SCC.*

Proof. Let X_i and X_j $(i < j)$ be two SCCs on such a path. If inside these two SCCs there are variables $x \in X_i$ and $y \in X_j$ whose equations are nonlinear in the variables belonging to X_i and X_j, respectively, then since there is a path from x to y in D (in other words x depends on y) we know from Theorem 3.3.10 that there is also a path from y to x. But that implies x and y are in the same SCC, a contradiction. □

In Figure 3.6 we can see the decomposition graph, H, of the underlying equation system of an example p1CA. Corollary 3.3.17 implies that any path in H has to contain at most one nonlinear SCC. Notice that this fact does not hold for general RMCs and even for 1-exit RMCs. For the example RMC depicted in Figure 3.5, if x_k denotes the

probability of termination in A_k ($k = 0, 1, \ldots, n$, for some fixed $n \geqslant 1$) then the equation system can written down (after some simplifications) as $x_k = \frac{1}{2}x_k^2 + \frac{1}{2}x_{k-1}^2$ for $1 \leqslant k \leqslant n$ and $x_0 = 1$. None of these variables is equal to 0 and each x_k depends only on x_{k-1} and nonlinearly on itself. Hence, after decomposing this equation system into SCCs we would get a long chain of n nonlinear SCCs one after the another.

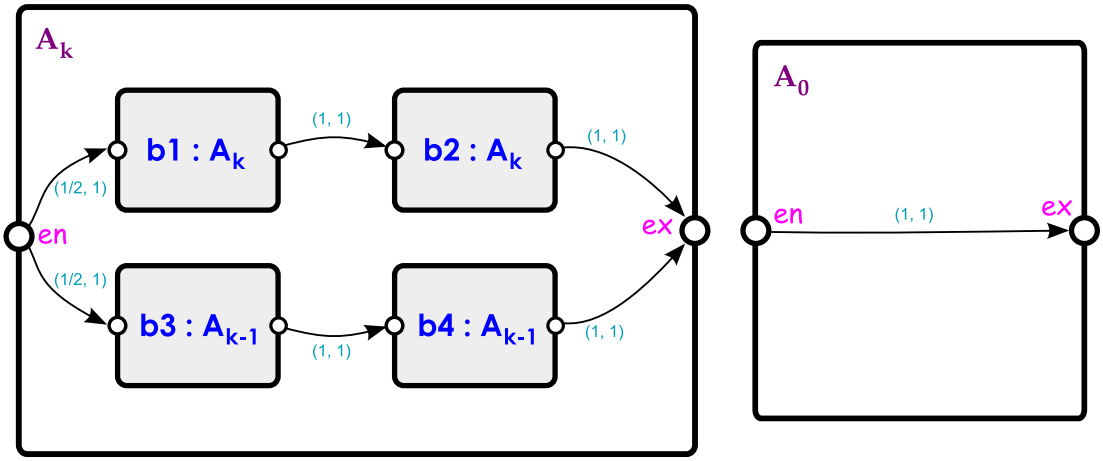

Figure 3.5: An example family of 1-exit RMCs ($k = 1, \ldots, n$, for some fixed $n \geqslant 1$) whose decomposition of the corresponding equation system into SCCs have multiple nonlinear SCCs one after the another.

3.4 New upper bounds on Newton's method for QBDs

We will now exploit the structural results about p1CAs established in Section 3.3, to establish strong new upper bounds on the performance of (decomposed) Newton's method on QBDs. In our analysis in this section, we assume a unit-cost exact rational arithmetic RAM model of computation. In other words, individual arithmetic operations on rationals have unit cost, regardless of the potential blow-ups involved in the encoding size of rational numbers.

Recall that in (multi-variate) Newton's method, we are given a suitably differentiable map $F : \mathbb{R}^n \mapsto \mathbb{R}^n$, and we wish to find a solution to the system of equations $F(\mathbf{x}) = \mathbf{0}$. Starting at some $\mathbf{x^0} \in \mathbb{R}^n$, the method works by iterating $\mathbf{x}^{k+1} := \mathbf{x}^k - (F'(\mathbf{x}^k))^{-1}F(\mathbf{x}^k)$, where $F'(\mathbf{c})$ is the ***Jacobian matrix*** of partial derivatives, whose (i, j) entry is $\frac{\partial F_i}{\partial x_j}$ evaluated at $\mathbf{c}$.

In the setting of p1CAs, we have a system of n equations in n variables, $x_i = P_i(x)$, which we can denote by $\mathbf{x} = P(\mathbf{x})$. Thus, we wish to find a solution to $F(\mathbf{x}) \doteq P(\mathbf{x}) - \mathbf{x} =$

$u \xrightarrow{+} v$ $\quad v \xrightarrow{-} v$ $\quad w \xrightarrow{+} w$ $\quad a \xrightarrow{+} a$ $\quad b \xrightarrow{+} b$ $\quad c \xrightarrow{+} b$ $\quad d \xrightarrow{-} d$

$u \xrightarrow{+} u$ $\quad v \xrightarrow{-} u$ $\quad w \to u$ $\quad a \xrightarrow{-} b$ $\quad b \xrightarrow{-} b$ $\quad d \to d$

$u \xrightarrow{-} u$ $\quad a \xrightarrow{-} c$ $\quad b \xrightarrow{+} d$

$$x_{vv} = \frac{1}{2}$$
$$x_{vu} = \frac{1}{2}$$
$$x_{uv} = \frac{1}{3}(x_{vv}^2 + x_{vu}x_{uv}) + \frac{1}{3}(x_{uv}x_{vv} + x_{uu}x_{uv})$$
$$x_{uu} = \frac{1}{3}(x_{vv}x_{vu} + x_{vu}x_{uu}) + \frac{1}{3}(x_{uv}x_{vu} + x_{uu}^2) + \frac{1}{3}$$
$$x_{wv} = \frac{1}{2}(x_{wv}x_{vv} + x_{wu}x_{uv}) + \frac{1}{2}x_{uv}$$
$$x_{wu} = \frac{1}{2}(x_{wv}x_{vu} + x_{wu}x_{uu}) + \frac{1}{2}x_{uu}$$
$$x_{ac} = \frac{1}{3}$$
$$x_{dd} = \frac{1}{2} + \frac{1}{2}x_{dd}$$
$$x_{bb} = \frac{1}{3}x_{bb}^2 + \frac{1}{3}$$
$$x_{bd} = \frac{1}{3}(x_{bd}x_{dd} + x_{bb}x_{bd}) + \frac{1}{3}x_{dd}^2$$
$$x_{cd} = x_{bd}x_{dd} + x_{bb}x_{bd}$$
$$x_{cb} = x_{bb}^2$$
$$x_{ab} = \frac{1}{3}(x_{ac}x_{cb} + x_{ab}x_{bb}) + \frac{1}{3}$$
$$x_{ad} = \frac{1}{3}(x_{ad}x_{dd} + x_{ac}x_{cd} + x_{ab}x_{bd})$$

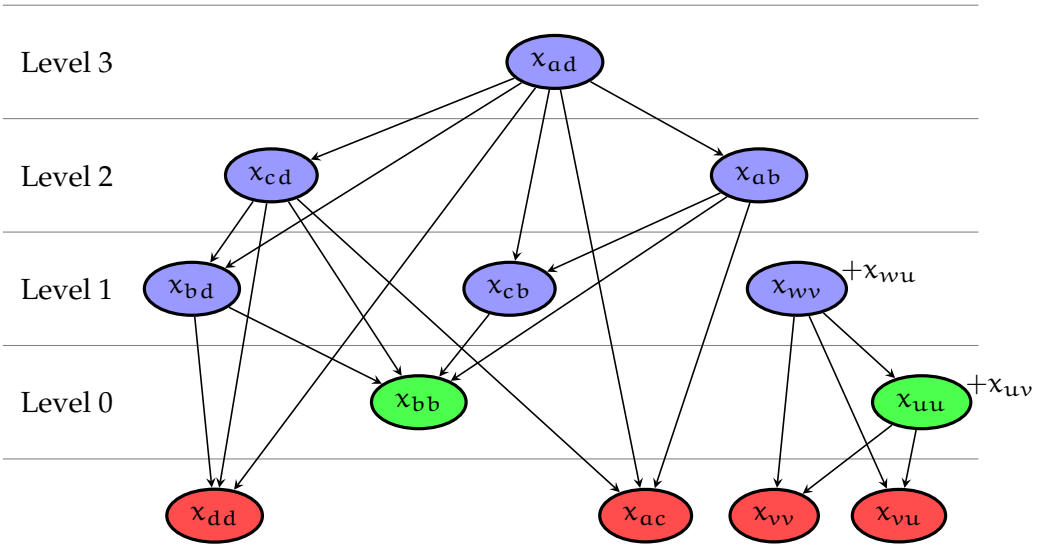

Figure 3.6: At the top, a list of transitions of an example p1CA ($\xrightarrow{+}, \xrightarrow{-}, \to$ modify the counter by $+1, -1, 0$ respectively). At any particular control state all available transitions are taken with equal probability. Below we can see the cleaned-up equation system (i.e., after all zero-valued variables were removed) whose LFP would give us all nonzero entries of the G matrix. Finally, a DAG of SCCs corresponding to that equation system is drawn. Each ellipse represents one SCC: the equation system for each red SCC is linear, the equation system for each green SCC is nonlinear and for each blue one is linear, but at least one of its constants depends on the value of some variable in a green nonlinear SCC. Notice that only two SCCs consist of more than one node: $\{x_{uu}, x_{uv}\}$ and $\{x_{wv}, x_{wu}\}$. A blue SCC is placed at level k if the longest path from it to some green SCC has lenght k. In this example the height of the decomposition (the maximal level of any SCC) is $h_{max} = 3$.

0. Note that these are polynomial functions, and thus certainly differentiable.

We shall solve this system of equations using the ***decomposed Newton's method*** of [EY05s], which applies more generally not just to systems $\mathbf{x} = P(\mathbf{x})$ arising for p1CAs, but to any monotone system $\mathbf{x} = P(\mathbf{x})$ of polynomial equations (i.e., where the coefficients in $P(\mathbf{x})$ are non-negative) which has a non-negative solution. Specifically, for any such system $\mathbf{x} = P(\mathbf{x})$ which has been ***cleaned up*** (i.e., variables which are necessarily zero in any least solution have been removed, something which can be done easily in polynomial time [EY05s]) we form the dependency graph D for the nonzero variables in the corresponding cleaned system of equations, we decompose D into SCCs, and form the DAG of SCCs, H. We then "solve" for the values of variables in each SCC of H, "bottom up" by applying Newton's method starting at the vector 0 to the equations for each SCC, beginning with bottom SCCs. Once one SCC is "solved" the values computed for the variables in that SCC are plugged into equations in higher SCCs that depend on those values. (See [EY05s] for details.)

Of course, since values may in general be irrational and are only converged to in the limit, we have to specify more carefully what we mean by "solve" an SCC. This is where we make crucial use of the special structure of SCCs in the case of p1CAs and QBDs (see Corollary 3.3.17 and Figure 3.6 for an example). By Corollary 3.3.17, for any nonlinear SCC, X_i, it must be the case that any other SCC, X_j, for which there is a path in H from X_i to X_j, is linear, i.e., any variable $x_{uv} \in X_j$ has a corresponding clean equation $x_{uv} = \alpha$ which is linear in the variables of X_j, assuming variables in even lower SCCs have been assigned fixed values. It was shown in [EY05s] (in the more general setting of monotone systems arising from RMCs and pPDSs) that for such linear SCCs, X_j, Newton's method converges in just one iteration, starting at the vector 0, to the exact rational least fixed point (LFP) solution we are after (i.e., to the values $G_{u,v}$ for these variables in $x_{uv} \in X_j$). Thus, in a bottom up fashion we can compute the exact solutions $G_{u,v}$ for those variables x_{uv} which are in linear SCCs below any nonlinear SCC. After computing these values we plug them into equations for variables in higher SCCs that depend on them, and we eliminate the linear SCC which was already solved. We do this until there are no bottom linear SCCs remaining.

We next have to apply Newton's method to nonlinear SCCs, which can have irrational solutions which are only converged to in the limit. How many iterations are "enough" to get to within a desired additive error $\varepsilon > 0$ of the nonzero termination probabilities $G_{u,v}$ for the variables in a nonlinear SCC? For this, we will use the following recent result by Esparza et. al. (Theorem 3.2 of [EKL08]) on the behavior of

Newton's method on precisely such strongly connected monotone nonlinear systems. Let $P(X)$ be a cleaned monotone system of polynomials (i.e., $P(X)$ consists of n multivariate polynomials, P_i, $i = 1,\ldots,n$, in the variables $X = x_1,\ldots,x_n$), such that $X = P(X)$ has a non-negative solution, and since it is cleaned, only positive solutions, and therefore a least fixed point (LFP) solution, $q^* > 0$. A vector q' is said to have i ***valid bits*** of q^* if $|q^*_j - q'_j|/q^*_j \leqslant 2^{-i}$ for every $1 \leqslant j \leqslant n$.

Theorem 3.4.1. *([EKL08]) Let $P(X)$ be a cleaned strongly connected monotone system of quadratic polynomials (i.e., $P(X)$ consists of n quadratic multi-variate polynomials in n variables). Let c_{min} be the smallest nonzero coefficient of any monomial in $P(X)$, and let μ_{min} and μ_{max} be the minimal and maximal components of the LFP vector $q^* > 0$, respectively. Let $k_f = n \cdot \log(\frac{\mu_{max}}{c_{min} \cdot \mu_{min} \cdot \min\{\mu_{min},1\}})$. Let $\mathbf{x}^j$ denote the vector of values obtained after j iterations of Newton's method on the system $F(X) = P(X) - X$, starting with the initial all 0 vector, $\mathbf{x}^0 = \mathbf{0}$. Then for every $i \geqslant 0$, $\mathbf{x}^{(\lceil k_f \rceil + i)}$ has i valid bits of q^*.*

For a given p1CA, we hereafter use m to denote the maximum number of bits required to encode the integer numerators and denominators of transition probabilities of the p1CA. Thus, in particular, the smallest nonzero transition probability is $p_{min} \geqslant 1/2^m$.

Now, using Theorem 3.4.1, together with the structural properties we have established for p1CAs, we prove the following strong bound on the number of iterations of Newton's method required to get i valid bits of precision of the termination probabilities $G_{u,v}$, for the nonlinear SCCs of the fixed point equations associated with p1CAs:

Theorem 3.4.2. *Let $P(X)$ be the cleaned strongly connected monotone system of quadratic polynomials associated with a nonlinear SCC, X_i, of the decomposed system of equations associated with a p1CA, and where the exact rational values $G_{u,v}$ associated with variables x_{uv} in already solved "lower" linear SCCs have been substituted for x_{uv} on the right hand side of equations for variables in X_i. Suppose that the p1CA has n control states, and thus $|X_i| \leqslant n^2$, and let $G|_{X_i}$ denote those entries $G_{u,v}$ of the matrix G, such that $x_{uv} \in X_i$. Then, starting with $\mathbf{x}^0 := \mathbf{0}$, for every $i \geqslant 0$, the Newton iteration $\mathbf{x}^{(4mn^5 + mn^2 + i)}$ has i valid bits of $G|_{X_i}$.*

Proof. For the cleaned system $X = P(X)$ associated with a p1CA, A, by Corollary 3.3.5, $p_{min}^{n^3} \leqslant q^* \leqslant 1$ (coordinate-wise inequality), where $p_{min} > 0$ is the smallest positive probability on any transition of A. Note, in particular, that $\mu_{max} \leqslant 1$, and $\mu_{min} \geqslant p_{min}^{n^3} \geqslant \frac{1}{2^{mn^3}}$. Furthermore, note that because the entire system of nonlinear equations for a p1CA is quadratic, the smallest coefficient c_{min} of any monomial in the system $X = P(X)$ for this nonlinear SCC, can only arise as the product of p_{min}

times at most 2 previously computed values $G_{u',v'}$ and $G_{u'',v''}$ for variables $x_{u'v'}$ and $x_{u''v''}$ which appeared in lower (linear) SCCs. Again, by Corollary 3.3.5, we know that $G_{u',v'}, G_{u'',v''} \geqslant p_{min}^{n^3}$, and thus $c_{min} \geqslant p_{min}^{2n^3+1} \geqslant 1/2^{m(2n^3+1)}$. Thus, noting that the cleaned system $X = P(X)$ for a p1CA with n control states has at most n^2 variables, the expression for k_f in Theorem 3.4.1 can be seen to be $k_f \leqslant n^2 \cdot \log(2^{2mn^3+m} 2^{mn^3} 2^{mn^3}) = 4mn^5 + mn^2$. □

Theorem 3.4.2 implies that we can compute i bits of the values $G_{u,v}$ for variables x_{uv} in nonlinear SCCs of the system $X = P(X)$ associated with a p1CA (QBD), using only a number of iterations of Newton's method which is polynomially bounded in the size of the p1CA, and linearly bounded in i.

We now have to confront a major difficulty: there may be other, linear, SCCs, X_r, which are "above" such nonlinear SCCs in H. Specifically, there may be a linear SCC X_r, from which there is a path in H to a nonlinear SCC, X_i. In order to be able to (approximately!) compute $G_{u,v}$ for variables $x_{uv} \in X_r$, we have to first approximately compute the (possibly irrational) values $G_{u',v'}$, for $x_{u'v'} \in X_i$, and substitute this value in occurrences of $x_{u'v'}$ in equations for higher linear SCCs. The question arises: how many bits of precision i, do we need to compute $G_{u',v'}$ to in order to compute $G_{u,v}$ to within i bits of precision? To answer this, we employ a classic bound, based on condition numbers, on errors in the solution of a linear systems.

Theorem 3.4.3. *(see, e.g., [IK66], Chap 2.1.2, Thm 3.[3]) Consider a system of linear equations,* $Bx = b$, *where* $B \in \mathbb{R}^{n \times n}$ *and* $b \in \mathbb{R}^n$. *Suppose* B *is non-singular, and* $b \neq 0$. *Let* $x^* = B^{-1}b$ *be the unique solution to this linear system, and suppose* $x^* \neq 0$. *Let* $\|\cdot\|$ *denote any vector norm and associated matrix norm (when applied to vectors and matrices, respectively). Let* $\mathrm{cond}(B) = \|B\| \cdot \left\|B^{-1}\right\|$ *denote the condition number of* B. *Let* $\varepsilon, \varepsilon' > 0$, *be values such that* $\varepsilon' < 1$, *and* $\varepsilon \cdot \mathrm{cond}(B) \leqslant \varepsilon'/4$. *Let* $\mathcal{E} \in \mathbb{R}^{n \times n}$ *and* $\zeta \in \mathbb{R}^n$, *be such that* $\frac{\|\mathcal{E}\|}{\|B\|} \leqslant \varepsilon$, $\frac{\|\zeta\|}{\|b\|} \leqslant \varepsilon$, *and* $\|\mathcal{E}\| < 1/\left\|B^{-1}\right\|$. *Then the system of linear equations* $(B + \mathcal{E})x = b + \zeta$ *has a unique solution* x_ε^* *such that:*

$$\frac{\|x_\varepsilon^* - x^*\|}{\|x^*\|} \leqslant \varepsilon'$$

We will apply this theorem using the l_∞ vector norm and induced matrix norm (***maximum absolute row sum***): $\|x\|_\infty = \max_i |x_i|$ and $\|A\|_\infty = \max_i \sum_j \left|a_{ij}\right|$.

Suppose that the fixed point equation system for a linear SCC of a p1CA, which lives "above" some nonlinear SCCs in the DAG H, looks like this: $x = Ax + b$. We

[3] Our statement is weaker, but derivable from that theorem.

know that $A \geqslant 0$ is an irreducible matrix (precisely because the variables being solved for are in the same SCC), $b \geqslant 0$, and $b \neq 0$ since otherwise the unique solution for this system would be $q^* = 0$, and zero variables were already eliminated. We can of course rewrite this linear equation as $(I - A)x = b$. It follows from a more general result in [EY05s] about the decomposed systems of equations arising for RMCs (pPDSs) (specifically, see Lemma 17 and Theorem 14 of [EY05s]), that $\rho(A) < 1$, where $\rho(A)$ denotes the spectral radius of A, and that therefore $(I - A)$ is non-singular, and furthermore $(I - A)^{-1} = (\sum_{i=0}^{\infty} A^i)$. Thus the LFP of this equation system is $q^* = (I - A)^{-1}b = (\sum_{k=0}^{\infty} A^k)b$. To prove bounds on errors in "higher" linear SCCs, when values in non-linear SCCs are approximated, we will need the following two lemmas:

Lemma 3.4.4. *Let $A \in \mathbb{R}^{n \times n}_{\geqslant 0}$ and $b \in \mathbb{R}^n_{\geqslant 0}$, such that: $(I - A)^{-1} = \sum_{k=0}^{\infty} A^k$, and we have $(\sum_{k=0}^{\infty} A^k)b \leqslant \mathbf{1}$, and A is an irreducible non-negative matrix whose smallest nonzero entry is $c > 0$, and $b \neq 0$ and $p > 0$ is the largest entry of b. Then: $\left\|\sum_{k=0}^{\infty} A^k\right\|_\infty \leqslant \frac{n}{pc^n}$.*

Proof. Let a^d_{ij} and a^*_{ij} denote the (i,j) entry of matrix A^d and $A^* = \sum_{k=0}^{\infty} A^k$ respectively. Since A is irreducible, for every pair of indices i, j, there exists a power $1 \leqslant d \leqslant n$ such that $a^d_{ij} > 0$. First, notice that it has to be $c < 1$ as otherwise all entries of $(\sum_{k=0}^{\infty} A^k)$ would diverge to ∞. Furthermore, since the smallest nonzero entry of A is c, we have $a^d_{ij} \geqslant c^d$.

We know that $A^*b \leqslant \mathbf{1}$. Wlog we can assume that the first entry of b is $b_1 = p$, by basically permuting rows/columns of A and b. Now the i-th entry of A^*b is $(A^*b)_i = \sum_{j=1}^{n} a^*_{ij} b_j \leqslant 1$ and thus obviously $(A^*b)_i \geqslant a^*_{i1} b_1 = a^*_{i1} p$. It follows that $a^*_{i1} \leqslant \frac{1}{p}$, for all i. At the same time, for all $d \geqslant 0$, $A^* A^d = (\sum_{k=0}^{\infty} A^k) A^d = \sum_{k=d}^{\infty} A^k \leqslant \sum_{k=0}^{\infty} A^k = A^*$. Thus $(A^* A^d)_{(i,1)} = \sum_{j=1}^{n} a^*_{ij} a^d_{j1} \leqslant a^*_{i1}$. Let $a'_{i1} = (\sum_{d=1}^{n} A^* A^d)_{(i,1)}$. Thus, $a'_{i1} \leqslant n a^*_{i1} \leqslant n/p$. On the other hand:

$$a'_{i1} = \sum_{d=1}^{n} \sum_{j=1}^{n} a^*_{ij} a^d_{j1} = \sum_{j=1}^{n} a^*_{ij} \left(\sum_{d=1}^{n} a^d_{j1} \right) \geqslant c^n \sum_{j=1}^{n} a^*_{ij}$$

The last inequality holds because, for every j, for some $1 \leqslant d \leqslant n$ we have $a^d_{j1} \geqslant c^d \geqslant c^n$. Therefore for all i we have $\sum_{j=1}^{n} a^*_{ij} \leqslant \frac{n}{pc^n}$ and thus $\|A^*\|_\infty \leqslant \frac{n}{pc^n}$. □

Note that computing the probability of reaching a given subset of states in an ordinary finite-state Markov Chain can be seen as finding a fixed point solution to a similar linear equation system $x = Ax + b$ in which matrix A is always stochastic (the sum of all the entries in each row is $\leqslant 1$). The following example shows that matrix A does not have to be stochastic for linear equation systems for p1CAs.

Example 3.4.5. The sum of one of the rows of matrix A of the following p1CA is greater than 1: $\delta = \{ (a,1.0,-1,v), (b,1.0,-1,v), (v,1/2,-1,a), (v,1/2,-1,b), (w,1.0,1,u), (u,3/4,1,w), (u,1/4,-1,v) \}$. The equation system for this p1CA looks as follows:

$$x_{uv} = \frac{1}{4} + \frac{3}{4}(x_{wa}x_{av} + x_{wb}x_{bv}) \qquad x_{av} = x_{bv} = 1$$

$$x_{wa} = x_{uv}x_{va} \qquad x_{va} = x_{vb} = \frac{1}{2}$$

$$x_{wb} = x_{uv}x_{vb}$$

The matrix A for the linear equation system for SCC $\{x_{uv}, x_{wa}, x_{wb}\}$ is equal to $\begin{pmatrix} 0 & 3/4 & 3/4 \\ 1/2 & 0 & 0 \\ 1/2 & 0 & 0 \end{pmatrix}$ and $b = [1/4,0,0]^T$.

In the case where c is very close to 1 then the following proposition (that will not be used later on) can provide a better estimate for $\left\|\sum_{k=0}^{\infty} A^k\right\|_\infty$:

Proposition 3.4.6. *Let $A \in \mathbb{R}^{n\times n}_{\geqslant 0}$ and $b \in \mathbb{R}^n_{\geqslant 0}$, such that: $(I-A)^{-1} = \sum_{k=0}^{\infty} A^k$, and $(\sum_{k=0}^{\infty} A^k)b \leqslant \mathbf{1}$, and A is an aperiodic irreducible non-negative matrix whose smallest nonzero entry is $c > 0$, and $b \neq 0$ and $p > 0$ is the largest entry of b. Then:*

$$\left\|\sum_{k=0}^{\infty} A^k\right\|_\infty \leqslant \frac{1}{pc^{n^2-2n+1}}$$

Proof. Since A is aperiodic all entries of A^{n^2-2n+1} are positive (see Theorem 4.14 in [BP94]) and similar as before all of these entries have to be larger than c^{n^2-2n+1}. The rest of the proof is almost identical to the proof of Lemma 3.4.4. Again we have $a^*_{i1} \leqslant \frac{1}{p}$, for all i (where as before a^d_{ij} and a^*_{ij} denote the (i,j) entry of matrix A^d and A^* respectively) and $A^*A^{n^2-2n+1} \leqslant A^*$. Thus the $(i,1)$ entry of $A^*A^{n^2-2n+1}$ which is equal to $\sum_{j=1}^n a^*_{ij}a^{n^2-2n+1}_{j1}$ is smaller or equal to a^*_{i1}. Therefore $c^{n^2-2n+1}\sum_{j=1}^n a^*_{ij} \leqslant \sum_{j=1}^n a^*_{ij}a^{n^2-2n+1}_{j1} \leqslant a^*_{i1} \leqslant \frac{1}{p}$ for any i and so $\|A^*\|_\infty \leqslant \frac{1}{pc^{n^2-2n+1}}$ follows. □

Lemma 3.4.7. *Let X_r be a linear SCC of the cleaned equation system for a p1CA, whose corresponding linear equation system is $x = Ax + b$, after variables x_{uv} in lower SCCs have been substituted by their exact (possibly irrational) values $G_{u,v}$. Let p_{min} denote the smallest positive probability on any transition of the p1CA, and let n be its number of control states (again we use m to denote the maximum number of bits required to represent the numerator and denominator of rational transition probabilities in the p1CA). Then the following bounds hold:*

1. $\frac{1}{2^{2mn^3+m}} \leqslant p_{min}^{2n^3+1} \leqslant \|(I-A)\|_\infty \leqslant n+1$
2. $\left\|(I-A)^{-1}\right\|_\infty \leqslant \frac{n^2}{p_{min}^{5n^5}} \leqslant n^2 \cdot 2^{5mn^5}$
3. $\mathrm{cond}(I-A) \leqslant \frac{2n^3}{p_{min}^{5n^5}} \leqslant 2n^3 \cdot 2^{5mn^5}$
4. $\|b\|_\infty \geqslant p_{min}^{2n^3+1} \geqslant \frac{1}{2^{2mn^3+m}}$

Proof. We first show that $\|A\|_\infty \leqslant n$, and therefore $\|I-A\|_\infty \leqslant n+1$ (because A is non-negative). To see this, note that because this is a linear SCC, this means that the equations (3.1) for every variable x_{uv} of a linear SCC, X_r, must take the form: $x_{uv} = b_{uv} + (\sum_w p_{uw}^{(0)} x_{wv}) + \sum_y p_{uy}^{(1)} \sum_z x'_{yz} x'_{zv}$, but such that for each z, either x'_{yz} has been assigned a fixed constant ($\leqslant 1$) or x'_{zv} is a fixed constant ($\leqslant 1$). This is because, one such variable in each quadratic term must belong to a lower SCC and was thus substituted by a constant. Thus, summing the coefficients for all variables on the right hand side, we see that since $\sum_{c=-1}^{1} \sum_w p_{uw}^{(c)} \leqslant 1$, the full sum $\sum_j a_{ij}$ of all entries in row i of A corresponding to the variable x_{uv}, cannot be more than n, the number of control states.

Before showing the lower bound on $\|I-A\|_\infty$, next we show the bound $\|b\|_\infty \geqslant p_{min}^{2n^3+1}$. Observe that since the equation system has been cleaned, the least fixed point solution for all variables, including in linear SCCs, is nonzero, and therefore there must exist at least one equation $x_{uv} = \alpha$ in the linear SCC with a non-negative constant term in α. The only ways such a constant term can arise is as a sum of terms of the form p, or px', or $px'x''$, where p is a transition probability of the p1CA and x' and x'' are variables in lower SCCs which have been assigned fixed constants. By Corollary 3.3.5, we have that $\|b\|_\infty \geqslant p_{min}^{2n^3+1}$.

Next, in order to estimate $\left\|(I-A)^{-1}\right\|_\infty$ note that, using Corollary 3.3.5, all nonzero entries of A are $\geqslant p_{min} \cdot (p_{min})^{n^3} = (p_{min})^{n^3+1}$. This is because all coefficients are either equal to some $p_{uv}^{(c)}$ or to $p_{uv}^{(c)} \cdot x_{wz}$ where x_{wz} is a variable from a lower SCC that has been substituted by a constant. We now use Lemma 3.4.4. Note that the dimensions of our matrix A here can in fact be as large as $n^2 \times n^2$ (because n is the number of control states, and the dimensions of A are based on the number of variables in the SCC). We thus get from Lemma 3.4.4, using the bound $\|b\|_\infty \geqslant p_{min}^{2n^3+1}$, and the fact that all nonzero entries of A are $\geqslant (p_{min})^{n^3+1}$, that $\left\|(I-A)^{-1}\right\|_\infty = \left\|\sum_{k=0}^{\infty} A^k\right\|_\infty \leqslant \frac{n^2}{p_{min}^{n^5+n^2+2n^3+1}} \leqslant \frac{n^2}{p_{min}^{5n^5}}$. It follows that $\mathrm{cond}(I-A) = \|I-A\|_\infty \cdot \left\|(I-A)^{-1}\right\|_\infty \leqslant \frac{2n^3}{p_{min}^{5n^5}}$.

Finally, to see that $p_{min}^{2n^3+1} \leqslant \|I-A\|_\infty$, we will show that for every variable x_{uv}, the diagonal entry $(I-A)_{uv,uv} \geqslant p_{min}^{2n^3+1}$. To see this it suffices to note that in the

original cleaned equation $x_{uv} = \alpha$ for a variable $x_{uv} \in X_r$, it cannot be the case that α consists of just one linear term cx_{uv}, because otherwise the LFP of $x_{uv} = cx_{uv}$ is 0, and we have already eliminated 0 variables. Hence, it must be the case that α contains either another linear term $c'x_{st}$ or a constant term c'', or both. In either case, if we plug in the actual LFP values for all other variables besides x_{uv} into α, we will have left an equation of the form $x_{uv} = cx_{uv} + c'$, where, by the arguments of the previous two paragraphs, it must be the case that $c' \geqslant (p_{min})^{2n^3+1}$. Thus, solving for the (unique) solution for x_{uv}, we have $x_{uv} = c'/(1-c) \leqslant 1$. Therefore, $c' \leqslant (1-c)$, and thus $(1-c) \geqslant (p_{min})^{2n^3+1}$. But note that $(1-c)$ is precisely the diagonal entry $(I-A)_{uv,uv}$. Therefore $p_{min}^{2n^3+1} \leqslant \|I-A\|_\infty$. □

For a "higher" linear SCC, X_r, i.e., one which can reach some nonlinear SCC in H, let us define its ***height***, $h_r < \infty$, to be the maximum finite distance in H between X_r and some lower nonlinear SCC that it can reach (again, see Figure 3.6). Let $h_{max} = \max_r h_r$, where the maximum is taken over all linear SCCs that can reach a nonlinear SCC. Note that, as a very loose upper bound, certainly $h_{max} \leqslant n^2$, where $n = |S|$ is the number of control states of the p1CA, because there are at most n^2 variables in the entire system. Now consider the decomposed Newton's method applied to the fixed point equations for a p1CA, with the following specification for the number of iterations to be applied to each SCC:

1. Use, one iteration of Newton's method (starting at vector $x^0 = 0$), or any linear system solving method, to solve a remaining bottom linear SCC exactly. Remove the linear SCC, and plug the corresponding values of variables into equations for higher SCCs. Do this until only nonlinear bottom SCCs remain, or all SCCs are solved.

2. For each remaining nonlinear SCC, apply Newton's method (starting with vector $x^0 = 0$) to the nonlinear equations for these SCCs, using the following number of iterations:

$$4mn^5 + mn^2 + h_{max}(9mn^5 + 4) + i$$

   Afterwards, plug the resulting (approximate) values for variables in each such nonlinear SCC into the equations for higher (linear) SCCs.

3. For each remaining linear SCC, use one iteration of Newton's method (or any other linear system solution method) to solve for the exact (unique) solution of the corresponding linear system (note that the coefficients of these equations will have errors because of the approximations below, but we still seek their exact

solution), then remove the linear SCC, and plug these values into higher (linear) SCCs that remain, until no SCCs remain.

Theorem 3.4.8. *Given a p1CA (or, equivalently, a QBD), the above algorithm, based on (a decomposed) Newton's method, approximates every entry of the matrix* G *of termination probabilities for the p1CA (QBD) to within* i *bits of precision (i.e., to within additive error* $1/2^i$*). In the unit-cost arithmetic RAM model of computation (i.e., rational Blum-Shub-Smale model), the algorithm has a running time which is polynomial in both the encoding size of the p1CA (QBD) and in* i*.*

Proof. First, note that up until the nonlinear SCCs, all values for lower linear SCCs are computed exactly. Next note that, given the number of iterations of Newton's method that are applied in step (2.) of the algorithm for nonlinear SCCs, by Theorem 3.4.2, the values $G_{u,v}$ for variables x_{uv} in nonlinear SCCs are computed to within $W_0 = h_{max}(9mn^5+4)+i$ valid bits of precision. In other words, for each such x_{uv}, a value $G'_{u,v}$ is computed such that $|G_{u,v} - G'_{u,v}|/G_{u,v} \leqslant \frac{1}{2^{W_0}}$. Moreover, since $0 < G_{u,v} \leqslant 1$, we can conclude that $|G_{u,v} - G'_{u,v}| \leqslant \frac{1}{2^{W_0}}$.

Thus, since $W_0 = h_{max}(9mn^5+4)+i \geqslant i$, for all nonlinear SCCs and all linear SCCs which are below them, we certainly do compute $G'_{u,v}$ which approximates the value $G_{u,v}$ for the variables x_{uv} in these SCC, to within at least i bits of precision (i.e., such that $|G_{u,v} - G'_{u,v}| \leqslant 2^i$).

The rest of the proof proceeds by induction on the height, h, of a given higher linear SCC, X_r, above the nonlinear SCCs, to show that for every variable $x_{uv} \in X_r$ we compute $G_{u,v}$ to within $W_h = (h_{max} - h)(9mn^5+4)+i$ bits of precision.

For the base case, $h = 0$, this follows from the fact that all nonlinear SCCs are computed to within $W_0 = h_{max}(9mn^5+4)+i$ bits of precision, and all "lower" linear SCCs are computed exactly.

For the inductive case, let X_r be an "upper" linear SCC in H at height $h > 0$ above nonlinear SCCs, and and suppose that the values of all SCCs below it have been computed to within at least $W_{h-1} = (h_{max} - h + 1)(9mn^5+4)+i$ bits of precision, and plugged into the equations for X_r. We will show that after the linear system associated with X_r has been solved exactly, the solution gives, for each $x_{uv} \in X_r$, a value $G'_{u,v}$ such that $|G_{u,v} - G'_{u,v}| \leqslant \frac{1}{2^{W_h}}$, i.e., such that $G'_{u,v}$ approximates $G_{u,v}$ to within i bits of precision.

To do this, we employ Theorem 3.4.3, which gives us bounds on the errors in solutions of linear systems in terms of condition numbers and other quantities associated

with the linear system, and Lemma 3.4.7, which gives us bounds on these quantities for the specific linear systems that arise for one linear SCC of a p1CA.

Suppose that, if the values of lower SCCs had been computed "exactly" (even though they can be irrational), then the resulting linear system for X_r, which may have irrational coefficients, would be $(I-A)x = b$.

Note that if the values of lower SCCs are approximated to within W_{h-1} bits of precision, then the resulting system can be written as $((I-A)+\mathcal{E})x = (b+\zeta)$. We will now bound the absolute values of entries of $\mathcal{E}$ and ζ.

Note that each entry of the matrix A, e.g., coefficient $a_{uv,st}$ for some variables x_{uv} and x_{st}, is equal to the sum of coefficients of linear expressions that contain x_{st} in the linear equation $x_{uv} = \alpha$. Now, the question is, how much can $a_{uv,st}$ change when the values of lower SCCs are approximated to W_{h-1} bits of precision?

First, let us consider the original quadratic equation $x_{uv} = \alpha'$ before some of the variables $x_{s't'}$ have been substituted by their approximate value. A linear expression containing x_{st} in α' can only result from a monomial term in α of the form px_{st} or $px_{s't'}x_{st}$. In the first case the coefficient p would contribute its exact value to $a_{uv,st}$, so it would add zero to the absolute error of $a_{uv,st}$. However, in the second case, since $p \leqslant 1$ and the value of $x_{s't'}$ is an under-approximation of $G_{s',t'}$ up to W_{h-1} bits of precision, then the coefficient $a_{uv,st}$ could be under-approximated by at most $1/2^{W_{h-1}}$. As we can see, an absolute error of at most $1/2^{W_{h-1}}$ can arise from each such monomial.

Next, we note that the coefficient $a_{uv,st}$ of x_{st} in the equation (3.1) for x_{uv} may actually arise as a sum of at most $n+2$ such monomials:

- ⋆ if $t = v$ (in other words $x_{st} \equiv x_{sv}$) then we can have one of the form: $p^{(0)}_{us}x_{sv}$
- ⋆ if $t = v$ then there can be n monomials for each control state w: $p^{(1)}_{uw}x_{ws}x_{sv}$
- ⋆ if $t = v$ and $s \neq v$ then we can have one extra one like this: $p^{(1)}_{us}x_{sv}x_{vv}$ (if $s = v$ then this expression is counted already for $w = s$ above)
- ⋆ if $t \neq v$ then we get at most one monomial from the expression: $p^{(1)}_{us}x_{st}x_{tv}$

Notice that the sum of all these coefficients is always smaller than 2 since:

$$p^{(0)}_{us} + p^{(1)}_{us}x_{vv} + (\sum_{w} p^{(1)}_{uw}x_{ws}) \leqslant (p^{(0)}_{us} + p^{(1)}_{us}) + \sum_{w} p^{(1)}_{uw} \leqslant 1+1$$

Furthermore, if $n = 1$ then there can be only one SCC. Hence, in such a case $h_{max} = 0$ and we would be done. As a consequence, from now on, we can safely

assume that $n \geqslant 2$ allowing us to make the following estimation: $\mathcal{E}_{uv,st} \leqslant 2/2^{W_{h-1}} \leqslant n/2^{W_{h-1}}$.

We can ask a similar question about b. Since a constant term may arise because both variables in a quadratic monomial of α' belonged to the lower SCCs, we now have that the resulting error $1/2^{W_{h-1}}$ could have arisen for both variables that were fixed in a monomial. It is not hard to see that the resulting error for the entire monomial is at most $2/2^{W_{h-1}}$, basically because such monomials in α' have a coefficient $\leqslant 1$, and because for values $x, x' > 0$, we have $(x-\varepsilon)(x'-\varepsilon) \geqslant xx' - 2\varepsilon$. Thus $\zeta_{uv} \leqslant 2n/2^{W_{h-1}}$. Since the pairs uv and st were arbitrary, and $\mathcal{E}$ is at most an $n^2 \times n^2$ matrix, we have $\|\mathcal{E}\|_\infty \leqslant n^3/2^{W_{h-1}}$, and $\|\zeta\|_\infty \leqslant 2n/2^{W_{h-1}}$.

Therefore, using Lemma 3.4.7, part (1.), we can conclude that $\frac{\|\mathcal{E}\|_\infty}{\|(I-A)\|_\infty} \leqslant \frac{n^3 2^{2mn^3+m}}{2^{W_{h-1}}}$, and also, using Lemma 3.4.7, part (4.), we can conclude that $\frac{\|\zeta\|_\infty}{\|b\|_\infty} \leqslant \frac{2n2^{2mn^3+m}}{2^{W_{h-1}}}$. Next, by Lemma 3.4.7, part (2.), we have $1/\left\|(I-A)^{-1}\right\|_\infty \geqslant 1/(n^2 \cdot 2^{5mn^5})$, and since $\|\mathcal{E}\|_\infty \leqslant n^3/2^{W_{h-1}}$, it is easy to check that $\|\mathcal{E}\|_\infty \leqslant 1/\left\|(I-A)^{-1}\right\|_\infty$. Finally, by Lemma 3.4.7, part (3.), $\mathrm{cond}(I-A) \leqslant 2n^3 \cdot 2^{5mn^5}$.

Now we use these bounds and apply Theorem 3.4.3. Let $\varepsilon = \frac{2n^3 2^{2mn^3+m}}{2^{W_{h-1}}}$, and let $\varepsilon' = 8\varepsilon n^3 \cdot 2^{5mn^5} = \frac{16n^6 2^{2mn^3+m} 2^{5mn^5}}{2^{W_{h-1}}}$. It can be checked that, by construction, the matrix equation $(I-A)x = b$ and its approximate version $(I-A+\mathcal{E})x = (b+\zeta)$, as well as $\|\mathcal{E}\|_\infty$, $\|\zeta\|_\infty$, ε, and ε', all satisfy the conditions of Theorem 3.4.3.

Recall that the unique solution x^* to the original system is $G|_{X_r}$: it consists of those values $G_{u,v}$ where $x_{uv} \in X_r$. Thus in particular $0 < \|x^*\|_\infty \leqslant 1$. Thus, by the conclusion of Theorem 3.4.3, there is a unique solution vector x^*_ε to the approximate system, such that $\|x^*_\varepsilon - x^*\|_\infty \leqslant \varepsilon' = \frac{16n^6 2^{2mn^3+m} 2^{5mn^5}}{2^{W_{h-1}}}$.

The proof of the inductive claim will now be completed by simply checking that $16n^6 2^{2mn^3+m} 2^{5mn^5} \leqslant 2^{2mn^3+m+5mn^5+n^5+4} \leqslant 2^{9mn^5+4}$, and thus since $W_h = (h_{max} - h)(9mn^5+4) + i$, that $\|x^*_\varepsilon - x^*\|_\infty \leqslant \frac{1}{2^{W_h}}$.

The fact that the algorithm has polynomial running time in the unit-cost RAM model follows immediately from the fact that there are only polynomially many iterations of Newton's method, and each iteration essentially involves solving a linear system (or matrix inversion), which can of course be done with polynomially many arithmetic operations (e.g., using Gaussian elimination). □

We emphasize that these (impractical) upper bounds for the number of iterations are very coarse, and are only intended to facilitate our proof that polynomially many iterations of Newton's method suffice. A more detailed analysis would likely yield polynomial bounds with much smaller exponents as the required number of itera-

tions.

3.5 Hardness results for QBD Markov Decision Processes

Quasi-Birth-Death Markov Decision Processes (QBD-MDPs) are QBDs in which the transition from some of the control states are controlled by a player. Since we showed in Chapter 2 that QBDs are equivalent to p1CA, instead of QBD-MDPs we will define here 1-Counter Markov Decision Processes (1C-MDPs) which are a controlled version of p1CAs. A very simple and similar proof to M-equivalence of QBDs and p1CAs can show that QBD-MDPs are M-equivalent to 1C-MDPs.

In this section we will show that the qualitative termination problem for maximizing 1C-MDPs (QBD-MDPs) is DP-hard, and in fact hard for the entire "Boolean Hierarchy". DP is the class of languages which can be composed as the intersection of an NP language and a coNP language. The "Boolean Hierarchy" over NP, denoted BH, is the class of languages that can be formed as "boolean combinations" of NP languages, under the boolean algebra operations of sets: $\{\cap, \cup, \neg\}$. This work is now part of [BBE$^+$09].

Formally, an 1C-MDP A is a tuple (S, δ, δ_0), where the set of control states S is partitioned into a set of probabilistic control states S_0 and a set of controlled ones S_1. For every $s \in S_1$ the transition in δ have the form $(s, \perp, c, s')$. In other words, these transitions are not labeled by probabilities, but are controlled by the player. Otherwise, a 1C-MDP is just like a p1CA and just like a p1CA has a corresponding countable-state Markov Chain, a 1C-MDP corresponds to a single player Simple Stochastic Game defined in an obvious way. We will be focusing here on analyzing the complexity of qualitative (probability 1) termination question about 1CMDPs. Namely, deciding whether starting in a given state $(s, 1)$ does there exist a strategy for the player such that, with probability 1, the 1CMDP terminates (i.e., reaches counter 0), in an accepting control state $F \subseteq S$. Let us notice that this is not the same as asking whether the optimal termination value is 1 or not, since there may exist a sequence of strategies for which the probability of termination is arbitrarily close to 1, but no strategy achieves value 1. We can see such an example in Figure 3.7. A strategy of the maximizer that first increases the counter n times and then moves to state labeled "Lose", reaches the state labeled "Win" with probability $1 - 1/2^n$ before reaching the counter value 0, but no strategy achieves value 1.

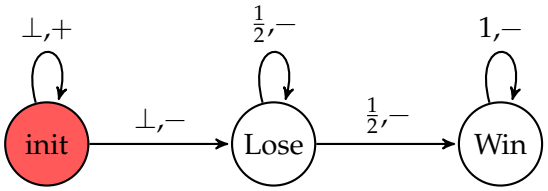

Figure 3.7: An example 1C-MDP, with the objective of maximizing the termination probability at the state "Win", with no optimal strategy, but with optimal (supremum) value 1. A strategy of the maximizer that first increases the counter n times and then moves to the state labeled "Lose", reaches the state labeled "Win" with probability $1 - 1/2^n$ before reaching the counter value 0, but no strategy achieves value 1.

Theorem 3.5.1. *Deciding the existence of an almost sure winning strategy for 1C-MDPs is DP-hard and even BH-hard, i.e., hard for the entire Boolean Hierarchy over NP.*

Proof. Our proof essentially mimics a proof by Serre [Ser06] which shows that the reachability problem for non-probabilistic 2-player 1-counter games is DP-hard. We show that similar arguments work more generally to show BH-hardness (and BH-hardness also applies to the 2-player non-probabilistic games).

First, we will show that the qualitative termination problem for 1C-MDPs is NP-hard and coNP-hard, and then we will show how to combine these to get BH-hardness.

We start with NP-hardness and reduce from SAT. Suppose we have a 3CNF formula : $\psi = C_1 \wedge \ldots \wedge C_m$, over variables $\{x_1, \ldots, x_r\}$. We will encode assignments to the variables of ψ by integers, as follows. Let $\pi_1, \ldots, \pi_r$ denote the first n prime numbers. Then an integer n corresponds to an assignment that assigns true to x_i if and only if $\pi_i | n$. Note that multiple integers map to the same assignment, but that all assignments are certainly mapped to by some integer. Moreover, note that by basic facts from number theory, $\pi_r \leqslant r^2$, thus any assignment can be obtained with a number no greater than $\prod_i \pi_i \leqslant r^{2r}$.

The 1C-MDP will have a start state s_0, which is controlled by the (maximizing) player. The game starts in state $(s_0, 1)$ and the player can choose to increment the counter and stay in state s_0, or to move to state s_1. Thus, after it has repeatedly incremented the counter up to a "guessed" number $n \geqslant 1$, the game moves to state (s_1, n). Control state s_1 is controlled by random, and it "chooses", uniformly at random, one of the clauses C_i, which it claims is not satisfied by the assignment associated with n, and the game moves to state (s'_i, n). State (s'_i, n) is controlled by the maximizing player, and he chooses a literal l_j, in C_i, and moves to state (s'_{i,l_j}, n). Suppose $l_j = x_j$.

From this state we deterministically decrement the counter, but keep track, using π_j auxiliary control states (one for each possible remainder of division by π_j), whether we have decremented a number of times which is dividable by π_j. Clearly, when we hit the counter value 0, we are in a state that indicates whether the original counter value n was $\equiv 0 (\mathrm{mod}\ \pi_j)$ or not, i.e., whether the assignment corresponding to n satisfies clause C_i. We can similarly check, if $l_j = \neg x_j$, that the number of times decremented is $\neq 0 (\mathrm{mod}\ \pi_j)$. Since random chose all clauses with equal probability, there the player has a strategy to terminate in such an "accepting" control states with probability 1 iff there is a satisfying assignment to ψ. It is not hard to see that the size of this 1C-MDP is polynomial in the size of the formula ψ.

Next, for coNP-hardness, suppose we have a 3CNF formula : $\psi = C_1 \wedge \ldots \wedge C_m$, over variables $\{x_1, \ldots, x_r\}$, and we want to check unsatisfiability. We do as before, but with player roles and accepting control states reversed. Starting in state $(s_0, 1)$ we let the probabilistic player with probability $1/2$ increment the counter or with probability $1/2$ move to control state s_1. Thus we reach state (s_1, n) for some $n \geqslant 1$ with probability 1 and any counter value n has nonzero probability of occurring. Next the maximizing player chooses a clause C_i which he thinks cannot be satisfied by the assignment n. Then the random player "picks" one of the literals uniformly at random. We then decrement as before, except that now when we terminate we accept precisely when we would have not accepted before. Specifically, we accept if "assignment" n did not assign true to literal l_j of clause C_i. Since the random player picked one of the literals uniformly at random, if one of the literals was assigned true with a positive probability the game would terminate in a rejecting state. Note that the probability of termination is still 1 (because the probability of incrementing the counter for ever is 0). Thus the maximizer has a strategy such that the probability of termination in an accepting state is 1 if and only if there is no satisfying assignment to ψ.

Finally, to show BH-hardness, consider any statement which is a $\wedge$-$\vee$ combination of statements of the form "ψ_i is satisfiable" and "ψ_j is not-satisfiable", where ψ_i's are boolean formulas. Deciding whether such statements are true is BH-complete.

In order to mimic this with a 1C-MDP, we do as follows: $\vee$ is mimicked by the maximizer picking one of the disjuncts, $\wedge$ is mimicked by the random player picking one of the conjuncts uniformly at random. When we hit a statement "ψ_i is (un)satisfiable", we play the corresponding game. It is not hard to see that the maximizer has a strategy to terminate in an accepting state with probability 1 if and only if the entire statement holds true. □

3.6 Conclusions

In this chapter we showed some of the results that follow as a consequence of equivalences, established in Chapter 2, between TL-QBDs, TS-QBDs on the one hand and RMCs, pPDSs on the other. One of them is the fact that there exists a family of non-strongly connected TL-QBDs which require exponentially many iteration of the decomposed Newton's method to obtain even a single bit of precision, although the classical convergence rate for them is "linear". A natural question to ask is whether there exist a family of strongly connected TL-QBDs (or equivalently RMCs or pPDSs) with the same property.

The main result in this chapter that does not follow from the previous work is the following: in order to approximate the termination probabilities of a QBD (even a non-strongly connected and null-recurrent one) to within $1/2^i$ additive error (absolute error), polynomially many iterations of the decomposed Newton's method suffices both in the size of the QBD, and in i, the number of bits of precision required. In order to derive that bound we established new structural properties of (p)1CAs that may be used in their analysis in the future. The precise bound on the length of the shortest terminating path in (p)1CA was left open as Conjecture 3.3.4. Resolving that conjecture would improve the derived $4mn^5 + mn^2 + h_{max}(9mn^5 + 4) + i$ bound on the number of iterations needed to converge to within $1/2^i$ absolute error. This bound could certainly be improved even further by using more precise estimates on some of the quantities used in our analysis.

Since all the values of the G matrix are less than 1, a stronger result would be to show that a similar polynomial bound exists on the number of iterations needed for the same method to approximate the G matrix to within $1/2^i$ multiplicative factor (relative error, i.e., for all nonzero values in G the ratio between the computed value and the actual value belongs to the interval $(1-1/2^i, 1+1/2^i)$). Unfortunately, the classical result in Theorem 3.4.3 that estimates the error of the solution of a linear system of equations with slightly perturbed coefficients is crucial to our analysis and does not allow us to extend our result in that direction. Using that theorem we can only bound the relative error of the biggest entry of such a solution, while its smallest entry can have potentially arbitrarily high relative error. To overcome this problem we could use a more precise analysis as done in [JE95], that gives an expression that bounds the maximal relative error of each entry of the solution vector to a slightly perturbed linear system. However, so far the coefficients involved in that expression

have not been analyzed precisely and their value can have a crucial impact on whether the resulting bound would be polynomial or not.

On the other hand, a more "practical" remaining open question directly related our result is the following: can polynomial time upper bounds for approximating the G matrix for QBDs be established in the standard Turing model of computation, rather than in the unit-cost rational arithmetic RAM model? It was established in [EY07], that for RMCs and pPDSs, and thus also for Tree-Like QBDs, any non-trivial approximation of the actual termination probabilities of a TL-QBD is at least as hard as the SQRT-SUM problem. Therefore, no such approximation algorithm can be found for TL-QBDs without a major breakthrough in the complexity of exact numerical analysis. However, this does not rule out the possibility of finding such an algorithm for QBDs. In fact, it is entirely plausible that, using the decomposed Newton's method approach, but rounding the computed values after each iteration to a given polynomial number of bits, yields such an approximation algorithm.

All these results lead us to suspect that a similar difference should exist in the worst-case behavior on QBDs and TL-QBDs for numerical methods other than Newton's methods, such as the Logarithmic or Cyclic Reduction algorithms (see, e.g, [BLM05]). We have however not analyzed these algorithms. It is well known that Logarithmic and Cyclic reduction "converge at least linearly" for strongly connected QBDs and TL-QBDs, but as we have already seen, this does not imply any upper bound on the total number of iterations needed as a function of the encoding size of the input QBD and the number of bits of precision required, i. It would be interesting to see whether these two methods can be combined with a decomposition of the equation system into SCCs in order to improve their performance on highly decomposable examples. Such an operation is a major challenge since decomposition of the equation system in general destroys the matrix form of the equations which these two methods are based on.

An even more practical open question is the following: It is known that for QBDs with n control states each iteration of Newton's method can be carried out directly (over $\mathcal{O}(n^2)$ sized matrix equations) with a cost of $\mathcal{O}(n^3)$ operations per iteration (better than the $\mathcal{O}(n^6)$ using, e.g., Gaussian elimination), using the fact that the occurring linear matrix equation at each step of Newton's method is a generalized Sylvester matrix equation system (see [BLM05] for details). However, while TL-QBDs and RMCs also have nonlinear equations with $\mathcal{O}(n^2)$ matrix representations, no such efficient solution method is known for the more general linear matrix equations that occur at

each step of Newton's method. Finding such a method would make Newton's method more practical on large "dense" TL-QBDs, RMCs, and pPDSs. But even with such an efficient method, it remains a challenge to combine it well with decomposition.

We recently showed in [BBE+09] that the problem of deciding whether there exists a strategy that terminates with probability 1 at a given control state of a given QBD-MDP (i.e., 1C-MDP) is in EXPTIME. However, many questions remain open, e.g., the quantitative version of this problem and even deciding whether the supremum probability over all strategies is equal to 1.

Chapter 4

Recursive Simple Stochastic Games with Positive Rewards

4.1 Introduction

In this chapter, motivated by the goal of analyzing the optimal/pessimal expected running time of probabilistic procedural programs, we study the complexity of reward-based stochastic games, 1-exit Recursive Simple Stochastic Games (1-RSSGs) with positive rewards, and their one-player version, 1-exit Recursive Markov Decision Processes (1-RMDPs) with positive rewards.[1] These form a class of (finitely-presented) countable-state turn-based zero-sum stochastic games (and MDPs) with strictly positive rewards, and with an undiscounted expected total reward objective. 1-RSSGs with positive rewards can be equivalently reformulated as Stochastic Context-Free Grammars (SCFGs) games. SCFGs games are essentially an extension of well-known context-free grammars to a probabilistic and controlled setting.[2] 1-RSSGs can also be viewed as a 2-player extension of Branching Markov Decision Chains extensively studied in OR ([Pli76, Vei69, RW82]).

We can see how a probabilistic procedural program can look like on a basic example in Figure 4.1. It represents a function $\mathbf{f}$ that, depending on the value of the parameter n, calls itself recursively with a modified parameter value. Some branches in the program depend on a random choice, and some other are controlled and its is the external controller who decides how the program proceeds from there. His objective could be, e.g., minimizing the number of steps the program makes before terminating.

[1] For their formal definitions and examples, see Section 2.2.1 of Chapter 2.

[2] Their formal definition was presented in Section 2.2.2 and their equivalence to 1-RSSGs was shown in Section 2.3.1, both in Chapter 2.

The 1-exit assumption essentially forbids the functions to return multiple values, and is necessary since we show that even qualitative decision questions about multi-exit RSSGs are undecidable. On the other hand, the assumption of having strictly positive rewards on all transitions is very natural for modeling probabilistic procedural programs since each step of a program takes some nonzero amount of time. Strictly positive rewards also endow our games with a number of important robustness properties. In particular, if defined as a SCFG game, their value does not depend on what derivation law (e.g., left-most or right-most) is imposed. This is not the case if we allow 0 rewards on grammar rules. Even in the 1-player setting, with 0 rewards allowed, the optimal value can be irrational and as it was shown in Section 2.2.2 of Chapter 2, it can be wildly different (e.g., 0 or ∞) depending on the derivation law. Moreover, we do not even know whether qualitative questions about 1-RMDPs with 0 rewards allowed are decidable.

```
void f(int n) {
    while[min] {
        :: n > 1 ->
            :0.5: f(n-1);
            :0.5: n = n/2; f(n);
        :: n > 10 -> f(n-9);
        :: else -> return;
    }
}
```

Figure 4.1: An example probabilistic recursive function with one parameter and not returning any value. *:: cond –>* represents a guarded conditional statement and if there are more than one branches with their guards satisfied at the same time, depending on to whom the statement is assigned we wither choose one of them uniformly at random or the controller chooses how the function proceeds. *:prob: [instruction list];* states that a given list of instructions will be executed with probability *prob* once the program reaches that probabilistic choice point.

For 1-RMDPs and 1-RSSGs with strictly positive rewards, we show that they always have a value which is either rational (with polynomial bit complexity) or ∞, and which arises as the least fixed point solution (over the extended reals) of an associated system of min-max-linear equations. Both players always have *stackless and memoryless (SM)* optimal strategies, i.e., deterministic strategies that depend only on the current

state of the program (e.g., which instruction is being executed and the current values of the variables), but not on how the program got to that state nor the stack contents.

Furthermore, we provide PTIME algorithms for computing the exact value for both the maximizing and minimizing 1-RMDPs with positive rewards. The two cases are not equivalent and require a separate treatment. In both we make use of the fact that linear programming (LP) can be solved in PTIME (see, e.g., [Sch86]). We show that for the 2-player games (1-RSSGs) deciding whether the game has value at least a given $r \in \mathbb{Q} \cup \{\infty\}$ is in NP $\cap$ co-NP. We also describe a practical simultaneous strategy improvement algorithm, analogous to similar algorithms for finite-state stochastic games, and show that it converges to the game value (even if it is ∞) in a finite number of steps. A corollary is that computing the game value and optimal strategies for these games is contained in the class PLS of polynomial local search problems ([JPY88]). Whether this strategy improvement algorithm runs in the worst-case in PTIME is open, just like its version for finite-state SSGs.

We observe that these games are essentially "harder" than Condon's finite-state SSG games in the following senses. We reduce Condon's quantitative decision problem for finite-state SSGs to a special case of 1-RSSG games with strictly positive rewards: namely to deciding whether the game value is ∞. By contrast, if finite-state SSGs are themselves equipped with strictly positive rewards, we can decide in PTIME whether their value is ∞. Moreover, it has recently been shown in [EY07] and [Jub06] that computing the value of Condon's SSG games is in the complexity class PPAD (defined in [Pap94]). The same proof, however, does not work for 1-RSSG with positive rewards, and we do not know whether these games are contained in PPAD. In these senses, the 1-RSSG reward games studied in this chapter appear to be "harder" than Condon's SSGs, and yet as we show their quantitative decision problems remain in NP $\cap$ coNP.

Application to the analysis of expected running time of recursive probabilistic programs in the tool PReMo described in Chapter 5 motivated the work in this chapter. PReMo implements, among other analyses, the strategy improvement algorithm for 1-RSSGs devised in Section 4.3 of this chapter. See Section 5.2.6 of Chapter 5 for implementation issues of that algorithm and encouraging experimental results that compare its performance with some alternative numerical methods.

The rest of this chapter is organized as follows. First, in Section 4.2, we prove that 1-RSSGs with positive rewards are determined and their value is the Least Fixed Point (LFP) of a certain min-max-linear equation system. Next, in Section 4.3, we

prove that both players always have stackless&memoryless optimal strategies and that we can find them via a simultaneous strategy improvement algorithm. In Section 4.4 we show that 1-RMDPs with positive rewards can be solved in PTIME, from which it easily follows that finding the optimal values and optimal strategies for 1-RSSGs is in NP∩coNP and in PLS. As for the lower bounds, we show that qualitative questions about 1-RSSGs with positive rewards are as hard as quantitative questions about Condon's stochastic games, while qualitative questions about multi-exit RSSGs are undecidable. Finally, we conclude in Section 4.5 and present some open problems for future work.

4.2 Determinacy and equation formulation of 1-RSSGs with positive rewards

Let us briefly recall the notation used for defining an 1-RSSG in Section 2.2.1 of Chapter 2. An RSSG A consists of the set of nodes N, the set of boxes B, the box-to-component mapping function Y, the set of component entries En, the set of component exits Ex, the transition function δ, and the box entry to reward mapping ξ. We define additional set of nodes, box call ports $(b, en) \in \mathit{Call}$ and box exit ports $(b, ex) \in \mathit{Ret}$. As pointed out in Section 2.3.1, we can safely assume that the mapping ξ is positive for all call ports, and equal to zero for all component exits and so we can ignore the ✠ state in our analysis. The set of all nodes is denoted by Q and for each $u \in Q$ we let $n(u)$ denote the neighbors of node u, i.e., $n(u) = \{v \mid (u, \bot, v, c_{uv}) \in \delta\}$ if u is Player 1's or Player 2's node and $n(u) = \{v \mid (u, p_{uv}, v, c_{uv}) \in \delta\}$ otherwise. In a (1-)RSSG with positive rewards the goal of Player 1 (the ***maximizer***) is to maximize the total expected reward gained during a play of the game, and the goal of Player 2 (the ***minimizer***) is to minimize this. We commonly denote the strategy of the maximizer as σ and the minimizer as τ. An RSSG A along with players strategies σ and τ define in a natural way a Markov Chain with (positive) rewards $M_A^{\sigma,\tau}$.

Let $r_u^{k,\sigma,\tau}$ denote the expected reward during the first k steps in $M_A^{\sigma,\tau}$, starting at initial state $\langle \varepsilon, u \rangle$. Given an initial vertex u, let $r_u^{*,\sigma,\tau} = \lim_{k\to\infty} r^{k,\sigma,\tau}$ denote the total expected reward obtained in a run of $M_A^{\sigma,\tau}$, starting at initial state $\langle \varepsilon, u \rangle$. Clearly, this sum may diverge, thus $r^{*,\sigma,\tau} \in [0, \infty]$. Note that, because of the constraint of all the rewards being strictly positive, this sum will be finite if and only if the expected number of steps until a play terminates is finite.

In ([EY05i]) a monotone system of min-max-*polynomial* equations was defined whose

Least Fixed Point (LFP) solution yields the values of termination 1-RSSGs (games with termination probability objective). We show here that we can adapt this to obtain analogous min-max-*linear* systems in the setting of 1-RSSGs with positive rewards. Furthermore, these systems generalize the linear Bellman's equations for MDPs [Bel57]. We use a variable x_u for each unknown r^*_u. Let $\mathbf{x}$ be the vector of all x_u-s, $u \in Q$. The system has one equation of the form $x_u = P_u(\mathbf{x})$ for each vertex u. Suppose that u is a node inside component A_i whose (unique) exit is ex. There are 5 cases based on the "*Type*" of u:

1. $\mathtt{Type}_0$: $u = ex$. In this case: $x_u = 0$.
2. $\mathtt{Type}_{\text{rand}}$: $\mathrm{pl}(u) = 0$ & $u \in (N_i \setminus \{ex\}) \cup Ret^i$: $x_u = \sum_{v \in n(u)} p_{uv}(x_v + c_{uv})$. (If u has no outgoing transitions, this equation is by definition $x_u = 0$, and similarly for $\mathtt{Type}_{\max}$ and $\mathtt{Type}_{\min}$.)
3. $\mathtt{Type}_{\text{call}}$: $u = (b, en)$ is a call port: $x_{(b,en)} = x_{en} + x_{(b,ex')} + c_u$, where $ex' \in Ex_{Y(b)}$ is the unique exit of $A_{Y(b)}$. [3]
4. $\mathtt{Type}_{\max}$: $\mathrm{pl}(u) = 1$ and $u \in (N_i \setminus \{ex\}) \cup Ret^i$: $x_u = \max_{v \in n(u)}(x_v + c_{uv})$
5. $\mathtt{Type}_{\min}$: $\mathrm{pl}(u) = 2$ and $u \in (N_i \setminus \{ex\}) \cup Ret^i$: $x_u = \min_{v \in n(u)}(x_v + c_{uv})$

We can represent this system of equations in a vector notation as $\mathbf{x} = P(\mathbf{x})$. Given a 1-RSSG, we can easily construct its associated system in linear time. For vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leqslant \mathbf{y}$ means $x_j \leqslant y_j$ for every j. Let $\mathbf{r}^* \in \mathbb{R}^n$ denote the n-vector of r^*_u's. Let $\mathbf{0}$ denote an all 0 vector, and define $\mathbf{x}^0 = \mathbf{0}$, $\mathbf{x}^{k+1} = P^{k+1}(\mathbf{0}) = P(\mathbf{x}^k)$, for $k \geqslant 0$.

Theorem 4.2.1.

1. *The map* $P : \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^n$ *is monotone on* $\mathbb{R}^\infty_{\geqslant 0}$ *and* $\mathbf{0} \leqslant \mathbf{x}^k \leqslant \mathbf{x}^{k+1}$ *for* $k \geqslant 0$.

2. $\mathbf{r}^* = P(\mathbf{r}^*)$.

3. *For all* $k \geqslant 0$, $\mathbf{x}^k \leqslant \mathbf{r}^*$.

4. *For all* $\mathbf{r}' \in \mathbb{R}^\infty_{\geqslant 0}$, *if* $\mathbf{r}' = P(\mathbf{r}')$, *then* $\mathbf{r}^* \leqslant \mathbf{r}'$.

5. *For all vertices* u,

$$r^*_u \doteq \sup_{\sigma \in \Psi_1} \inf_{\tau \in \Psi_2} r^{*,\sigma,\tau}_u = \inf_{\tau \in \Psi_2} \sup_{\sigma \in \Psi_1} r^{*,\sigma,\tau}_u.$$

(In other words, these games are determined.)

[3] Recall that $c_u = \xi(u)$ is positive for convenience. As it was shown in Section 2.3.1 of Chapter 2, any 1-RSSG with some $c_u = 0$ has an essentially equivalent 1-RSSG with all $c_u > 0$.

6. $\mathbf{r}^* = \lim_{k\to\infty} \mathbf{x}^k$.

Proof.

1. All equations in the system $P(x)$ are min-max linear with non-negative coefficients and constants, and hence are monotone.

2. The proof that $\mathbf{r}^* = P(\mathbf{r}^*)$ is similar to the one for 1-RSSG termination games from [EY05i], but it uses in a crucial way the fact that rewards on all transitions are strictly positive.

   (a) For $u = ex \in Type_0$, $\mathbf{r}^*_u = 0$, so it fulfills the corresponding equation $x_u = 0$.

   (b) For $u \in Type_{rand}$, from the definition $r^*_u = \sup_{\sigma\in\Psi_1} \inf_{\tau\in\Psi_2} r^{*,\sigma,\tau}_u$ it follows that $\mathbf{r}^*_u = \sum_{v\in n(u)} p_{uv}(\mathbf{r}^*_v + c_{uv})$. Note that this holds even when some of the expected rewards are infinite, because if $p_{uv} > 0$ and the game starting at v has infinite reward value, then this is also the case starting at u.

   (c) For $u \in Type_{call}$, $u = (b, en)$ is a call port. We claim that

   $$\mathbf{r}^*_u = \mathbf{r}^*_{en} + \mathbf{r}^*_{(b,ex')} + c_u \tag{4.1}$$

   where ex' is the unique exit of $Y(b)$. For this we make crucial use of the assumption that rewards on all transitions are strictly positive[4]. Consider the game starting at $u = (b, en)$, as a combination of two games: the two players play inside b, starting at en, with Player 1's goal to maximize the total (expected) reward. The two players also (in a "separate" game) play starting at (b, en). The payoff to Player 1 is as follows: If the game inside b terminates, then the payoff is the total of the payoffs gained in both games, and if the game inside b does not terminate, then the payoff is just the payoff gained inside b.

   It should be clear that this "modified" version of the game in fact describes the same game. In particular, in the original game both players can, upon first encountering (b, ex') (in the empty context) safely ignore the history and try to maximize/minimize the payoff in the game starting at (b, ex'), without changing the reward value.

   Fix strategies for both players. What is the expected total reward starting at u? It is c_u plus the expected reward gained inside box b, plus the expected

[4]We note that this assumption would be unnecessary if we were working with SCFG games with simultaneous expansion (Sections 2.2.2 and 2.2.3 of Chapter 2). The entire proof would go through for such games even with 0 rewards on rules.

reward after exiting box b *times the probability of exiting box* b. The key point is that, since all transitions have positive reward, the only circumstance under which the expected reward value within box b is finite, i.e., $\mathbf{r}^*_{en} < \infty$, is when for every strategy of the maximizer there is a strategy for the minimizer that assures finite expected reward inside b. This also necessarily assures that box b is exited with probability 1 (because otherwise, since all transitions have positive reward bounded below by some minimum value $c > 0$, infinite expected reward would be gained inside b). Consequently, equality (4.1) holds when $\mathbf{r}^*_{en} < \infty$. But if $\mathbf{r}^*_{en} = \infty$, then the equality holds regardless of the value of $\mathbf{r}^*_{(b,ex')}$, so it holds in all circumstances.

(d) For $u \in Type_{max}$, we know that $\mathbf{r}^*_u \geqslant \mathbf{r}^*_v + c_{uv}$ for any $v \in n(u)$, for otherwise the maximizer would be better off taking the transition to node v in the first step, and thereafter obtaining $\mathbf{r}^*_v + c_{uv}$. On the other hand we also have that $\mathbf{r}^*_u \leqslant \mathbf{r}^*_v + c_{uv}$ for some $v \in n(u)$, as otherwise no matter what first transition the maximizer picks from u, the minimizer has a strategy such that the maximizer will not be able to obtain expected reward $\mathbf{r}^*_u$.

(e) For $u \in Type_{min}$ we know that $\mathbf{r}^*_u \leqslant \mathbf{r}^*_v + c_{uv}$ for all $v \in n(u)$, as otherwise it would be better for the minimizer to take the transition leading to node v and giving to the maximizer expected reward $\mathbf{r}^*_v + c_{uv}$ that is lower than $\mathbf{r}^*_u$. However, we also have to have that $\mathbf{r}^* \geqslant \mathbf{r}^*_v + c_{uv}$ for some $v \in n(u)$, because otherwise player the maximizer could always obtain expected reward higher than $\mathbf{r}^*_u$ no matter what the minimizer does.

3. Note that P is monotonic, and $\mathbf{r}^*$ is a fixed point of P. Since $x^0 = \mathbf{0} \leqslant \mathbf{r}^*$, it follows by induction on k that $\mathbf{x}^k \leqslant \mathbf{r}^*$, for all $k \geqslant 0$.

4. Consider any fixed point $\mathbf{r}'$ of the equation system $P(\mathbf{x})$. We will prove that $\mathbf{r}^* \leqslant \mathbf{r}'$. Let us denote by τ^* a strategy for the minimizer that picks for each vertex the successor with the minimum value in $\mathbf{r}'$, i.e., for each state $s = \langle \beta, u \rangle$, where u is one of the minimizer's nodes, we choose $\tau^*(s) = \arg\min_{v \in n(u)} \mathbf{r}'_v$ (breaking ties lexicographically).

Lemma 4.2.2. *For all strategies* $\sigma \in \Psi_1$ *of Player 1, and for all* $k \geqslant 0$, $\mathbf{r}^{k,\sigma,\tau^*} \leqslant \mathbf{r}'$.

Proof. Base case $\mathbf{r}^{0,\sigma,\tau^*} = \mathbf{0} \leqslant \mathbf{r}'$ is trivial.

(a) $u = ex$, then $\mathbf{r}^{k,\sigma,\tau^*}_u = 0 = \mathbf{r}'_u$ for all $k \geqslant 0$.

(b) $u \in \mathrm{Type}_{\mathrm{rand}}$ is a random node and after we define a strategy $\sigma'(\theta) = \sigma(\langle \varepsilon, u \rangle \theta)$ we get:

$$\mathbf{r}_u^{k+1,\sigma,\tau^*} = \sum_{v \in n(u)} p_{uv}(\mathbf{r}_v^{k,\sigma',\tau^*} + c_{uv}) \leqslant \sum_{v \in n(u)} p_{uv}(\mathbf{r}'_v + c_{uv}) = \mathbf{r}'_u$$

by the inductive assumption and the fact that $\mathbf{r}'$ is a fixed point of $P(\mathbf{x})$.

(c) If $u = (b, en)$ is an entry en of the box b then we claim

$$\mathbf{r}_u^{k+1,\sigma,\tau^*} \leqslant \max_{\rho \in \Psi_1} \mathbf{r}_{en}^{k,\rho,\tau^*} + \max_{\rho \in \Psi_1} \mathbf{r}_{(b,ex')}^{k,\rho,\tau^*} + c_u \tag{4.2}$$

where (b, ex') is the only return port of box b. To see this, note that in any specific trajectory, the total reward gained in $k+1$ steps starting at call port (b, en) is c_u plus the remaining reward, which is split into two parts: that gained in i steps inside box b, and the rest gained in j steps after returning from box b, and such that $i + j = k$. Thus clearly the total expected reward in $k+1$ steps starting at u is no more than c_u plus the expected reward in k steps starting inside box b (i.e., starting at the entry en of $Y(b)$) plus the expected gain in k steps starting at (b, ex'). We now have

$$\max_{\rho \in \Psi_1} \mathbf{r}_{en}^{k,\rho,\tau^*} + \max_{\rho \in \Psi_1} \mathbf{r}_{(b,ex')}^{k,\rho,\tau^*} + c_u \leqslant \mathbf{r}'_{en} + \mathbf{r}'_{(b,ex')} + c_u = \mathbf{r}'_u \tag{4.3}$$

by the inductive assumption, and by the fact that $\mathbf{r}'$ is a fixed point of $P(\mathbf{x})$. So, combining equations (4.2) and (4.3), we have $\mathbf{r}_u^{k+1,\sigma,\tau^*} \leqslant \mathbf{r}'_u$.

(d) For $u \in \mathrm{Type}_{\max}$ we claim

$$\mathbf{r}_u^{k+1,\sigma,\tau^*} \leqslant \max_{v \in n(u)} \mathbf{r}_v^{k,\sigma',\tau^*} + c_{uv}$$

because the player has to move to some neighbor v of $\langle \varepsilon, u \rangle$ in one step, and thus it cannot gain more that $\mathbf{r}^{k,\sigma',\tau^*}$, where σ' is defined from σ in the same way as for $\mathrm{Type}_{\mathrm{rand}}$. Thus

$$\mathbf{r}_u^{k+1,\sigma,\tau^*} \leqslant \max_{v \in n(u)} \mathbf{r}_v^{k,\sigma',\tau^*} + c_{uv} \leqslant \max_{v \in n(u)} \mathbf{r}'_v + c_{uv} = \mathbf{r}'_u$$

(e) For $u \in \mathrm{Type}_{\min}$ we know that $\tau^*(u) = \arg\min_{v \in n(u)}(\mathbf{r}'_u + c_{uv}) = v^*$, so:

$$\mathbf{r}_u^{k+1,\sigma,\tau^*} = \mathbf{r}_{v^*}^{k,\sigma',\tau^*} + c_{uv^*} \leqslant \mathbf{r}'_{v^*} + c_{uv^*} = \min_{v \in n(u)} (\mathbf{r}'_v + c_{uv}) = \mathbf{r}'_u$$

□

Now by the lemma we have $\mathbf{r}_u^{*,\sigma,\tau^*} = \lim_{k \to \infty} \mathbf{r}_u^{k,\sigma,\tau^*} \leqslant \mathbf{r}'_u$ for every vertex u and for any maximizer's strategy σ, so $\sup_{\sigma \in \Psi_1} \mathbf{r}_u^{*,\sigma,\tau^*} \leqslant \mathbf{r}'_u$. Thus for all vertices u:

$$\mathbf{r}_u^* = \sup_{\sigma \in \Psi_1} \inf_{\tau \in \Psi_2} \mathbf{r}_u^{*,\sigma,\tau} \leqslant \inf_{\tau \in \Psi_2} \sup_{\sigma \in \Psi_1} \mathbf{r}_u^{*,\sigma,\tau} \leqslant \sup_{\sigma \in \Psi_1} \mathbf{r}_u^{*,\sigma,\tau^*} \leqslant \mathbf{r}'_u \tag{4.4}$$

5. In equation (4.4) above, choose $\mathbf{r}' = \mathbf{r}^*$. Then for all vertices u we have

$$\sup_{\sigma \in \Psi_1} \inf_{\tau \in \Psi_2} \mathbf{r}_u^{*,\sigma,\tau} = \inf_{\tau \in \Psi_2} \sup_{\sigma \in \Psi_1} \mathbf{r}_u^{*,\sigma,\tau}.$$

6. We know that $\mathbf{z} = \lim_{k\to\infty} \mathbf{x}^k$ exists in $[0,\infty]$, because it is a monotonically non-decreasing sequence (note some entries may be infinite). In fact we have $\mathbf{z} = \lim_{k\to\infty} P^{k+1}(\mathbf{0}) = P(\lim_{k\to\infty} P^k(\mathbf{0}))$, and thus $\mathbf{z}$ is a fixed point of the equation $P(\mathbf{x}) = \mathbf{x}$. So from (4) we have $\mathbf{r}^* \leqslant \lim_{k\to\infty} \mathbf{x}^k$. Since $\mathbf{x}^k \leqslant \mathbf{r}^*$ for all $k \geqslant 0$, $\lim_{k\to\infty} \mathbf{x}^k \leqslant \mathbf{r}^*$ and thus $\lim_{k\to\infty} \mathbf{x}^k = \mathbf{r}^*$.

□

The following is a simple corollary of the proof.

Corollary 4.2.3. *In 1-RSSG positive reward games, the minimizer has an optimal deterministic Stackless and Memoryless (SM) strategy.*

Proof. It is enough to consider the strategy τ^*, from Part 4 of Theorem 4.2.1, when we let $\mathbf{r}' = \mathbf{r}^*$. For then, by equation (1), we have $r_u^* = \sup_{\sigma\in\Psi_1} r^{*,\sigma,\tau^*} = \inf_{\tau\in\Psi_2} \sup_{\sigma\in\Psi_1} r^{*,\sigma,\tau^*}$.

□

Note that for a 1-RMC (i.e., without players) with positive rewards, the vector $\mathbf{r}^*$ of expected total rewards is the LFP of a system $x = Ax + b$, for some non-negative matrix $A \in \mathbb{R}^{n\times n}$, $A \geqslant 0$, and a positive vector $b > 0$. The following will be useful later.[5]

Lemma 4.2.4. *For any $x \in \mathbb{R}^n_{\geqslant 0}$, $A \in (\mathbb{R}^\infty_{\geqslant 0})^{n\times n}$ and $b \in (\mathbb{R}^\infty_{>0})^n$, if $x \leqslant Ax + b$ then $x \leqslant (\sum_{k=0}^\infty A^k)b$. This holds even if for some indices i we have $b_i = 0$, as long as the entries in any such row i of the matrix A are all zero.*

Proof. Let $D = \sum_{k=0}^\infty A^k$ and $y = Db$. We have to prove that $x \leqslant y$. Some of the entries of D can be infinite. Let $R = \{r_1, r_2, \ldots, r_m\}$ be the set of indices of the rows of D that contain at least one ∞ entry. For every $r \in R$, $y_r = \sum_{i=1}^n D_{r,i} b_i$. Since for all i, $b_i > 0$, and for at least one i entry $D_{r,i}$ is ∞, we have $y_r = \infty$ and so $x_r \leqslant y_r$ is trivially fulfilled for every $r \in R$. Now let us construct a new matrix A' by zeroing all the entries of the rows of A that are in R. Similarly let x' be a vector x with zeroed entries x_r where $r \in R$. Let $D' = \sum_{k=0}^\infty (A')^k$.

[5] Note that if we assume both that $A \in (\mathbb{R}_{\geqslant 0})^{n\times n}$ and that $(\sum_{k=0}^\infty A^k)$ converges, the lemma is trivial: we have $(I-A)^{-1} = (\sum_{k=0}^\infty A^k)$, and thus $x \leqslant Ax + b \Rightarrow x - Ax \leqslant b \Rightarrow (I-A)x \leqslant b \Rightarrow x \leqslant (I-A)^{-1}b$. But we need this lemma even when $(\sum_{k=0}^\infty A^k)$ is not convergent.

We will prove that $x' \leqslant A'x' + b$. For entries $r \in R$, it is trivial since $(A'x')_r + b_r = 0 + b_r \geqslant 0 = x'_r$. If $r \notin R$ then $x'_r = x_r$ and

$$(A'x')_r = \sum_{i=1}^{n} A'_{r,i}x'_i = \sum_{\{i|A'_{r,i}>0\}} A'_{r,i}x'_i$$

Proposition 4.2.5. *If* $A_{i,j} > 0$, *and for some* k *we have that* $D_{j,k} = \infty$ *then* $D_{i,k} = \infty$.

Proof. We have that $D = I + AD$ and so $D_{i,k} = \delta_{ik} + \sum_{l=1}^{n} A_{i,l}D_{l,k} \geqslant A_{i,j}D_{j,k} = \infty$. (where δ_{ik} is equal to 1 if $i = k$ and 0 otherwise) □

Suppose that $r \notin R$. If for some i, $x'_i \neq x_i$, then $i \in R$ and we must have $D_{i,j} = \infty$ for some j. If $A'_{r,i} > 0$ then $A_{r,i} = A'_{r,i}$, and by Proposition 4.2.5 we get that $D_{r,j} = \infty$, which contradicts the fact that $r \notin R$. Thus for $r \notin R$, and for i such that $A'_{r,i} > 0$, we must have $x'_i = x_i$ and $A'_{r,i} = A_{r,i}$. Thus $(A'x')_r + b_r = (Ax)_r + b_r \geqslant x_r = x'_r$ for all $r \notin R$. Hence we can conclude that $x'_r \leqslant (A'x')_r + b_r$ for all r.

We will now prove that $\lim_{k\to\infty}(A')^k = 0$. For contradiction, note that if we had $\lim_{k\to\infty}((A')^k)_{i,j} \neq 0$ for some i and j, then it must be the case that $D'_{i,j} = \infty$ because $((A')^k)_{i,j} \geqslant 0$ for all k, and if there is some $\varepsilon > 0$ such that for infinitely many k, $(A')^k_{i,j} > \varepsilon$, then $D'_{i,j} = \infty$. Since $A' \leqslant A$, we get that $(A')^k \leqslant A^k$ for any $k \geqslant 0$ and thus $\sum_{k=0}^{\infty}(A')^k \leqslant \sum_{k=0}^{\infty}A^k$. Thus if $D'_{i,j} = \infty$ then $D_{i,j} = \infty$. Hence, when obtaining A' from A all the entries in the i-th row were zeroed. However, if the i-th row in A' has all zeroes, then so does the i-th row in $(A')^k$ for any k. That contradicts the assumption that $\lim_{k\to\infty}((A')^k)_{i,j} \neq 0$.

By substituting x' by $A'x' + b$ in $x' \leqslant A'x' + b$, we get that $x' \leqslant A'x' + b \leqslant A'(A'x' + b) + b = (A')^2x' + A'b + b \leqslant (A')^2(A'x' + b) + A'b + b = (A')^3x' + ((A')^2 + A' + I)b$ and by iterating we see that $x' \leqslant (A')^{l+1}x' + (\sum_{k=0}^{l}(A')^k)b$ for any $l \geqslant 0$. As x' is a vector of finite values and $\lim_{k\to\infty}(A')^k = 0$ we have $x' \leqslant (\sum_{k=0}^{\infty}(A')^k)b$ and so also $x' \leqslant (\sum_{k=0}^{\infty}A^k)b = y$. The last fact proves that for $r \notin R$ $x_r = x'_r \leqslant y_r$, and we can finally conclude that $x \leqslant y = (\sum_{k=0}^{\infty}A^k)b$.

Now we show that we can also handle the case when for some indices i, $b_i = 0$. We proceed by induction on the number, d, of indices i such that $b_i = 0$ and the i-th row of A contains only zeroes. For the base case $d = 0$, the claim was already proved. For the inductive case, suppose $d > 0$, and let i be the smallest such index. Since we assume $Ax + b \geqslant x$, it must be that $x_i = 0$. Let M' denote the matrix obtained by removing the i-th row and the i-th column in some matrix M. Similarly for a vector v by v' denote the vector v with removed i-th entry. Since $x_i = 0$, $M'x' = (Mx)'$ for any matrix M. Also, since the i-th row of A contains only zeroes we have that

$(A')^k = (A^k)'$ for any $k \geqslant 0$ and we can also conclude that $\sum_{k=0}^{\infty}(A')^k = (\sum_{k=0}^{\infty} A^k)'$. Now assuming $Ax + b \geqslant x$ we can see that $(Ax+b)' \geqslant x'$ and so $A'x' + b' \geqslant x'$. But it is easy to confirm that A' and b' have the same property as before: if $b'_j = 0$ then the j-th row of A' contains only zeroes. Moreover, there are now $d-1$ such indices. Thus, by inductive hypothesis, $x' \leqslant (\sum_{k=0}^{\infty}(A')^k)b' = (\sum_{k=0}^{\infty} A^k)'b' = ((\sum_{k=0}^{\infty} A^k)b)'$, and since the inequality is trivial for the i-th position of x, we get that $x \leqslant (\sum_{k=0}^{\infty} A^k)b$. □

4.3 SM-determinacy and strategy improvement

We now prove SM-determinacy, and also show that strategy improvement can be used to compute the values and optimal strategies for 1-RSSG positive reward games. Consider the following ***(simultaneous) strategy improvement*** algorithm.

Initialization: Pick some (any) SM strategy, σ, for Player 1 (the maximizer).
Iteration step: First, compute the optimal value, $r_u^{*,\sigma}$, starting from every vertex, u, in the resulting minimizing 1-RMDP. (We show in Theorem 4.4.1 that this can be done in PTIME.) Then, update σ to a new SM strategy, σ', as follows. For each vertex $u \in \mathrm{Type}_{\max}$, if $\sigma(u) = v$ and u has a neighbor $w \neq v$, such that $r_w^{*,\sigma} + c_{uw} > r_v^{*,\sigma} + c_{uv}$, let $\sigma'(u) := w$ (e.g., choose a w that maximizes $r_w^{*,\sigma} + c_{uw}$). Otherwise, let $\sigma'(u) := \sigma(u)$. Repeat the iteration step, using the new σ' in place of σ, until no further local improvement is possible, i.e., stop when $\sigma' = \sigma$.

Theorem 4.3.1 shows that this algorithm always halts, and produces an optimal final SM strategy for Player 1. (The proof shows it works even if we switch any non-empty subset of improvable vertices in each iteration.) Combined with Corollary 4.2.3, both players have optimal SM strategies, i.e., the games are SM-determined.

Theorem 4.3.1. *(1)* SM-determinacy. *In 1-RSSG positive reward games, both players have optimal SM strategies. (and thus by Corollary 4.2.3 these games are SM determined). (2)* Strategy Improvement. *Moreover, we can compute the value and optimal SM strategies using the above simultaneous strategy improvement algorithm. (3) Computing the value and optimal strategies in these games is contained in the class PLS.*

Proof. Let σ be any SM strategy for Player 1. Consider $\mathbf{r}_u^{*,\sigma} = \inf_{\tau \in \Psi_2} \mathbf{r}_u^{*,\sigma,\tau}$. (Note that some entries in the vector $\mathbf{r}^{*,\sigma}$ may be ∞.) First, note that if $\mathbf{r}^{*,\sigma} = P(\mathbf{r}^{*,\sigma})$ then $\mathbf{r}^{*,\sigma} = \mathbf{r}^*$. This is because, by Theorem 4.2.1, $\mathbf{r}^* \leqslant \mathbf{r}^{*,\sigma}$, and on the other hand, σ is just one strategy for Player 1, and for every vertex u, $\mathbf{r}_u^* = \sup_{\sigma' \in \Psi_1} \mathbf{r}_u^{*,\sigma'} \geqslant \mathbf{r}_u^{*,\sigma}$. Now we claim that, for all vertices u such that $u \notin \mathrm{Type}_{\max}$, $\mathbf{r}_u^{*,\sigma}$ satisfies its equation in

$\mathbf{x} = P(\mathbf{x})$. In other words, $\mathbf{r}_u^{*,\sigma} = P_u(\mathbf{r}^{*,\sigma})$. To see this, note that for vertices u of $\mathtt{Type}_0$, $\mathtt{Type}_{\text{call}}$ and $\mathtt{Type}_{\text{rand}}$, no choice of either player is involved and the equation holds by definition of $\mathbf{r}^{*,\sigma}$. (In particular, the expected reward value at a call port u is c_u plus the sum of the expected reward values of the game starting at the entry inside the box, and the game starting at the return port.) For nodes $u \in \mathtt{Type}_{\min}$, we have the equation $\mathbf{x}_u = \min_{v \in n(u)} \mathbf{x}_v + c_{uv}$. But note that the best the minimizer can do against strategy σ, starting at $\langle \varepsilon, u \rangle$, is to move to a neighboring vertex v such that $v = \arg\min_{v \in n(u)}(\mathbf{r}_v^{*,\sigma} + c_{uv})$. Thus, the only equations that may fail are those for $u \in \mathtt{Type}_{\max}$, $\mathbf{x}_u = \max_{v \in n(u)}(\mathbf{x}_v + c_{uv})$. Suppose $\sigma(u) = v$, for some neighbor v. Clearly then, $\mathbf{r}_u^{*,\sigma} = \mathbf{r}_v^{*,\sigma} + c_{uv}$. Thus, $\mathbf{r}_u^{*,\sigma} \leqslant \max_{v' \in n(u)}(\mathbf{r}_{v'}^{*,\sigma} + c_{uv'})$. Thus equality fails iff there is another vertex $w \neq v$, with $(u, \perp, w) \in \delta$, such that $\mathbf{r}_v^{*,\sigma} + c_{uv} < \mathbf{r}_w^{*,\sigma} + c_{uw}$.

Suppose now that the nodes $(u_1, u_2, \ldots u_m)$ are all those nodes where the SM strategy σ is not locally optimal, i.e., for $i = 1, 2, \ldots, m$, $\sigma(u_i) = v_i$, and thus $\mathbf{r}_{u_i}^{*,\sigma} = \mathbf{r}_{v_i}^{*,\sigma} + c_{u_i v_i}$, but there is some w_i such that $\mathbf{r}_{v_i}^{*,\sigma} + c_{u_i v_i} < \mathbf{r}_{w_i}^{*,\sigma} + c_{u_i w_i}$. Let $\mathbf{u} = (u_1, u_2, \ldots, u_m)$ and similarly define $\mathbf{v}$ and $\mathbf{w}$. Consider now a revised SM strategy σ', which is identical to σ, except that $\sigma'(u_i) = w_i$ for all i. Next, consider a parametrized 1-exit RSSG, $A(\mathbf{t})$ where $\mathbf{t} = (t_1, t_2, \ldots, t_m)$, which is identical to A, except that all edges out of vertices u_i are removed, and replaced by a single probability 1 edge labeled by reward t_i, to the exit of the same component node u_i is in. Fixing the value of the vector $\mathbf{t} \in [0, \infty]^m$ determines an 1-RSSG, $A(\mathbf{t})$. Note that if we restrict SM strategies σ or σ' to vertices other than those in $\mathbf{u}$, then they both define the same SM strategy for the 1-RSSG $A(\mathbf{t})$. Define $r_z^{*,\sigma,\tau,\mathbf{t}}$ to be the expected total reward starting from $\langle \varepsilon, z \rangle$ in the Markov chain $M_{A(\mathbf{t})}^{z,\sigma,\tau}$. Now, for each vertex z, define the function $f_z(\mathbf{t}) = \inf_{\tau \in \Psi_2} r_z^{*,\sigma,\tau,\mathbf{t}}$. In other words, $f_z(\mathbf{t})$ is the infimum of the expected rewards, over all strategies of Player 2, starting at $\langle \varepsilon, z \rangle$ in $A(\mathbf{t})$. This reward is parametrized by $\mathbf{t}$. Now, let $\mathbf{t}^\sigma$ be a vector such that $\mathbf{t}_{u_i}^\sigma = \mathbf{r}_{u_i}^{*,\sigma}$, and observe that $f_z(\mathbf{t}^\sigma) = \mathbf{r}_z^{*,\sigma}$ for every z. This is so because any strategy for minimizing the total reward starting from z would, upon hitting a state $\langle \beta, u_i \rangle$ in some arbitrary context β, be best off minimizing the total expected reward starting from $\langle \beta, u \rangle$ until that context is exited, i.e., until reaching $\langle \beta, ex' \rangle$ (and unless the minimizer has a strategy that assures the context is exited with probability 1, the expected reward will be ∞).

Note that, by Corollary 4.2.3, in the 1-RSSG reward game on $A(\mathbf{t})$, for any values in vector $\mathbf{t}$, and any start vertex z, the minimizer has an optimal SM strategy $\tau_{z,\mathbf{t}}$, such that $\tau_{z,\mathbf{t}} = \arg\min_{\tau \in \Psi_2} r_z^{*,\sigma,\tau,\mathbf{t}}$. Let $g_{(z,\tau)}(\mathbf{t}) = r_z^{*,\sigma,\tau,\mathbf{t}}$. Note that $f_z(\mathbf{t}) = \min_\tau g_{z,\tau}(\mathbf{t})$, where the minimum is over SM strategies. Now, note that the function $g_{z,\tau}(\mathbf{t})$ is the

expected reward in a positive reward 1-RMC starting from a particular vertex, and it is given by $g_{z,\tau}(\mathbf{t}) = (\lim_{k\to\infty} R^k(\mathbf{0}))_z$ for a linear system $\mathbf{x} = R(\mathbf{x})$ with non-negative coefficients in R, where $R(\mathbf{x}) = A_{\sigma,\tau}\mathbf{x} + b_{\sigma,\tau}(\mathbf{t})$, for some nonnegative matrix $A_{\sigma,\tau}$, and vector $b_{\sigma,\tau}(\mathbf{t})$ which describes the average 1-step rewards from each vertex. All of these 1-step rewards are positive, except that at positions u_i the entry is the variable t_i, i.e., $b_{u_i}(\mathbf{t}) = t_i$. (Note that for all i the u_i'th row vector of $A_{\sigma,\tau}$ is all zero.) Simple iteration then shows that $g_{z,\tau}(\mathbf{t}) = \lim_{k\to\infty} R^k(\mathbf{0})_z = ((\sum_{k=0}^{\infty} A^k_{\sigma,\tau})b(\mathbf{t}))_z$. (Note that if $\lim_{k\to\infty} A^k_{\sigma,\tau} = 0$, then $(\sum_{k=0}^{\infty} A^k_{\sigma,\tau}) = (I - A_{\sigma,\tau})^{-1}$.) Now $g_{z,\tau}(\mathbf{t})$ has the following properties: it is a continuous, nondecreasing, and linear function of $\mathbf{t} \in [0,\infty]^m$, and for $\mathbf{t} \in [0,\infty]^m$, $g_{z,\tau}(\mathbf{t}) \in [0,\infty]$. Specifically, we can think of it as a function $g_{z,\tau}(\mathbf{t}) = \boldsymbol{\alpha}^{z,\tau}\mathbf{t} + \beta^{z,\tau}$, where $\boldsymbol{\alpha}^{z,\tau} = (\alpha_1^{z,\tau}, \alpha_2^{z,\tau}, \ldots, \alpha_m^{z,\tau})$ and $\alpha_i^{z,\tau}, \beta^{z,\tau} \in [0,\infty]$.

Let $\mathbf{g}^\tau(\mathbf{t}) = (g_{w_1,\tau}(\mathbf{t}'), g_{w_2,\tau}(\mathbf{t}'), \ldots, g_{w_m,\tau}(\mathbf{t}'))$ where $\mathbf{c}^{\mathbf{u},\mathbf{w}} = (c_{u_1w_1}, c_{u_2w_2}, \ldots, c_{u_mw_m})$ and $\mathbf{t}' = \mathbf{t} + \mathbf{c}^{\mathbf{u},\mathbf{w}}$. Note $\mathbf{t} \in (-c_{u_1w_1}, \infty] \times (-c_{u_2w_2}, \infty] \times \ldots \times (-c_{u_mw_m}, \infty]$. We can represent $\mathbf{g}^\tau(\mathbf{t})$ as $D^\tau\mathbf{t} + \mathbf{d}^\tau$, where $D^\tau = [\boldsymbol{\alpha}^{w_1,\tau}; \boldsymbol{\alpha}^{w_2,\tau}; \ldots; \boldsymbol{\alpha}^{w_m,\tau}]$ and $\mathbf{d}^\tau_j = \sum_{i=0}^{m} \alpha_i^{w_j,\tau} c_{uw_i} + \beta^{w_j,\tau}$. Note that if $\mathbf{d}^\tau_j = 0$ then necessarily $\boldsymbol{\alpha}^{w_j,\tau} = \mathbf{0}$ and $\beta^{w_j,\tau} = 0$.

Consider function $\mathbf{f}(\mathbf{t}) = \min_\tau \mathbf{g}^\tau(\mathbf{t})$. This is well defined, since whatever the values in $\mathbf{t}$, the minimizer always has, by Corollary 4.2.3, an optimal SM strategy τ^* in $A(\mathbf{t})$ such that for any strategy σ of the maximizer, and any strategy τ of the minimizer, and all z we have $r_z^{*,\sigma,\tau^*,\mathbf{t}} \leqslant r_z^{*,\sigma,\tau,\mathbf{t}}$. Note that $\mathbf{f}(\mathbf{t}) = (f_{w_1}(\mathbf{t} + \mathbf{c}^{\mathbf{u},\mathbf{w}}), f_{w_2}(\mathbf{t} + \mathbf{c}^{\mathbf{u},\mathbf{w}}), \ldots, f_{w_m}(\mathbf{t} + \mathbf{c}^{\mathbf{u},\mathbf{w}}))$.

Lemma 4.3.2. *If $\mathbf{f}(\mathbf{t}) > \mathbf{t}$ for some finite vector $\mathbf{t}$, then for any fixed point $\mathbf{t}^*$ of $\mathbf{f}$, $\mathbf{t} \leqslant \mathbf{t}^*$.*

Proof. Suppose that $\mathbf{t}^*$ is some fixed point of $\mathbf{f}$. Since $\mathbf{f}(\mathbf{t}^*) = \min_\tau \mathbf{g}^\tau(\mathbf{t}^*)$, for some τ^* we have $\mathbf{g}^{\tau^*}(\mathbf{t}^*) = \mathbf{t}^*$. From the fact that $\mathbf{f}(\mathbf{t}) > \mathbf{t}$, we get that for all τ we have $\mathbf{g}^\tau(\mathbf{t}) > \mathbf{t}$. In particular we have $\mathbf{g}^{\tau^*}(\mathbf{t}) > \mathbf{t}$, which means that $D^{\tau^*}\mathbf{t} + \mathbf{d}^{\tau^*} > \mathbf{t}$. Now, for all i, either $\mathbf{d}_i^{\tau^*} = 0$ and the i-th row in D^{τ^*} is all zeroes, or $\mathbf{d}_i^{\tau^*} > 0$, thus by Lemma 4.2.4 we can conclude that $\mathbf{t} \leqslant \sum_{k=0}^{\infty} (D^{\tau^*})^k \mathbf{d}^{\tau^*}$. However, letting $h(\mathbf{t}) = \mathbf{g}^{\tau^*}(\mathbf{t}) = D^{\tau^*}\mathbf{t} + \mathbf{d}^{\tau^*}$ be the linear operator on $[0,\infty]^m$, note that the least fixed point solution (in $[0,\infty]^m$) of $h(\mathbf{t})$ is $\mathbf{t}_0 = \lim_{k\to\infty} h^{k+1}(\mathbf{0}) = \lim_{k\to\infty} D^{\tau^*} h^k(\mathbf{0}) + \mathbf{d}^{\tau^*} = \sum_{k=0}^{\infty} (D^{\tau^*})^k \mathbf{d}^{\tau^*}$. Thus, any other fixed point of h has to be greater than $\mathbf{t}_0$ and in particular $\mathbf{t}^* \geqslant \mathbf{t}_0 \geqslant \mathbf{t}$. □

Now, we know that $\mathbf{f}(\mathbf{t}^\sigma - \mathbf{c}^{\mathbf{u},\mathbf{w}})_i = f_{w_i}(\mathbf{t}^\sigma) = \mathbf{r}^{*,\sigma}_{w_i} > \mathbf{r}^{*,\sigma}_{v_i} + c_{u_iv_i} - c_{u_iw_i} = \mathbf{r}^{*,\sigma}_{u_i} - c_{u_iw_i} = (\mathbf{t}^\sigma - \mathbf{c}^{\mathbf{u},\mathbf{w}})_i$ which proves that $\mathbf{f}(\mathbf{t}^\sigma - \mathbf{c}^{\mathbf{u},\mathbf{w}}) > \mathbf{t}^\sigma - \mathbf{c}^{\mathbf{u},\mathbf{w}}$. Therefore, by Lemma 4.3.2, any fixed point of $\mathbf{f}$ has to be greater or equal to $\mathbf{t}^\sigma - \mathbf{c}^{\mathbf{u},\mathbf{w}}$. Also, if we switch strategy σ to σ', then $\mathbf{t}^{\sigma'} - \mathbf{c}^{\mathbf{u},\mathbf{w}}$ is a fixed point of $\mathbf{f}$ because $\mathbf{f}(\mathbf{t}^{\sigma'} - \mathbf{c}^{\mathbf{u},\mathbf{w}})_i = f_{w_i}(\mathbf{t}^{\sigma'}) = \mathbf{r}^{*,\sigma'}_{w_i} = \mathbf{r}^{*,\sigma'}_{u_i} - c_{u_iw_i} = (\mathbf{t}^{\sigma'} - \mathbf{c}^{\mathbf{u},\mathbf{w}})_i$. Thus $\mathbf{t}^\sigma \leqslant \mathbf{t}^{\sigma'}$. Since $\mathbf{f}$ is non-decreasing, then $\mathbf{r}^{*,\sigma'}_z = f_z(\mathbf{t}^{\sigma'}) \geqslant f_z(\mathbf{t}^\sigma) = \mathbf{r}^{*,\sigma}_z$ for any z, and for $u_1, u_2, \ldots, u_m$ the inequality is strict: $\mathbf{r}^{*,\sigma'}_{u_i} - c_{u_iw_i} =$

$\mathbf{r}^{*,\sigma'}_{w_i} \geqslant \mathbf{r}^{*,\sigma}_{w_i} > \mathbf{r}^{*,\sigma}_{v_i} + c_{u_i v_i} - c_{u_i w_i} = \mathbf{r}^{*,\sigma}_{u_i} - c_{u_i w_i}$.

Thus, switching to the new SM strategy σ', we get $\mathbf{r}^{*,\sigma'}$ which dominates $\mathbf{r}^{*,\sigma}$, and is strictly greater in some coordinates, including all the u_i's. There are finitely many SM strategies, thus repeating this we eventually reach some SM strategy σ^* that can't be improved. Thus $\mathbf{r}^{*,\sigma^*} = P(\mathbf{r}^{*,\sigma^*})$, and by our earlier claim $\mathbf{r}^{*,\sigma^*} = \mathbf{r}^*$. Thus, the maximizer has an optimal SM strategy, arrived at via simultaneous strategy improvement.

The containment in PLS of the problem of finding the optimal strategies of both players is an immediate consequence of Theorem 4.4.1 which will show that computing the optimal values of a minimizing 1-RMDP with positive rewards can be done in PTIME and hence each step of simultaneous strategy improvement (finding the improvable nodes for the current maximizer's strategy) can be performed in PTIME. □

4.4 The complexity of RMDPs and RSSGs with positive rewards

Theorem 4.4.1. *There is a* PTIME *algorithm for computing the exact optimal value (including the possible value ∞) of a 1-RMDP with positive rewards, in both the case where the single player aims to maximize, or to minimize, the total reward.*

We consider maximizing and minimizing 1-RMDPs with positive rewards separately.

4.4.1 Maximizing 1-RMDPs with positive rewards

We are given a maximizing reward 1-RMDP with positive rewards (i.e., no $\mathtt{Type}_{\min}$ nodes in the 1-RSSG with positive rewards). Let us call the following LP "*max-LP* ":
Minimize $\sum_{u \in Q} x_u$
Subject to:

$x_u = 0$	for all $u \in \mathtt{Type}_0$
$x_u \geqslant \sum_{v \in n(u)} p_{uv}(x_v + c_{uv})$	for all $u \in \mathtt{Type}_{\text{rand}}$
$x_u \geqslant x_{en} + x_{(b,ex')} + c_u$	for all $u = (b, en) \in \mathtt{Type}_{\text{call}}$; ex' is the exit of $Y(b)$.
$x_u \geqslant (x_v + c_{uv})$	for all $u \in \mathtt{Type}_{\max}$ and all $v \in n(u)$
$x_u \geqslant 0$	for all vertices $u \in Q$

We show that, when the value vector $\mathbf{r}^*$ is finite, it is precisely the optimal solution to the above max-LP, and furthermore that we can use this LP to find and eliminate vertices u for which $r^*_u = \infty$. Note that if $\mathbf{r}^*$ is finite then it fulfills all the constraints of the max-LP, and thus it is a feasible solution. We will show that it also has to be an optimal feasible solution. We first have to detect vertices u such that $\mathbf{r}^*_u = \infty$. For the

max-linear equation system P, we define the underlying directed dependency graph G, where the nodes are the set of vertices, Q, and there is an edge in G from u to v if and only if the variable x_v occurs on the right hand side in the equation defining variable x_u in P. We can decompose this graph in linear time into strongly connected components (SCCs) and get an SCC DAG $SCC(G)$, where the set of nodes are SCCs of G, and an edge goes from one SCC A to another B, iff there is an edge in G from some node in A to some node in B. Let us sort topologically the SCCs of G as $S_1, S_2, \ldots, S_l$, where the bottom SCCs are listed first. We will call a subset $U \subseteq Q$ of vertices ***decent*** if all vertices reachable in G from the vertices in U are already in U. We also use U to refer to the corresponding set of variables. Clearly, such a decent set U must be a union of SCCs, and the equations restricted to variables in U do not use any variables outside of U, so they constitute a proper equation system on their own. For any decent subset U of G, we will denote by max-LP$|_U$ a subset of equations of max-LP, restricted to the constraints corresponding to variables in U and with new objective $\sum_{u \in U} x_u$. Analogously we define $P|_U$, and let $\mathbf{x}|_U$ be the vector $\mathbf{x}$ with entries indexed by any $v \notin U$ removed.

Proposition 4.4.2. *Let U be any decent subset of vertices. (I) The vector $\mathbf{r}^*|_U$ is the LFP of $P|_U$. (II) If $r^*_u = \infty$ for some vertex u in an SCC S of G, then $r^*_v = \infty$ for all $v \in S$. (III) If $\mathbf{r}'$ is an optimal bounded solution to* max-LP$|_U$*, then $\mathbf{r}'$ is a fixed point of $P|_U$. (IV) If max-LP$|_U$ has a bounded optimal feasible solution $\mathbf{r}'$, then $\mathbf{r}' = \mathbf{r}^*|_U$.*

Proof. Part (I) follows immediately from definitions. Part (II) follows by induction on the length of the shortest path from any vertex $v \in S$ to u. In particular, if $x_v = \max\{x_w, \ldots\}$, and $r^*_w = \infty$, then $r^*_v = \infty$, and likewise for other vertex types. For part (III), observe that for each vertex $u \in Type_{max}$, if $\mathbf{r}'$ is an optimal bounded solution of the max-LP, then at least one of the constraints $x_u \geqslant x_v + c_{uv}$ holds ***tightly***, i.e., $x_u = x_v + c_{uv}$. For otherwise, we could decrease the value of x_u, letting $x_u = \max_{v \in n(u)}(x_v + c_{uv})$, and still satisfy all constraints. (Notice that $u \notin n(u)$ since otherwise x_u would not be bounded.) Similarly, the inequality for $u \in Type_{call}$ holds tightly. For $u \in Type_{rand}$ we can have $u \in n(u)$, but then necessarily $p_{uu} < 1$, and if the inequality is not tight for x_u, it can be assigned a value of $(p_{uu}c_{uu} + \sum_{v \in n(u) \setminus \{u\}} p_{uv}(x_v + c_{uv}))/(1 - p_{uu})$ that is smaller and still satisfy all the constraints. For part (IV), if max-LP$|_U$ has a feasible bounded solution, then the optimal (minimum) solution $\mathbf{r}'$ is bounded. From part (III), we know $\mathbf{r}'$ is a fixed point of $P|_U$, but then from the objective function of max-LP$|_U$, we know that $\mathbf{r}'$ is the LFP of $P|_U$, so we must have $\mathbf{r}' = \mathbf{r}^*|_U$. □

Theorem 4.4.3. *We can compute* $\mathbf{r}^*$ *for the max-linear equation system* P*, including the values that are infinite, in time polynomial in the size of the 1-RMDP with positive rewards.*

Proof. Build dependency graph G of P and decompose it into SCC DAG $SCC(G)$. We will find the LFP solution to P, bottom-up starting at a bottom SCC, S_1. We solve max-LP$|_{S_1}$ using a PTIME LP algorithm. If the LP is feasible then the optimal (minimum) value is bounded, and we plug in the values of the (unique) optimal solution as constants in all other constraints of max-LP. We know this optimal solution is equal to $\mathbf{r}^*|_{S_1}$, since S_1 is decent. We do the same, in bottom-up order, for remaining SCCs $S_2, \ldots, S_l$. If at any point after adding the new constraints corresponding to the variables in an SCC S_i, the LP is infeasible, we know by Proposition 4.4.2 (IV), that at least one of the values of $\mathbf{r}^*|_{S_i}$ is ∞. So by Proposition 4.4.2 (II), all are. We can then mark all variables in S_i as ∞, and also mark all variables in the SCCs that can reach S_i in $SCC(G)$ as ∞. Also, at each step we add to a set U the SCCs that have finite optimal values. At the end we have a maximal decent set U, i.e., every variable outside of U has value ∞. We obtain the vector $\mathbf{r}^*$ by assigning variables in U the values obtained by solving LPs and setting all the rest to ∞. All of this can be done in polynomial time. □

4.4.2 Minimizing 1-RMDPs with positive rewards

Given a minimizing 1-RMDP with positive rewards (i.e., no $Type_{max}$ nodes) we want to compute $\mathbf{r}^*$. Call the following LP "*min-LP:* "

Maximize $\sum_{u \in Q} x_u$

Subject to:

$x_u = 0$	for all $u \in Type_0$
$x_u \leqslant \sum_{v \in n(u)} p_{uv}(x_v + c_{uv})$	for all $u \in Type_{rand}$
$x_u \leqslant x_{en} + x_{(b,ex')} + c_u$	for all $u = (b, en) \in Type_{call}$; ex' is the exit of $Y(b)$.
$x_u \leqslant (x_v + c_{uv})$	for all $u \in Type_{min}$ and all $v \in n(u)$
$x_u \geqslant 0$	for all vertices $u \in Q$

Lemma 4.4.4. *For any decent set* U*, if an optimal solution* $\mathbf{x}$ *to* min-LP$|_U$ *is bounded, it is a fixed point of the min-linear operator* $P|_U$*. Thus, if min-LP*$|_U$ *has a bounded optimal feasible solution then* $\mathbf{r}^*|_U$ *is bounded (i.e., is a real vector).*

The proof is analogous to the max-LP case. Now, from min-LP we can remove variables $x_u \in Type_0$, by substituting their occurrences with 0. Assume, for now, that we

can also find and remove all variables x_u such that $r^*_u = \infty$. By removing these 0 and ∞ variables from P we obtain a new system P', and a new LP, min-LP$'$.

Lemma 4.4.5. *If ∞ and 0 nodes have been removed, i.e., if $\mathbf{r}^* \in (0,\infty)^n$, then $\mathbf{r}^*$ is the unique optimal feasible solution of min-LP$'$.*

Proof. By Corollary 4.2.3, Player 2 has an optimal SM strategy, call it τ, which yields the finite optimal reward vector r^*. Once strategy τ is fixed, we can define a new equation system $P'_\tau(\mathbf{x}) = A_\tau \mathbf{x} + b_\tau$, where A_τ is a nonnegative matrix and b_τ is a vector of average rewards per single step from each node, obtained under strategy τ. We then have $\mathbf{r}^* = \lim_{k\to\infty}(P'_\tau)^k(0)$, i.e., $\mathbf{r}^*$ is the LFP of $x = P'(x)$.

Proposition 4.4.6. *(I) $\mathbf{r}^* = (\sum_{k=0}^{\infty} A_\tau^k) b_\tau$. (II) If $\mathbf{r}^*$ is finite, then $\lim_{k\to\infty} A_\tau^k = 0$, and thus $(I - A_\tau)^{-1} = \sum_{i=0}^{\infty}(A_\tau)^i$ exists (i.e., is a finite real matrix).*

Proof. (I): $\mathbf{r}^* = \lim_{k\to\infty}(P'_\tau)^{k+1}(0) = \lim_{k\to\infty} A_\tau (P'_\tau)^k(0) + b_\tau = \lim_{k\to\infty}(\sum_{i=0}^{k}(A_\tau)^k) b_\tau$. (This holds regardless of whether $\mathbf{r}^*$ is finite. We shall use this fact in a subsequent proof.)
(II): since $\mathbf{r}^* = P'_\tau(\mathbf{r}^*)$, we have, for any $k \geqslant 0$, $\mathbf{r}^* = A_\tau^k \mathbf{r}^* + (I + A_\tau + A_\tau^2 + \ldots + A_\tau^{k-1}) b_\tau$. The second part of the right hand side, in the limit, is equal to $\mathbf{r}^*$, thus $A_\tau^k \mathbf{r}^*$ in the limit is an all-zero vector. It follows that the limit of A_τ^k is an all-zero matrix since all the entries/rewards in $\mathbf{r}^*$ are positive (we have already removed 0 entries). □
Now pick an optimal SM strategy τ for Player 2 that yields the finite $\mathbf{r}^*$. We know that $\mathbf{r}^* = (I - A_\tau)^{-1} b_\tau$. Note that $\mathbf{r}^*$ is a feasible solution of the min-LP$'$. We show that for any feasible solution $\mathbf{r}$ to min-LP$'$, $\mathbf{r} \leqslant \mathbf{r}^*$. From the LP we can see that $\mathbf{r} \leqslant A_\tau \mathbf{r} + b_\tau$ (because this is just a subset of the constraints) and in other words $(I - A_\tau)\mathbf{r} \leqslant b_\tau$. We know that $(I - A_\tau)^{-1}$ exists and is non-negative (and finite), so multiply both sides by $(I - A_\tau)^{-1}$ to get $\mathbf{r} \leqslant (I - A_\tau)^{-1} b_\tau = \mathbf{r}^*$. Thus $\mathbf{r}^*$ is the optimal feasible solution of min-LP$'$. □

For $u \in Q$, consider the LP: **Maximize** x_u, **subject to:** the same constraints as min-LP, except, again, remove all variables $x_v \in \mathtt{Type}_0$. Call this u-min-LP$'$.

Theorem 4.4.7. *In a minimizing 1-RMDP with positive rewards, for all vertices u, value $\mathbf{r}^*_u$ is finite iff u-min-LP$'$ is feasible and bounded. Thus, combined with Lemma 4.4.5, we can compute the exact value (even if ∞) of minimizing 1-RMDPs with positive rewards in* PTIME.

Proof. We first need some preliminary claims. Let W be the set of vertices u such that u-min-LP$'$ is bounded and let S be the minimum decent set such that $W \subseteq S$. From min-LP remove all the constraints for variables outside of the set S and remove the variables of $\mathtt{Type}_0$ in the same way as before. Call this set of constraints LP_S.

Proposition 4.4.8. *For any two vectors* $\mathbf{x} = [x_1, x_2, \ldots, x_n], \mathbf{y} = [y_1, y_2, \ldots, y_n]$ *and vector* $\mathbf{z} = \max(\mathbf{x}, \mathbf{y}) = [\max(x_1, y_1), \max(x_2, y_2), \ldots, \max(x_n, y_n)]$, *and subset* $A \subseteq \{1, 2, \ldots, n\}$, *and constants* $p_{ij} \geqslant 0, c_{ij} \geqslant 0$ *we have that:*

1. *if vectors* $\mathbf{x}, \mathbf{y}$ *fulfil a linear constraint* $\widetilde{x}_i \leqslant \sum_{j \in A} p_{ij}(\widetilde{x}_j + c_{ij})$ *then so does* $\mathbf{z}$
2. *if vectors* $\mathbf{x}, \mathbf{y}$ *fulfil a constraint* $\widetilde{x}_i \leqslant \min_{j \in A}(\widetilde{x}_j + c_{ij})$ *then so does* $\mathbf{z}$

Proof.

1. Function max is monotonic, hence if $x_i \leqslant x_j$ and $y_i \leqslant y_j$, then $\max(x_i, y_i) \leqslant \max(x_j, y_j)$. Thus $\max(x_i, y_i) \leqslant \max\big(\sum_{j \in A} p_{ij}(x_j + c_{ij}), \sum_{j \in A} p_{ij}(y_j + c_{ij})\big)$ based on the fact that they fulfil the underlying constraint. However we know that for all j we have that $x_j \leqslant \max(x_j, y_j) = z_j$ and $y_j \leqslant \max(x_j, y_j) = z_j$, hence $\sum_{j \in A} p_{ij}(x_j + c_{ij}) \leqslant \sum_{j \in A} p_{ij}(z_j + c_{ij})$ and $\sum_{j \in A} p_{ij}(y_j + c_{ij}) \leqslant \sum_{j \in A} p_{ij}(z_j + c_{ij})$, which means that $z_i = \max(x_i, y_i) \leqslant \max\big(\sum_{j \in A} p_{ij}(x_j + c_{ij}), \sum_{j \in A} p_{ij}(y_j + c_{ij})\big) \leqslant \sum_{j \in A} p_{ij}(z_j + c_{ij})$
2. Again we know that $\max(x_i, y_i) \leqslant \max\big(\min_{j \in A}(x_j + c_{ij}), \min_{j \in A}(y_j + c_{ij})\big)$ and for all j we have $x_j + c_{ij} \leqslant z_j + c_{ij}$ and $y_j + c_{ij} \leqslant z_j + c_{ij}$. We also know that the min function is monotonic, hence $\min_{j \in A}(x_j + c_{ij}) \leqslant \min_{j \in A}(z_j + c_{ij}) \geqslant \min_{j \in A}(y_j + c_{ij})$. This means that $z_i = \max(x_i, y_i) \leqslant \max\big(\min_{j \in A}(x_j + c_{ij}), \min_{j \in A}(y_j + c_{ij})\big) \leqslant \min_{j \in A}(z_j + c_{ij})$.

□

Corollary 4.4.9. *For any two feasible solutions* $\mathbf{x}, \mathbf{y}$ *to* LP_S *we have that* $\mathbf{z} = \max(\mathbf{x}, \mathbf{y}) = [\max_i(\mathbf{x}_i, \mathbf{y}_i)]$ *(vector with entries being the maximum of the respective entries in* $\mathbf{x}$ *and* $\mathbf{y}$*) is a feasible solution to* LP_S *as well.*

Now we can proceed to the proof of the theorem.

($\Rightarrow$) First, let us show that for any u if $\mathbf{r}^*_u$ is finite, then u-min-LP$'$ has to be feasible and bounded. Feasibility is easy as the all zero vector $\mathbf{0}$ fulfills all the constraints in u-min-LP$'$.

Now pick the optimal SM strategy τ for the minimizer that yields the optimal reward vector $\mathbf{r}^*$ and take any feasible vector $\mathbf{x}$. From the u-min-LP$'$ we can see that $\mathbf{x} \leqslant A_\tau \mathbf{x} + b_\tau$ (because this is just a subset of the constraints). Since we removed all zero reward nodes ie. exits of components, all entries of b_τ are positive and by Lemma 4.2.4 we have $\mathbf{x} \leqslant (\sum_{k=0}^{\infty} A_\tau^k) b_\tau$. However, by Proposition 4.4.6 (I) (which holds regardless of whether $\mathbf{r}^*$ is finite) this means that $\mathbf{x} \leqslant \mathbf{r}^*$ for any feasible $\mathbf{x}$.

For contradiction, assume u-min-LP$'$ was feasible but unbounded. Then there would exist a sequence of feasible vectors $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots$ such that $\lim_{k\to\infty} \mathbf{x}^k_u = \infty$. But we know that $\mathbf{x}^k \leqslant \mathbf{r}^*$ for all k, thus $\mathbf{r}^*_u$ would have to be infinite, contradicting our assumption.

($\Leftarrow$) Now let us show that if u-min-LP$'$ is feasible and bounded then $\mathbf{r}^*_u$ has to be finite. Consider an LP with LP_S constraints and with objective: **Maximize** $\sum_{u\in W} x_u$. Call it W-min-LP and for any optimal solution $\mathbf{x}^*$ denote by $\overline{\mathbf{x}}^*$ the vector such that $\overline{\mathbf{x}}^*_u = \mathbf{x}^*_u$ for all $u \in W$ and $\overline{\mathbf{x}}^*_u = \infty$ for all $u \in S \setminus W$. Notice that $\overline{\mathbf{x}}^*$ is unique, because if two different optimal vectors $\mathbf{x}$ and $\mathbf{x}'$ differ at a value of some variable $x_u \in W$, then $\max(\mathbf{x}, \mathbf{x}')$ would be larger than $\mathbf{x}$ or $\mathbf{x}'$, and also a feasible solution by Corollary 4.4.9, which would contradict the optimality of $\mathbf{x}$ or $\mathbf{x}'$.

Lemma 4.4.10. *The vector $\overline{\mathbf{x}}^*$ is a fixed point of $\mathrm{P}|_S$.*

Proof. Since for every variable $u \in W$, u-min-LP$'$ is bounded, and we removed from u-min-LP$'$ only the constraints that these variables do not depend on (even in a transitive way), the maximum value of x_u cannot possibly increase after we remove these constraints, because that would mean x_u could have obtained a higher value in u-min-LP$'$. Hence the LP W-min-LP is feasible and bounded.

Let us take any optimal solution $\mathbf{x}^*$ to W-min-LP. We show now that for $\mathbf{x}^*$ no constraint with a variable $x_u \in W$ on the left hand side can hold tightly (i.e., with equality) when there is some variable from $S \setminus W$ on the right hand side. Let $S \setminus W = \{v_1, v_2, \ldots, v_n\}$ be the set of unbounded variables, i.e., v_i-min-LP is unbounded. We know that for each of them there is a sequence of feasible solutions $\mathbf{x}^{v_i}_{\mathbf{1}}, \mathbf{x}^{v_i}_{\mathbf{2}}, \mathbf{x}^{v_i}_{\mathbf{3}}, \ldots$ to v_i-min-LP (the bold subscripts denote the position in this sequence, not the position inside the vector), such that the value of entry x_{v_i} in this sequence of vectors is nondecreasing and becomes arbitrarily large. If we project this sequence onto the variables in S then $\mathbf{x}^{v_i}_{\mathbf{1}}|_S, \mathbf{x}^{v_i}_{\mathbf{2}}|_S, \mathbf{x}^{v_i}_{\mathbf{3}}|_S, \ldots$ is a sequence of feasible solutions to W-min-LP, such that v_i becomes arbitrarily large. Let us construct a sequence of vectors $\mathbf{x}'_{\mathbf{i}} = \max(\mathbf{x}^*, \mathbf{x}^{v_1}_{\mathbf{i}}|_S, \mathbf{x}^{v_2}_{\mathbf{i}}|_S, \ldots, \mathbf{x}^{v_n}_{\mathbf{i}}|_S)$. By Corollary 4.4.9 we know that all vectors in this sequence are feasible solutions to W-min-LP. We also know that all of them are optimal solutions, because we always take the maximum of their entries, including the entries in the optimal solution $\mathbf{x}^*$. So we obtain as high a value of objective function $\sum_{u\in W} x_u$ as before, and we cannot improve this value as it would contradict the assumption that $\mathbf{x}^*$ was optimal. Now, notice three things:

1. Since every variable $x_u \in W$ is bounded, at some point in this sequence, we

will reach a point such that the r.h.s. of any constraint which involves some variable $x_u \in S \setminus W$ will be larger than the highest possible value of all variables in W. This means that at that point there cannot be a constraint that holds with equality such that $x_u \in W$ is the l.h.s. and where there is a variable from $S \setminus W$ on the r.h.s.

2. For all k, for every $x_u \in W$ there has to be some constraint with x_u on the l.h.s. such that $\mathbf{x}'_\mathbf{k}$ satisfies this constraint tightly (with equality) because otherwise we could increase the value of x_u without altering the value of any other variables, to obtain a larger value for the objective, which contradicts the optimality of $\mathbf{x}'_\mathbf{k}$.

3. All variables $x_v \in S \setminus W$ become arbitrarily large in this sequence, thus it cannot be the case that there are only variables from W on the r.h.s. in any constraint with x_v on the l.h.s. (that would force this variable to be bounded).

Using these facts, we can see that for a large enough k, from the vector $\mathbf{x}'_\mathbf{k}$ we can construct a vector $\overline{\mathbf{x}}^*$ which is a fixed point of $P|_S$. We do so by setting all variables in $S \setminus W$ to ∞, and leaving the variables in W unchanged from $\mathbf{x}'_\mathbf{k}$. The claim that $\overline{\mathbf{x}}^*$ is a fixed point of $P|_S$ follows because for every variable $x_u \in W$ of type $\mathtt{Type}_{\text{rand}}$ or $\mathtt{Type}_{\text{call}}$, $\mathbf{x}'_\mathbf{k}$ satisfies the correlated constraint with x_u on the l.h.s. with equality, and this can only be the case if the r.h.s. of that constraint contains only variables in W, and thus $\overline{\mathbf{x}}^*$ also satisfies this constraint with equality. Likewise, for variables x_u in W of type $\mathtt{Type}_{\text{min}}$, for $\mathbf{x}'_\mathbf{k}$ all constraints such that x_u is the l.h.s. and there is at least one variable from $S \setminus W$ on the r.h.s., must hold with strict inequality. Hence, since equality must hold in $\mathbf{x}'_\mathbf{k}$ for one of the constraints involving x_u on the l.h.s., there must exist one such constraint such that the r.h.s. only involves variables in W. Thus, equality also holds for these constraints for $\overline{\mathbf{x}}^*$ for these variables. Thus $\overline{\mathbf{x}}^*$ satisfies the corresponding *min* equation in $P|_S$. Also, for variables in $x_v \in S \setminus W$, all the equations in $P|_S$ will clearly be fulfilled after setting their values to ∞, because both sides of the equations correlated to x_v have at least one variable from $S \setminus W$, and that makes both sides have value ∞. □

Now finally we can finish the proof of the theorem using the previous lemma. Since we know that $\mathbf{r}^*|_S$ is the LFP of the operator $P|_S$, it has to be the case that $\mathbf{r}^*|_S \leqslant \overline{\mathbf{x}}^*$, which means that for all $u \in W$, we have that $\mathbf{r}^*_u|_S \leqslant \overline{\mathbf{x}}^*_u = \mathbf{x}^*_u$, which is finite. □

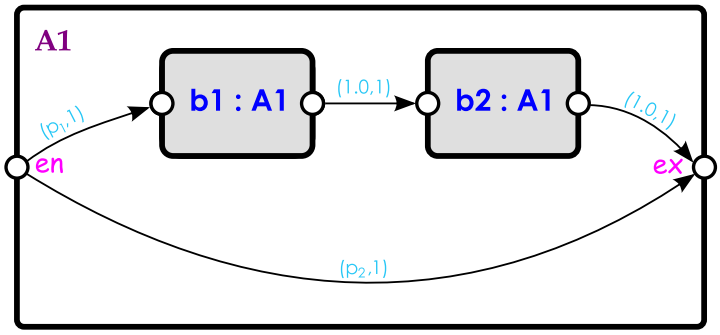

Figure 4.2: Standard 1-RMC gadget used in the proof of Theorem 4.4.12. If $p_1 > 1/2$ then we terminate with probability strictly less than 1 (and of course the expected termination time is ∞). If $p_1 < 1/2$ then we terminate with probability 1 and the expected time if finite. However, if $p_1 = 1/2$ then we terminate with probability 1, but the expected time of termination is ∞.

4.4.3 Complexity of (1-)RSSGs with positive rewards

Theorem 4.4.11. *Deciding whether the value r^*_u of a 1-RSSG with positive rewards is $\geqslant a$ for a given $a \in [0,\infty]$, is in NP $\cap$ coNP.*

This is immediate from PTIME upper bounds for 1-RMDPs, and SM-determinacy: guess a player's SM strategy, and compute the value for the remaining 1-RMDP.

Theorem 4.4.12. *Condon's quantitative termination problem for finite SSGs is* PTIME *many-one reducible to the problem of deciding whether $r^*_u = \infty$.*

Proof. Consider the standard 1-RMC from [EY05s], depicted in Figure 4.2. From the entry, en, this 1-RMC goes with probability p_1 to a sequence of two boxes labeled by the same component and with probability p_2 goes to the exit. We assume $p_1 + p_2 = 1$. As shown in ([EY05s], Theorem 3), in this 1-RMC the probability of termination starting at $\langle \varepsilon, en \rangle$ is equal to 1 if and only if $p_2 \geqslant 1/2$.

Now, given a finite SSG, G, and a vertex u of G, do the following: first "clean up" G by removing all nodes where the minimizer has a strategy to achieve probability 0. We can do this in polynomial time. (If u is among these nodes, we would already be done, but assume it is not.) The revised SSG will have two designated terminal nodes, the old terminal node, labeled "1", and another terminal node labeled "0". From every node v in the revised SSG which does not carry full probability on its outedges, we direct all the "residual" probability to "0", i.e., we add an edge from v to "0" with probability $p_{v,"0"} = 1 - \sum_w p_{vw}$, where the sum is over all remaining nodes w in the SSG. In the resulting finite SSG, we know that if the maximizer plays with an

optimal memoryless strategy (which he has), and the minimizer plays arbitrarily with a memoryless strategy, there is no bottom SCC in the resulting finite Markov chain other than the two designated terminating nodes "0" and "1". In other words, all the probability exits the system, as long as the maximizing player plays optimally. Note also that, importantly, the "expected time" that it takes for the probability to exit the system when the maximizer plays optimally is finite (because there are no "null recurrent" nodes in a finite Markov chain).

Another way to put this fact is as follows: consider the resulting SSG to be a finite reward SSG with reward 1 on each transition, and switch the role of the maximizer and the minimizer, and now the goal of the maximizer is to maximize the total reward before termination (at either exit), and that of the minimizer is to minimize it. Translating the above to this setting, the "cleaned up" SSG has the property that the minimizer has a memoryless strategy using which, no matter what the maximizer does, the total reward will be finite: we will terminate, at "0" or at "1", in finite expected time (because there are no "null recurrent" nodes in finite Markov chains, and both players have optimal memoryless strategies).

Now, take the remaining finite SSG, call it G'. Just put a copy of G' at the entry of the component A_1 of the 1-RMC, identifying the entry en with the initial node, u, of G'. Take every edge that is directed into the terminal node "1" of G, and instead direct it to the exit ex of the component A_1. Next, take every edge that is directed into the terminal "0" node and direct it to the first call, (b_1, en) of the left box b_1. Both boxes map to the unique component A_1. Call this 1-RSSG A.

We now claim that the value q^*_u is less or equal to $1/2$ in the finite SSG G' for terminating at the terminal "1" iff r^*_u is equal to ∞, i.e., the expected reward value in the resulting 1-RSSG with positive rewards A is ∞ (recall: with the role of the minimizer and the maximizer reversed, and with all transitions having reward 1).

The reason is as follows: we know that in A the minimizer has at least one SM strategy that obtains finite reward inside any copy of G', and she must play one such strategy each time she goes through G' if she wants to avoid payoff ∞.

Now, there are only a finite number of SM strategies for the minimizer inside G' which yield a finite expected reward (after an optimal response by the maximizer). Let $D \in [0, \infty)$ be the maximum finite expected reward among those SM strategies. Also, no matter what the two players do, we know the maximizer earns reward at least 1, each time we go through G'. So, each time going through G' he accumulates a reward $D' \in [1, D]$. So, from the point of view of trying to make sure the total expected

reward is finite, it is really of no relevance what the specific value of D' is when we go through G'. Rather, what is important is whether we "visit" a copy of G', i.e., a copy of the entry u, infinitely often.

Now, to make sure that that the expected number of times u is visited is finite, the minimizer must in fact maximize the probability of terminating at "1", and thus minimize the probability of termination at "0". In addition, the minimizer must also make sure that the expected reward inside G' is finite, but this we know she can do while maximizing the probability of terminating at "1". Thus, the total reward r^*_u is equal to ∞ precisely when the value of the SSG termination game G' is $\leqslant 1/2$. □

By contrast, for finite-state SSGs with strictly positive rewards, we can decide in PTIME whether the value is ∞, because this is the case iff the value of the corresponding termination game is not 1. This is basically because null-recurrence is not possible in finite state spaces. Deciding whether an SSG termination game has value 1 is in PTIME (see, e.g., [EY06s]).
Finally, we show undecidability for multi-exit RMDPs and RSSGs.

Theorem 4.4.13. *For multi-exit positive RMDPs with positive rewards it is undecidable to distinguish whether the optimal expected reward for a node is finite or ∞.*

Proof. We will use in our construction the component named A in the proof of Theorem 6 in [EY05i] that we can see in Figure 4.3 on the right. This single-entry n-exit component relates RMDPs with n exits with Probabilistic Finite Automata (PFA) with n states. More precisely the supremum probability of termination at the n-th exit starting at the entry of A is equal to the supremum probability with which the correlated PFA accepts some word. It was proved in [BC03] that deciding whether a given PFA with 46 states accepts any word with probability greater than $\frac{1}{2}$ is undecidable. This means it is undecidable to resolve whether the supremum probability of termination at the n-th exit ($n = 46$) in the correlated RMDP A is greater than $\frac{1}{2}$.

To prove that it is also undecidable to resolve whether the reward at a given node is finite or not, we will combine the RMDP A with a gadget 1-RMDP C, as can be seen on the left in Figure 4.3. Let us denote by p the supremum probability of termination at the n-th exit of the component A that box $b0$ is mapped to. We will argue that $p > 1/2$ iff the infimum total reward for 1-RMDP with positive rewards C is finite.

We will need the following observation about the component A. Namely, for any strategy that yields probability > 0 of exiting from the n-th exit of component A, it must be the case that the total probability of exiting from one of the exits of component

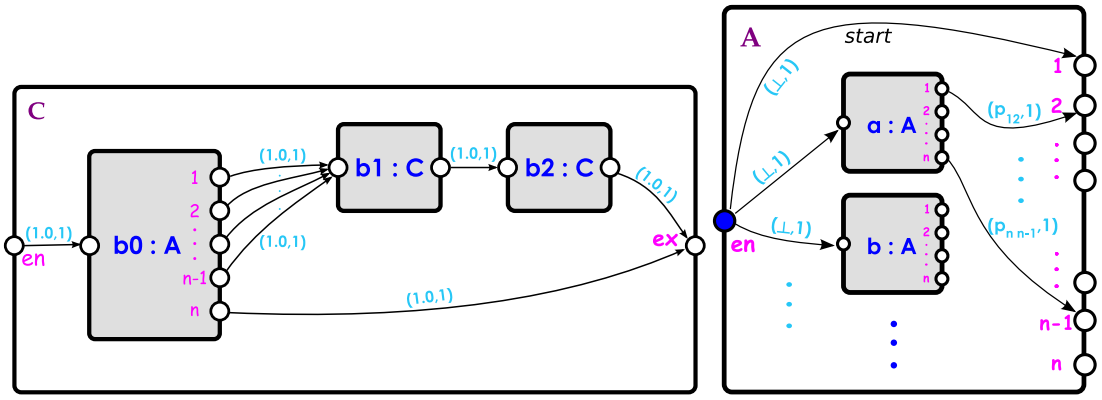

Figure 4.3: A multi-exit RMDP with positive rewards. Deciding if the expected time of termination at ex when starting in en (both in C) is ∞ is undecidable. The component A (on the right) with n exits, which was used in the proof of Theorem 6 in [EY05i], is tightly related to Probabilistic Finite Automata (PFA) with n states. Each letter $a, b, \ldots$ of the alphabet of the PFA has a corresponding box. There is a transition from an exit i of box x to an exit j of A with probability p_{ij} iff there is a transition in that PFA that upon reading x changes the control state i to j with probability p_{ij}. In order to check whether a given word w is accepted with probability $\geqslant 1/2$, the minimizer enters boxes corresponding to the letters in w in a reverse order and then picks the *start* transition. The expected time of termination in C is finite iff there exists a word w that reaches the n-th exit port of box $b0$ in C with probability $\geqslant 1/2$. Since establishing whether the language of a PFA with threshold $1/2$ is empty is undecidable ([CL89, BC03]), we also know that establishing whether the optimal expected reward of an multi-exit RMDP is ∞ is undecidable.

A is 1. It is easy to verify this fact based on the structure of component A given in [EY05i].

Now, first suppose $p > 1/2$. It follows from the previously mentioned fact that in the reward game the minimizer has a strategy with which to exit from A with probability 1, and simultaneously to exit from the n-th exit with probability $> 1/2$. Therefore, note that component C, under an optimal strategy played inside box B, acts like our favorite gadget in which the probability of exiting directly is $p > 1/2$. For this gadget, with $p > 1/2$ we know that the resulting expected time until termination is finite.

Moreover, the component A has the property that if $p > 1/2$, then the corresponding PFA accepts a word w with probability $> 1/2$, and we can use word w as a strategy σ_w in A such that starting at the entry of A, the strategy σ_w will exit A with probability 1, exit from the n-the exit with probability $p > 1/2$, and exit from A in finite expected

time $2|w|$. Thus the expected time taken until termination inside A, i.e., inside the box B is finite, and hence the total expected time until termination starting at the entry of C is finite.

Next suppose that the infimum total reward is finite, but that $p \leqslant 1/2$. Then in C we either stay inside a copy of $B(A)$ with nonzero probability, in which case the total reward is infinite, or else we exit from the n-th exit with probability $p \leqslant 1/2$ and we exit from the other exits with probability $\geqslant 1/2$. It follows easily from the properties of the gadget in C that the expected termination time is infinite in such a case. Thus if we can decide whether the optimal reward at the entry of C is finite or not, we can also decide whether the termination probability at the n-th exit of B is greater than $\frac{1}{2}$, which we know is undecidable. □

4.5 Conclusions

In this chapter we studied an important class of 1-RSSGs with all rewards on transitions being positive. This assumption is very natural for modeling optimal/pessimal expected running time in probabilistic procedural programs: each discrete step of the program is assumed to take some nonzero amount of time. At the same time this assumption is crucial to all our results since even determinacy is unknown for games that allow some of the rewards to be equal to zero.

We showed that 1-RSSGs with positive rewards always have a value (which is not immediately obvious since Martin's theorem ([Mar98, MS98]) can not be applied) that is either rational with polynomial bit complexity or ∞, and which arises as the least nonnegative solution over the extended reals of a min-max linear equation system. Furthermore, both players have optimal stackless and memoryless strategies in such games.

We provided a PTIME algorithm for computing the exact value of these games in a one player setting (both maximizing and minimizing 1-RMDPs with positive rewards) via a Linear Programming formulation. It easily follows that the decision questions about the two player version can be solved in NP ∩ co-NP.

We also described a practical simultaneous strategy improvement algorithm, analogous to similar algorithms for finite-state stochastic games, and showed that it converges to the game value (even if it is ∞) in a finite number of steps. It follows that computing the game value and optimal strategies for these games are contained in the complexity class PLS ([JPY88]). Whether this strategy improvement algorithm runs in

the worst-case in PTIME is open, just like its version for finite-state SSGs.

We observed that quantitative questions about Condon's finite-state Simple Stochastic Games can be reduced to deciding whether the value of a given 1-RSSG with positive rewards is ∞. By contrast, such a qualitative question for turn-based finite-state stochastic games with nonnegative rewards can be solved in PTIME. On the other hand, we showed that the same qualitative question for multi-exit RMDPs with positive rewards is undecidable.[6]

One remaining open question is whether computing the value of 1-RSSGs with positive rewards is contained in PPAD (defined in [Pap94]). It was shown in [EY07] that finding a fixed point of a min-max-linear system of equations over a compact convex domain can be done in PPAD. It follows from this that Condon's games are in PPAD. Although the equation system for computing the values of a 1-RSSG with positive rewards is precisely a min-max-linear equation system with nonnegative constants, the solution we are looking for may not be finite. As a result, we cannot establish its containment in PPAD since the domain is not compact.

A natural extension of the 1-RSSGs with positive rewards is to allow the players to make concurrent moves, i.e., choose simultaneously and independently at each node one of the available actions. To each node and each pair of chosen actions, we assign a reward and a probability distribution on the possible target nodes where the game proceeds in the next step. (Basically, they describe (undiscounted) concurrent countable-state stochastic games with positive rewards and the total expected reward objective.) In order to solve such a model, results presented in this chapter would need to be combined with results in [EY06i]. In fact, most of the work involved is straight forward, a technical difficulty being that no ε-optimal strategies may exist in such games.

It would interesting to apply the technique used in the proof of Theorem 4.3.1, which establishes the simultaneous strategy improvement algorithm for 1-RSSGs with positive rewards, to the setting of termination 1-RSSGs, their concurrent extension, and a concurrent extension of 1-RSSGs with positive rewards.

As we can see, the natural assumption of all rewards being positive endows our model with some crucial properties that allows for an efficient analysis. It would be interesting to know whether decision questions about the more general model of 1-RSSGs (with nonnegative rewards) are decidable or not. One basic problem is that in

[6] We cannot state this fact for multi-exit RSSGs since it is not even clear whether such games are determined or not.

general we do not even know whether such games are determined or not, since Martin's theorem ([Mar98, MS98]) can not be applied to such a setting. Furthermore, as discussed in Section 2.2.1 of Chapter 2, even qualitative questions about a special class of these games, namely reachability 1-RSSGs (which are known to be determined), are not fully answered yet.

Chapter 5

PReMo – Probabilistic Recursive Models analyzer

5.1 Introduction

In this chapter we describe our tool PReMo (**P**robabilistic **Re**cursive **Mo**dels analyzer) that we implemented in Java and report on some interesting experiments that we conducted with it. The main aim of PReMo (read as *primo*) is to allow the user to specify and analyze abstract models of probabilistic procedural programs and other systems involving recursion and probability. Probability comes either from an explicit randomization like in the Quicksort algorithm, or by abstracting some aspects of a program, or by modeling input data to a program. PReMo is capable of analyzing Recursive Markov Chains (RMCs), Stochastic Context-Free Grammars (SCFGs) and probabilistic 1-Counter Automata (p1CAs) (equivalent to Quasi-Birth-Death processes (QBDs)). Various other models like probabilistic Pushdown Systems (pPDSs), Multi-Type Branching Processes (MT-BPs) and discrete-time Tree-like QBDs can be analyzed indirectly since they can be translated by hand to the models supported by PReMo. These models have been well-studied in domains such as model checking, natural language processing, population dynamics and queueing theory. Also, controlled and game extensions of these models: Recursive Markov Decision Processes (RMDPs) and Recursive Simple Stochastic Games (RSSGs) can be analyzed in PReMo. These extensions allow for modeling of nondeterministic and interactive behavior. PReMo is the first (and still only) software tool that is capable of analyzing all these models.[1]

[1] For formal definitions of all these models see Section 2.2 and for detailed relationships between them and equivalences see Section 2.3, both part of Chapter 2.

These models can be specified in PReMo in several different input formats, including a simple imperative-style language for specifying RMCs and RSSGs. Furthermore, for RMCs and RSSGs, PReMo can generate a graphical depiction of the model, which is useful for visualizing small models (see an example in Figure 5.1).

In order to describe RMCs, let us discuss an example depicted in Figure 5.1. Informally, an RMC consists of several component Markov Chains (in Figure 5.1 these are named A and B) that can call each other recursively. Each component consists of nodes and boxes with possible probabilistic transitions between them. Each box is mapped to a specific component so that every time we reach an entry of a box, we jump to the corresponding entry of the component it is mapped to. When/if we finally reach an exit node of this component, we jump back to the respective exit of the box that we have entered this component from. This process models, in an obvious way, function invocation in a probabilistic procedural program. Every potential function call is represented by a box. Entry nodes represent possible parameter values passed to the function, while exit nodes represent returned values. An instruction of a function paired with one of possible valuations of its variables is represented by an internal node of a component.

We are interested in the following question about termination of a given RMC: what is the probability of reaching from a given entry of a component and an empty stack, a given exit of the same component with an empty stack. Computation of these probabilities is a key ingredient for model checking and other analyses for RMCs and pPDSs. This problem was shown to be equivalent to finding the Least Fixed Point (LFP) of a nonlinear polynomial equation system ([EY05s, EKM04]). For general RSSGs this question was shown to be undecidable, but for a restricted class of 1-exit RMDPs and 1-exit RSSGs, this problem was reduced to finding the LFP of a nonlinear polynomial equation system with min and max operators ([EY05i]). PReMo implements a number of efficient numerical algorithms for analyzing such systems of equations. Methods provided include both dense and sparse versions of the decomposed Newton's method developed in [EY05s], as well as versions of value iteration, optimized using nonlinear generalizations of Gauss-Seidel and SOR techniques. All of them can be further enhanced by running transparently on top of them a strategy improvement procedure. In addition to computing termination probabilities, PReMo can compute the (maximum/minimum/game) *expected termination time* in 1-RMCs, 1-RMDPs, 1-RSSGs, and SCFGs. It does so by generating a different monotone system of linear (min-max) equations, whose LFP is the value of the game where the objectives

of the two players are to maximize/minimize the expected termination time (these expected times can be ∞).

We conducted a range of experiments with PReMo in order to compare the performance of all these different numerical procedures. Our experiments indicate very promising potential for some of them. In particular, our decomposed Sparse Newton's method performed very well on most models we tried, up to quite large sizes.[2] Although these numerical methods appear to work well in practice on most instances, the exact complexity of the underlying computational problems still remains unknown ([EY05s, EY07]). Recently, much progress has been made on analyzing how many Newton's method iterations are needed to compute the LFP of monotone min-max polynomial equation systems within a given relative error ε for certain classes of such systems ([KLE07, EKL08, EWY08q] and Chapter 3). However, these analyses apply only to Newton's method performed with exact arithmetic, while all the computations in PReMo are performed in the IEEE standard double precision floating-point arithmetic (see Section 5.2.5). Because of this, PReMo returns approximation not within a given error but rather within an absolute/relative ***tolerance***, which is the absolute/relative difference between two consecutive iteration steps. PReMo could be implemented with arbitrary precision arithmetic, but it would slow its performance significantly, since very often the size of the rational values grows exponentially in the number of, e.g., Newton's method iterations performed.

The rest of the chapter is organized as follows. In Section 5.2 we discuss the internal workings and structure of PReMo: parsers and input models supported, GUI, what kind of numerical algorithms are implemented and their implementation issues. Later, in Section 5.3 we present the experimental performance of PReMo on various case studies, including grammars used in Natural Language Processing and a toy Quicksort model. In Section 5.4 we compare PReMo with SMCSolver, a solver for QBDs that implements their specific most efficient known methods. We conclude in Section 5.5 and list some possible future extensions to PReMo.

PReMo (version 1.3) is available for download for the main operating systems at: groups.inf.ed.ac.uk/premo

[2] M-J. Neiderhof and G. Satta also implemented this method ([NS06]) based on [EY05s] and used it for stochastic parsing in NLP. However, they did not exploit the sparsity of the transition matrix by using a sparse linear solver to perform each iteration of Newton's method. We would like to thank them for pointing us in the direction of large SCFG libraries used in NLP.

5.2 Tool description

Thanks to the fact that PReMo is implemented entirely in Java, it was easy to adapt it to run on the three main operating systems: Windows, Linux and MacOS. PReMo is composed of the following four main components:

1. Parsers of several text input formats that allow for specifying RMCs, RSSGs, SCFGs and p1CAs. Also, a parser for general monotone systems of nonlinear equations including, e.g., exponentiation and logarithm operators.

2. A menu-driven GUI, with an advanced syntax-aware text editor. Through the menu the user has a direct access to many available options of the tool and can run directly any of the numerical algorithms and adjust its parameters.

3. A graphical depiction generator for RMCs and RSSGs, which produces an output in the dot format. Furthermore, if Graphviz([GN99]) is installed on the user's computer, a visualization of an RMC/RSSG in a PS/JPEG/GIF format can be exported directly from PReMo.

4. *Optimized solvers:* Several solvers are implemented for computing the smallest nonnegative solution of a monotone system of equations. Approximate solutions of such equation systems can approximate the termination probabilities, expected termination times and steady state distributions[3] of our models.

We will now briefly describe each of these parts.

5.2.1 Parsers

PReMo supports the following input formats: RMC, RSSG, SCFG, QBD (a.k.a. p1CA), and an arbitrary equation system input format. Parsers for the first four of them were generated using the JavaCC tool (javacc.dev.java.net) from description files consisting of productions in an extended Backus-Naur form. For parsing an arbitrary equation system we rely on Java Expression Parser (JEP) (www.singularsys.com/jep). We will now briefly describe each input language and what analyses PReMo supports for the model corresponding to such a language. A formal grammar of all these input languages, along with a step by step PReMo usage guide, can be found in PReMo's manual [Woj06]. Notice that all these input languages have a support for specifying the

[3] This feature is only available for p1CA/QBD models.

```
A(2,2);
B(1,2);

A {
    L1(A);
    L2(B);
    entry 0:
        0.5: goto L3; 0.5: call L2(0);
    entry 1:
        0.3: call L1(0); 0.7: call L1(1);
    L1 {
        exit 0:
            0.8: goto L3; 0.2: call L2(0);
        exit 1:
            1.0: return 1;
    }
    L2 {
        exit 0:
            1.0: goto L3;
        exit 1:
            1.0: return 0;
    }
    L3 {
        0.5: return 0; 0.5: return 1;
    }
}

B {
    L4(A);
    entry 0:
        0.3: call L4(0); 0.3: call L4(1); 0.4: goto L5;
    L4 {
        exit 0:
            0.5: return 0; 0.5: return 1;
        exit 1:
            0.5: return 0; 0.5: goto L5;
    }
    L5 {
        0.4: goto L5; 0.6: return 1;
    }
}
```

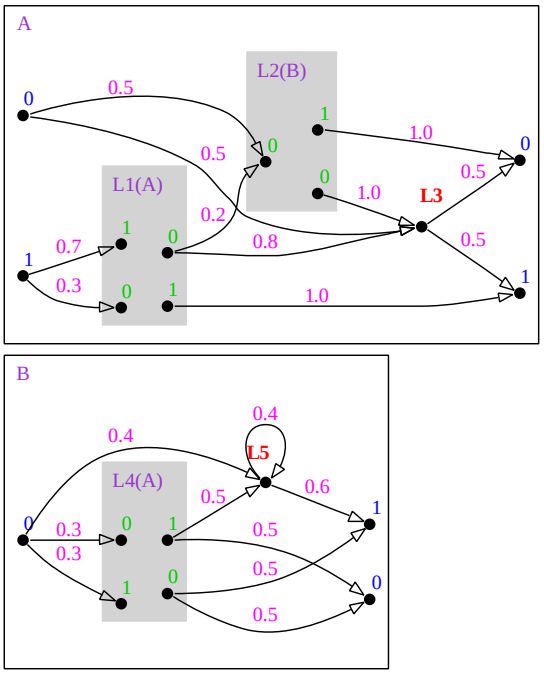

Figure 5.1: Source code of an RMC, and its visualization generated by PReMo.

probabilities as a floating-point number, e.g., `0.3333`, as well as a rational number, e.g., `1/3`.

5.2.1.1 RMCs

We can see an example textual description of an RMC along with its graphical depiction generated by PReMo in Figure 5.1. The PReMo syntax for defining RMCs is as follows: first we declare all components (procedures), using two numbers as parameters that denote the number of entries and exits (for instance in our example `B(1,2);` declares "Component B has 1 entry and 2 exits"). Next, we define the components. Each component definition starts with a declaration of all the boxes contained in that component, together with the information concerning which component they are mapped to (e.g., `L2(B);` declares a box named L2 that is mapped to component B). Next, for all entries, internal nodes, and exits of the boxes, we specify a list of transitions available from that control state. A single transition is a probability value followed by a `goto`, `call` or `return` instruction. We use the `goto` instruction when the transition leads to

an internal node of the same component. A `call name_of_the_box(entry_number)` instruction is used when we want to invoke a component that is mapped to a box labeled name_of_the_box with a parameter entry_number. Finally, we use `return i;` when we want to exit the current component and return value i, in other words, reach the i-th exit node of the current component we are inside of.

PReMo can generate equations for computing the termination probabilities of an RMC and in the case of 1-exit RMCs, it can also generate equations for computing the expected total reward (e.g., the expected time of termination).

5.2.1.2 RMDPs and RSSGs

We can see an example source code of an RSSG with its transition graph generated by PReMo in Figure 5.2. The specification of an RSSG is almost the same as for an RMC except that any entry, internal node or a box exit can be preceded by a name of one of the players who owns it, min or max, in square brackets (e.g., `[max] L1 {...}` declares that the internal node L1 is controlled by the maximizer player). In the transition graph, nodes controlled by the *maximizer* are represented by red dots, the nodes controlled by the *minimizer* are represented by blue dots and the probabilistic nodes, just like it was for RMCs, are represented by black dots.

PReMo supports analysis of the optimal termination probability and the expected time of termination for 1-exit RMDPs and 1-exit RSSGs. Unfortunately, computing or even approximating the optimal termination probabilities for general multi-exit RMDPs and multi-exit RSSGs is undecidable ([EY05i]). Also, even deciding if the optimal reward is ∞ or not in multi-exit RMDPs and multi-exit RSSGs with positive rewards is undecidable (Chapter 4 and [EWY08i]).

5.2.1.3 SCFGs

To specify an SCFG as an input the user writes down a set of productions, one on each line. Each production specifies first the probability of this production happening, then which nonterminal this production rewrites followed by `->` and the yield of this production – a sequence of terminals and nonterminals. The user does not have to specify which symbols are terminals since any symbol without any associated production is assumed to be a terminal symbol by default. An example SCFG is shown in Figure 5.3.

PReMo can compute for a given SCFG the probability that it generates a finite word (in terms of RMCs: the probability of termination) and the expected number of gram-

```
A(2,2);
B(1,2);

A {
    L1(A);
    L2(B);
    [min]entry 0:
        goto L3; call L2(0);
    entry 1:
        0.3: call L1(0); 0.7: call L1(1);
    L1 {
        exit 0:
            0.8: goto L3; 0.2: call L2(0);
        exit 1:
            1.0: return 1;
    }
    L2 {
        exit 0:
            1.0: goto L3;
        [max]exit 1:
            return 0; return 1;
    }
    L3 {
        0.5: return 0; 0.5: return 1;
    }
}

B {
    L1(A);
    entry 0:
        0.3: call L1(0); 0.3: call L1(1); 0.4: goto L2;
    L1 {
        [min]exit 0:
            return 0; return 1;
        exit 1:
            0.5: return 0; 0.5: goto L2;
    }
    [max]L2 {
        goto L2; return 0;
    }
}
```

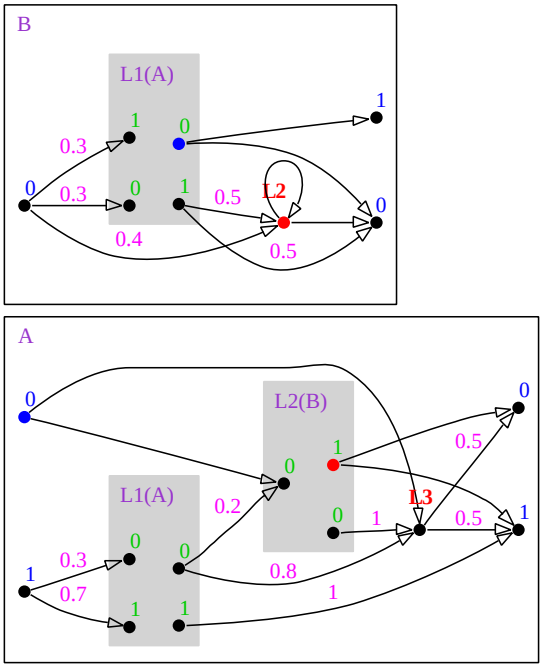

Figure 5.2: Source code of an example RSSG, and its underlying transition graph.

```
0.5 A -> B C D
0.5 A -> A A
0.7 B -> C
0.3 B -> A
0.4 C -> B
0.6 C -> D
```

Figure 5.3: Source code of an example SCFG. The only terminal symbol is D and the nonterminal symbols are A, B and C.

mar rules applied before termination. Note that the last value is equal to ∞ if the probability of termination is strictly less than 1, but the opposite does not hold.

5.2.1.4 p1CAs/QBDs

To input a p1CA/QBD the user specifies for each control state all its outgoing transitions. A rate can be assigned to each transition, not just a probability. These rates are later normalized for each control state to sum up to 1 in order to obtain the embedded discrete-time Markov Chain. Furthermore, each transition specifies whether the

```
a ->
  1.5 + b (2)
  1.5 = a (3)
b ->
  1/2 + b (5)
  1/2 - b
```

Figure 5.4: Source code of an example p1CA.

counter is increased (+), decreased (-) or retained (=) by this transition and the target control state. Finally, each transition has a reward associated with it (equal to 1 by default), which can be redefined by specifying its new value in brackets.

PReMo can approximate for a given p1CA/QBD the probability of reaching a given control state with an empty queue when starting in a given control state with one client in the queue (the so-called G matrix, i.e., the termination probabilities). In order to do that PReMo first applies and outputs the result of a decision procedure for determining which of these variables have a nonzero value, i.e., which control states can be reached with a nonzero probability with the counter value 0, when starting from a given state with the counter equals to 1. Furthermore, PReMo can compute the expected time of termination and the second moment of this time. This allows to compute the standard deviation of this time in a standard way. Moreover, PReMo can compute steady-state probabilities as described in Section 2.2.7 of Chapter 2.

5.2.1.5 Arbitrary equation systems

Here the user defines a list of equations separated by a new-line character. For each variable, its equation defines an expression that this variable has to fulfill, e.g., `x = sin(y) + 1/2*x; y = cos(y)` defines a fixed point equation system for variables x and y. Variable x depends both on variables y and x, while y depends only on the value of itself. We rely on Java Expression Parser (JEP) to parse and interpret each such equation. For the formal grammar of well-formed expressions see www.singsurf.org/ /djep/html/grammar.html and for the list of supported standard math functions see www.singsurf.org/djep/html/op_and_func.html. We added to JEP a support for functions max/min with an arbitrary number of arguments, whose value is the maximum/minimum of its arguments. Obviously any expression that uses such a function is not differentiable and Newton's method cannot be applied to it directly. However, when a suitable strategy improvement algorithm is applied to such an equation system, it can remove all $\min/\max$ operators from the equations and make Newton's method be applicable again (for details see Section 5.2.6).

5.2.2 Tool interface

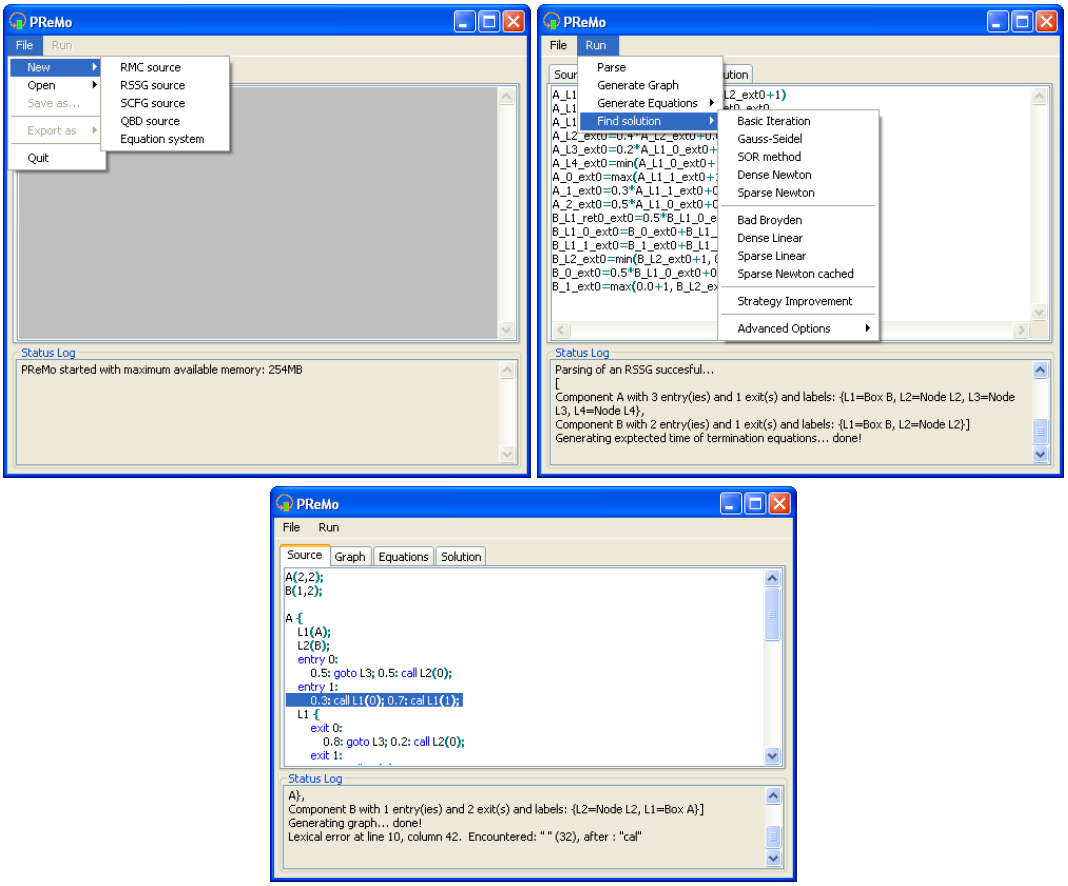

Figure 5.5: On the left a screenshot of choosing one of the supported models in PReMo, on the right a generated equation system along with a list of different numerical solvers available in PReMo for use and at the bottom the text editor that supports syntax highlighting and parsing errors indication.

PReMo's GUI was built using SWT (Standard Widget Toolkit) that was created for the Eclipse IDE as a much more efficient replacement of Java's standard Swing widget toolkit and allows to build user interfaces with a native look and feel. We can see few screenshots of PReMo when run in Windows XP in Figure 5.5. PReMo's GUI was designed using a free version of WindowBuilder (www.instantiations.com/windowbuilderpro). The problem with SWT is that the operating system's specific SWT native library has to be first preinstalled on the user's computer in order to run any application that uses SWT. In order to save the user some trouble and keep PReMo just as a single JAR (Java ARchive) file we used a trick by Dan Rubel that was used in the SWTLoader library (www.moioli.net/files/SWTLoader.zip). We basically automatically extract to the

current directory the native library from inside PReMo's JAR file and invoke Java Virtual Machine again but this time with the native library loaded. We also use this opportunity to increase the amount of memory available to PReMo to 256MB since the default is much too low. Increasing the amount of available RAM improves PReMo's performance significantly since otherwise Java's time-consuming garbage collector is invoked very often.

Furthermore, PReMo contains a source code editor that supports auto-indentation, syntax-highlighting, and highlighting the lines that contain parsing errors once they occur. The available options in the menu allow the user to load and save source code, generate visual depiction of RMCs and RSSGs, generate their corresponding equation systems, find their approximate solutions using various numerical algorithms and adjust their parameters, and after that obtain their performance data, and export the results to external files.

5.2.3 Graphical depiction

Dot is a language for defining graphs in the GraphViz tool. This tool performs auto-placement of node and edges, based on an advanced physical spring model, in order to create a non-cluttered graph and allows to export them in to various graphical formats. In order to generate a dot file from a RMC/RSSG model we used the Apache Velocity (velocity.apache.org) library which allowed us to easily write a parameterized text file template in a programming language tailored to this task. We wrote two different templates so that the user can choose between two different graphical layouts. However, Velocity handles very poorly white-characters in the text files and so in order for the output dot source file to be human-readable, we implemented a module that automatically indent text output streams. If Graphviz[GN99] is already installed on the user's computer, a dot file can be automatically exported from PReMo in a PS/JPEG/GIF format. Example dot files turned into a PNG graphic can be seen in Figure 5.1 and 5.2.

5.2.4 Optimized solvers for min-max-polynomial equations

The core numerical computation for all the analyses provided by PReMo involves solving a monotone system of nonlinear min-max equations. Namely, we have a vector of variables $\mathbf{x} = (x_1, \ldots, x_n)$, and one equation per variable of the form $x_i = P_i(\mathbf{x})$, where $P_i(\mathbf{x})$ is a monotone polynomial-min-max expression with rational coefficients. In vec-

tor notation, this system of equations can be denoted by $\mathbf{x} = \mathbf{P}(\mathbf{x})$. The goal is to find its Least Fixed Point (LFP) solution, i.e., the least non-negative vector, $\mathbf{q}^* \in \mathbb{R}^n_{\geqslant 0}$, such that $\mathbf{q}^* = \mathbf{P}(\mathbf{q}^*)$. (In fact such $\mathbf{q}^*$ is equal to $\lim_{k\to\infty} \mathbf{P}^k(\mathbf{0})$ if all P_i-s are polynomial-min-max expressions.) Notice that the values we are looking for can be irrational and even not expressible by radicals, and so instead of computing them exactly we try to approximate them to within some given error. In brief, the solvers in PReMo work as follows (see [EY05s] for more background): First, we decompose the equations into SCCs (Strongly Connected Components) and calculate the solution "bottom-up", solving the bottom SCCs first and plugging in the solution as constants into higher SCCs. To solve each SCC, PReMo provides several methods:

Value iteration – nonlinear Jacobi

Jacobi, or basic iteration, just computes $\mathbf{x}^0 = \mathbf{0}, \mathbf{x}^1, \mathbf{x}^2, \ldots$, where $\mathbf{x}^i = \mathbf{P}(\mathbf{x}^{i-1})$.

Nonlinear Gauss-Seidel method

Gauss-Seidel iteration is a slight optimization of Jacobi iteration: inductively, having computed x_j^{k+1} for $j < i$, let $x_i^{k+1} := P_i(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, x_i^k, x_{i+1}^k, \ldots, x_n^k)$. In other words, we use a new value of a variable as soon as it is computed for the current iteration. The order in which we consider the variables plays an important role; in extreme cases we observe a ten fold speed-up in the total running time for the same equation system. We observed that in practice for random equation system instances (see Section 5.3.2) this simple technique allows us to converge twice as fast as the Jacobi method.

Successive Overrelaxation (SOR)

SOR is an "optimistic" modification of Gauss-Seidel, which isn't guaranteed to converge in our case. Suppose we are at the k-th step of the iteration with an approximate solution $\mathbf{x}^k$. Let $\overline{\mathbf{x}}^{k+1}$ denote the new assignment to the variables as if one step of the Gauss-Seidel method was applied to the vector $\mathbf{x}^k$. The SOR method with parameter $\omega \in (0,2)$ takes as the next approximation a vector equal to $\omega \cdot \overline{\mathbf{x}}^k + (1-\omega) \cdot \mathbf{x}^k$. This method is widely used for linear equation systems and often significantly increases the efficiency of finding their solutions. However, in our nonlinear case we did not observe any significant advantages. A possible explanation of this fact is the following: if the Gauss-Seidel method converges fast to the solution of a given example then SOR method, which tries to increase the convergence rate, will "overshoot" and the con-

secutive approximations will jump between an approximation that is strictly greater than the actual solution and an approximation that is strictly smaller. Furthermore, each iteration of SOR costs a little bit more than Gauss-Seidel.

We checked whether the SOR method with different values of the parameter $\omega \in (1,2)$ (for larger values SOR always diverges) can help to increase the convergence rate of the termination probability equations for the gadget presented in Figure 4.2, where $p_1 = \frac{1}{2}$. The corresponding equation system is equivalent to a single equation $x = \frac{1}{2}x^2 + \frac{1}{2}$. The termination probability for this example is equal to 1, but the expected number of steps to termination is ∞ and the Gauss-Seidel method needs $\Theta(2^i)$ iterations in order to approximate x to within 2^{-i} additive factor as was shown in [EY05s]. We examined the total running time with different absolute tolerance stoppage criteria of $10^{-4}, 10^{-5}, \ldots, 10^{-14}$. As we can see from Table 5.1, we still need an exponential number of steps even for ω being as close to 2 as 1.999 and there is just a small difference between the running times of the Gauss-Seidel method and SOR with such a parameter. Furthermore, SOR can already be unstable with a small value of ω (see Table 5.2 in Section 5.3.1). In conclusion, if we need to use a basic iterative solver in PReMo, Gauss-Seidel is almost always the best choice.

Dense and sparse decomposed Newton's method

Newton's method attempts to compute a solution to $F(\mathbf{x}) = \mathbf{0}$. In n-dimensions, it works by iterating $\mathbf{x}^{k+1} := \mathbf{x}^k - (F'(\mathbf{x}^k))^{-1}F(\mathbf{x}^k)$ where $F'(\mathbf{x})$ is the ***Jacobian*** matrix of partial derivatives of F. In our case we apply this method to $F(\mathbf{x}) = P(\mathbf{x}) - \mathbf{x}$. It was shown in [EY05s] that if the system is decomposed into SCCs appropriately, convergence to the LFP is guaranteed if we start with $\mathbf{x}^0 = \mathbf{0}$. The expensive task at each step of Newton's method is the matrix inversion $(F'(\mathbf{x}^k))^{-1}$. Explicit matrix inversion is too expensive for huge matrices. But this matrix is typically sparse for RMCs, and we are able to handle much larger matrices if instead of inverting $(F'(\mathbf{x}^k))$, we solve the following equivalent sparse linear system of equations: $(F'(\mathbf{x}^k))(\mathbf{x}^{k+1} - \mathbf{x}^k) = F(\mathbf{x}^k)$ to compute the value of $\mathbf{x}^{k+1} - \mathbf{x}^k$, and then add $\mathbf{x}^k$ to obtain $\mathbf{x}^{k+1}$. Our Dense Newton's method uses LU decomposition to invert $(F'(\mathbf{x}^k))$. The Sparse Newton's method uses at each step a sparse linear iterative solver to find the next approximation of the solution. We used sparse linear equation solvers implemented in the MTJ (Matrix Toolkit for Java) library (described in Section 5.2.5) that provides us with the following solvers:

- Biconjugate Gradient

- Biconjugate Gradient Stabilized
- Iterative Refinement (preconditioned Richardson method) (see, e.g., [Var62])
- Conjugate Gradients Squared
- Quasi Minimal Residual

The user can choose to use any of these five solvers from the *Advanced options* menu in PReMo. According to our performance tests the Iterative Refinement was the fastest on the examples that we tried. However, if the system of equations is ill-conditioned at any step of the approximation process then the convergence of this method is either very slow or it does not converge at all. In such a case the user should choose the Biconjugate Gradient method that does not have this problem and is just a little bit slower on the average.

Broyden method

The Broyden method, first described in [Bro65] is a generalization of the secant method to multiple dimensions. Instead of computing the Jacobian matrix at each step like in Newton's method, it approximates the Jacobian matrix based on its previous value. If J_k denotes the Jacobian matrix at the k-th step of Broyden's method then we approximate it by $J_k = J_{k-1} + \frac{\Delta F_k - J_{k-1}\Delta x_k}{||\Delta x_k||^2}\Delta x_k^T$ and then the method proceeds just like Newton's method: $x_{k+1} = x_k - J_k^{-1}F(x_k)$. Broyden also suggested updating the inverse of the Jacobian directly: $J_k^{-1} = J_{k-1}^{-1} + \frac{\Delta x_k - J_{k-1}^{-1}\Delta F_k}{\Delta x_k^T J_{k-1}^{-1}\Delta F_k}(\Delta x_k^T J_{k-1}^{-1})$. This method is commonly referred to as the "good Broyden's method". By using a slightly different modification to J_{k-1} we get the so-called "bad Broyden's method": $J_k^{-1} = J_{k-1}^{-1} + \frac{\Delta x_k - J_{k-1}^{-1}\Delta F_k}{\Delta F_k^T \Delta F_k}\Delta F_k^T$. Notice that the bad Broyden's method requires less arithmetic operations than the "good" one but it updates the inverse of the Jacobian in a less precise way. According to our experiments the good and the bad Broyden's methods are never faster in practice than the Sparse Newton's method and sometimes even slower than Gauss-Seidel. The possible explanation is the fact that most of the savings in using Broyden's method comes from not having to recompute the Jacobian matrix over and over again at each step of the iteration. However, in our case we compute the Jacobian matrix symbolically just once at the very beginning and computing it on later stages requires us just to evaluate some polynomial expressions which is very fast to perform.

tolerance	Gauss Seidel	SOR $\omega = 1.2$	SOR $\omega = 1.999$	Dense Newton
10^{-4}	0.010(136)	0.006(124)	0.000(2)	0.001(13)
10^{-5}	0.022(441)	0.021(402)	0.000(2)	0.001(16)
10^{-6}	0.049(1407)	0.081(1283)	0.000(2)	0.002(19)
10^{-7}	0.092(4463)	0.087(4074)	0.007(1164)	0.002(23)
10^{-8}	0.105(14132)	0.127(12900)	0.035(8002)	0.002(26)
10^{-9}	0.208(44710)	0.183(40814)	0.131(29629)	0.002(27)
10^{-10}	0.731(141409)	0.231(129087)	0.155(98023)	0.001(27)
10^{-11}	0.947(447202)	0.648(408236)	0.492(314303)	0.002(27)
10^{-12}	1.849(1414194)	1.974(1290916)	1.651(998148)	0.001(27)
10^{-13}	5.694(4472044)	6.241(4080682)	4.932(3160483)	0.001(27)
10^{-14}	17.673(14089115)	19.732(12766680)	15.075(9925181)	0.001(27)

Table 5.1: The results of running Gauss-Seidel, the SOR method with different ω values and the Dense Newton's method on the equation $x = \frac{1}{2}x^2 + \frac{1}{2}$. The first column of each row specifies the target absolute tolerance to be achieved, and then the running time in seconds is given for each solver including, in parentheses, the number of iterations that solver required. Notice that SOR always needs less iterations that Gauss-Seidel, but the difference in its running time is not that significant, since each of its iterations takes longer time than one step of Gauss-Seidel. This clearly can be observed for tolerance 10^{-14}: SOR with $\omega = 1.2$ runs 2 seconds slower than Gauss-Seidel although it needs to perform 10% less iterations. As we can see, the SOR technique does not provide any significant speedup unless the parameter ω is very close to 2, and for small values of this parameter its running time is even greater than Gauss-Seidel's. Moreover, even for small values of ω the SOR method can be unstable (see Table 5.2 in Section 5.3.1) and for $\omega = 2$ the SOR method is always unstable. For comparison, in the last column we can see the running time of the Dense Newton's method, which is negligible compared to all the other methods. (In this particular example, Newton's method after 27 iterations obtains all the available bits of precision in the double precision arithmetic.)

Dense and sparse linear solver

These solvers are highly optimized solvers that are applicable to linear equations only. Equations like that can occur, e.g., for 1-exit RMCs with positive rewards. The Sparse Linear Solver in PReMo can use any of the sparse linear solvers mentioned in Section 5.2.4. Both of these solvers handle SCCs with a single variable in a special way since such SCCs are trivial to solve in the setting where all expressions are linear. Moreover, Dense Linear Solver is implemented to be robust. It detects for each SCC whether in the corresponding equation system $\mathbf{x} = A\mathbf{x} + b$, the matrix $I - A$ is singular or not and if

it is then all the variables in such an SCC are set to ∞. In the case when the matrix $I-A$ is invertible, but one of the entries of the vector $(I-A)^{-1}b$ is negative, ∞ is assigned to all the variables as well. In all other cases, $(I-A)^{-1}b$ is the only nonnegative finite solution of the equation $\mathbf{x} = A\mathbf{x} + b$, and so, also its LFP. This fact is not obvious and requires a proof which is provided in the appendix of this chapter. Although checking the nonnegativity of $(I-A)^{-1}b$ is performed when using the Sparse Linear Solver, checking whether $I-A$ is singular or not is not done. The reason is that iterative algorithms for sparse matrices are unreliable in detecting nonsingular matrices due to the approximation errors involved in their computation. Of course it is possible to check nonsingularity of a matrix via its dense representation, but such a check would be more costly than computing the approximation itself and the benefit of having sparse matrix representation would disappear.

5.2.5 PReMo's implementation

Thanks to the fact that PReMo is implemented in a object oriented way, a big amount of common code, e.g. initialization and performance analysis, is shared between all the numerical solvers. Since these solvers can only converge to within some interval of the actual solution, PReMo provides different mechanisms for the user to choose when to stop the approximation process: absolute tolerance, relative tolerance, and a specified number of iterations. The first two are parameterized with a tolerance value ε:

- absolute tolerance – stop when $(\left\|\mathbf{x}^k - \mathbf{x}^{k-1}\right\|_\infty < \varepsilon)$
- relative tolerance – stop when $(\left\|\mathbf{x}^k - \mathbf{x}^{k-1}\right\|_\infty / \left\|\mathbf{x}^{k-1}\right\|_\infty < \varepsilon)$
- stop after some constant number of iterations

None of them can actually guarantee how far the returned approximation is away from the actual solution and this is why we refer to ε as ***tolerance***, not an ***error***. Nevertheless, these stopping criteria are most commonly used in practice and behave well. Moreover, each solver has a maximum number of iterations it can perform after which the current approximation method is declared to be not convergent. This number is different for basic iterative solvers like Gauss-Seidel and for Newton based methods.

For parsing mathematical expressions we used a free version of Java Math Expression Parser (JEP) library (www.singularsys.com/jep/) and its extension that implements a symbolic differentiator for a very broad class of mathematical expressions. However,

this differentiator turned out to be too slow and memory consuming for most of our purposes and it could not handle long expressions that occurred, e.g., for SCFGs derived from the Penn Treebank corpora (Section 5.3.1). As a result we had to replace it with our own implementation of a symbolic differentiator that is specifically tailored to only polynomial expressions that we are mostly dealing with. Our implementation turned out to be ten times faster than the differentiator in JEP. Nevertheless, the user can still choose which differentiator he prefers to use. This allows to use Newton's method for a more general class of equation systems in the arbitrary equation system input mode.

Further performance improvements in PReMo can potentially be obtained by implementing our own parser and expression evaluator tailored to only min-max-polynomial expressions instead of using the JEP library for this purpose. This has not yet been done in PReMo, but there is a special parser, an evaluator, and a differentiator, for linear expressions only, implemented in PReMo. This greatly improves the efficiency of analyzing (best/worst) expected time of termination of 1-exit RSSGs. There are two special solvers that use them, Dense Linear solver and the Sparse Linear solver, which are analogs of Dense and the Sparse Newton's method. These methods converge in just one step to the actual solution and although they essentially perform just one iteration of Dense and the Sparse Newton's method respectively, they are much more efficient, because they are tailored to linear equation systems only.

The decomposed Newton's method relies on the fact that the equation system is firstly decomposed into SCCs. For the value iteration methods the user can choose whether to decompose the equation system into SCCs or not, but as it turned out from our experiments significantly fewer steps overall are often needed in order to converge for the decomposed system. We found all available Java libraries that can perform SCC graph decomposition too inefficient and unreliable for the size of half a million nodes that we considered during our experiments. We thus implemented our own highly efficient SCC graph decomposer using a recursion-free Depth First Search. Our implementation takes less than a second for half a million variables. Therefore, there is essentially no reason for the user to switch the SCC decomposition off.

For handling dense and sparse matrix operations we used the Matrix Toolkits for Java (MTJ) library (ressim.berlios.de). MTJ implements all standard algebraic operations for matrices represented in dense and various sparse formats. The library's implementation and its API is based on the BLAS interface and Fortran's numerical library LAPACK (Linear Algebra PACKage). As an option, MTJ can use machine-

optimized BLAS libraries (such as ATLAS) for improved performance when dealing with dense matrix operations. However, we did not use ATLAS in PReMo, although it could have improved its performance significantly on the large dense examples that we describe in Section 5.4. Instead, we used the default Java implementation of BLAS and LAPACK in MTJ and this allowed for an easy portability of PReMo to many operating system platforms.

The solvers' testing module collects extensive information about each run of a solver and the input data. For each input equation system, it records the number of SCCs after decomposition and the size of the biggest SCC. Then, for each particular numerical algorithm, it computes the number of iterations performed in total for all SCCs, records the size of the SCC that required the largest number of iterations and their number. Furthermore, it records the time spent by the solver in the computation phase and the initialization phase, and for methods that involve differentiation also the time spent computing the Jacobian matrix symbolically. The module can also automatically generate a LaTeX table with all the recorded data about each individual run and generate data plots for gnuplot (www.gnuplot.info). It also compares the returned approximations of different solvers to each other in order to verify whether they return the same solution within a given tolerance.

Java's garbage collector introduces a bias into our experimental running times. If it starts in the middle of computation of a numerical method then the computations are stopped completely for a while and this can alter significantly the performance of that algorithm. In order to tackle this problem, we performed all our tests with the maximum possible amount of memory available to PReMo. We also invoke the garbage collection manually before each solver starts and take an average from multiple tests (more than a hundred if possible). This reduces significantly the influence of the garbage collector on the solvers' performance.

Before Newton's method can be applied to an equation system, we first need to remove the variables that are equal to 0 in its LFP. However, instead of writing a special routine for this purpose, we perform for each SCC one iteration of the Gauss-Seidel method first. If all the variables in an SCC are equal to zero after that step (i.e., there is no positive constant term in any equation) then we declare all the variables in this SCC to be equal to zero and proceed to the next SCC.

In the strategy improvement method (see the description in Section 5.2.6) we constantly update the equations associated with some of the variables. As a result, the Jacobian matrix is constantly changing as well. If we run the Sparse Newton's method

at each step of the strategy improvement, it will keep recomputing from scratch the Jacobian matrix for the modified set of equations. In order to avoid this costly operation, we implemented a special version of the Sparse Newton's method that keeps the Jacobian matrix stored in the memory even after it has finished all the computations. When using this method, after each step of the strategy improvement, we can modify any of the equations of the equation system and the Jacobian matrix will be automatically updated.

5.2.6 Strategy improvement in PReMo

When considering models with players, like 1-RMDPs/1-RSSGs with positive rewards and terminating 1-RMDPs/1-RSSGs, the underlying system of equations can contain max and min operators. This makes standard Newton's method inapplicable since the equations are not differentiable. However, thanks to the strategy improvement algorithm, even in such cases we can use Newton's method indirectly. Henceforth, we will refer to each variable whose associated equation involves a max/min operator as a ***maximizer/minimizer (player)'s variable***, also referred to as its ***type***. Each max/min operator has several polynomial expressions as its arguments and by definition its value is the value of one of these arguments. The maximizer/minimizer chooses for each maximizer/minimizer's variable which of its associated polynomial expressions is going to be the value of its max/min operator and this defines that player's ***strategy*** for that variable. Notice that, once we fix a strategy for both players, the equation system will contain only polynomial expressions and again a standard Newton's method can be used to approximate its LFP.

We will call a maximizer's/minimizer's variable ***improvable*** if its current value is lower/higher than the current value of one of its other associated expressions. It was shown that, in the case of termination 1-exit RSSGs and 1-exit RSSGs with positive rewards, by switching the strategy at the improvable maximizer's variables to their associated expression with currently the highest value, we will reach the globally optimal strategy in a finite number of steps. The user can also use a strategy improvement method for minimizing 1-RMDPs in PReMo, but such a method may not converge to the correct fixed point.

Furthermore, the user can choose in what way the strategy will be modified at each step: improve only a single variable for which the difference in its value and the value of one of its associated expressions is the largest/smallest (depending on the type of that node), or just pick one of the improvable variables uniformly at random, or maybe

a parallel strategy improvement that switches the strategy for all improvable variables simultaneously at once. Regardless of this, for any single improvable variable, we always switch its strategy to the expression with currently the highest/lowest value (again depending on its type). We implemented the strategy improvement in PReMo in such a way that it is easy to extend it with a new method of picking the variables to improve at each step.

The strategy improvement can be run transparently on top of any other solver or even another strategy improvement algorithm. In fact, the strategy improvement for the maximizer is a separate solver from the strategy improvement for the minimizer. The strategy improvement works by intercepting all expressions with max or min operators (depending on the type of the strategy improvement) and substituting each such expression with one of its arguments. Such a simplified equation system is passed on to the solver that this strategy improvement algorithm runs on top of. The strategy improvement and its underlying numerical solver can have different stopping criteria. To see all the options available to the user in PReMo, see Figure 5.6.

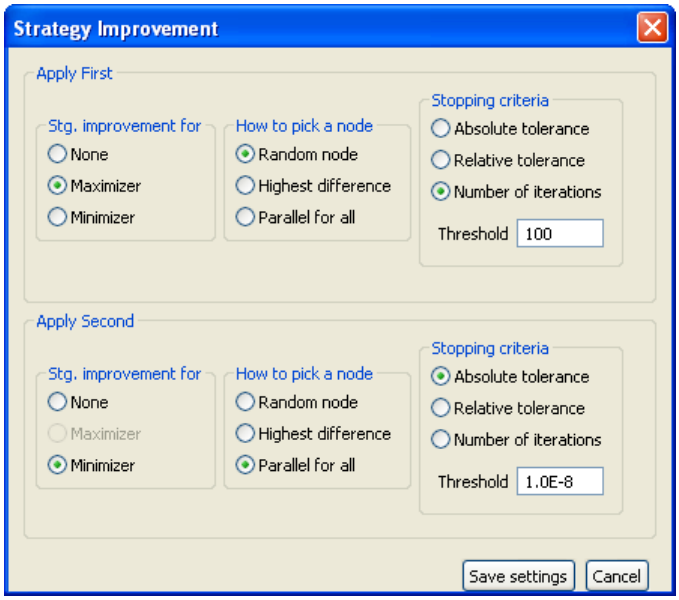

Figure 5.6: Configuration of the strategy improvement algorithm. The strategy improvements for maximizer and minimizer can be applied in any order and with their own stopping criteria. Also, three different methods for strategy updates can be chosen: pick one improvable variable uniformly at random, pick an improvable variable with the highest/lowest (depending on the type of this variable) difference between its current value and one of its associated expressions or switch the strategy in parallel for all improvable variables simultaneously at once.

We will now describe how the simultaneous strategy improvement for the maximizer's variables works in practice. The initial strategy for each variable belonging to the maximizer player is chosen uniformly at random[4]. At each step of the simultaneous strategy improvement we first compute the solution with the strategy fixed for each of the variables. We then switch the strategy for each maximizer's variable whose current value can be improved by more than a small threshold (in our implementation set to 10^{-10}) and we pick as a new strategy one of its associated expressions with currently the highest reward. We again fix the strategy for all the variables, update the expressions for the variables whose strategy was just switched, and then compute the LFP of such an updated system of equation. We repeat this procedure as long as each step increases the reward of at least one of the variables by more than ε (set to 10^{-8} by default). The small threshold, 10^{-10}, as described above, is crucial since otherwise the errors in the floating-point operations can make the strategy improvement algorithm believe that it improves the value of a variable at each step although in fact it is alternating between just two possible strategies for that variable and it would continue to do so forever. For an experimental comparison of a strategy improvement procedure with an iterative solver, Gauss-Seidel, see Section 5.3.5.

5.3 Experimental results

We ran a wide range of experiments on a Pentium 4 3GHz with 1GB RAM, running Linux Fedora 5, kernel 2.6.17, using Java 5.0. We shortly report on some of them here. We used standard floating-point numbers in double precision in all our experiments.

5.3.1 SCFGs generated from the Penn Treebank NLP corpora

We checked the ***consistency***[5] of a set of large SCFGs, with 10,000 to 50,000 productions, used by the Natural Language Processing (NLP) group at University of Edinburgh and derived by them from the Penn Treebank NLP corpora[6].

The results are listed in Table 5.2, size is indicated in # of productions, and max-scc column provides the size of the largest SCC after the corresponding nonlinear equa-

[4]Some heuristic could be used here instead, but this would require us to perform some precomputations on the equation system first.

[5]An SCFG is called *consistent* if starting at any nonterminal in the grammar, a random derivation terminates, and generates a finite string, with probability 1.

[6]Penn Treebank corpora (www.cis.upenn.edu/∼treebank) is a standard benchmark collection of grammatically annotated parse trees of sentences in English that appeared in articles published in various well-known journals, e.g., the Wall Street Journal. These SCFGs were assumed to be consistent by construction. ([DK06])

tion system was decomposed into SCCs. Time is measured in seconds and in parentheses is the number of iterations needed for the biggest SCCs. The stopping condition for the iteration was reaching an absolute tolerance of $\varepsilon = 10^{-12}$. There is a ✓ if the grammar was found to be consistent (to within 10^{-4} error), and ✘ otherwise. Two out of seven SCFGs turned out to be (very) inconsistent, namely those derived from the brown and switchboard corpora of Penn Treebank, with termination probabilities as low as 0.3 for many nonterminals. This inconsistency was a surprise to our NLP colleagues, and was subsequently identified by them to be caused by annotation errors in Penn Treebank itself ([DK06]). Note that both dense and sparse versions of decomposed Newton's method are by far the fastest. Since the largest SCCs have around 1000 vertices, dense Newton also worked on these examples. Most of the execution time of Newton's method at first was in fact taken up by computing all the entries of the Jacobian $F'(\mathbf{x})$. We thus optimized greatly the computation of the Jacobian by implementing our own symbolic differentiator of polynomial expressions (Section 5.2.5).

name	#prod		max-scc	Jacobi	Gauss Seidel	SOR ω=1.05	DNewton	SNewton
brown	22866	✘	448	312.084(9277)	275.624(7866)	diverge	2.106(8)	2.115(9)
lemonde	32885	✓	527	234.715(5995)	30.420(767)	diverge	1.556(7)	2.037(7)
negra	29297	✓	518	16.995(610)	4.724(174)	4.201(152)	1.017(6)	0.499(6)
swbd	47578	✘	1123	445.120(4778)	19.321(202)	25.654(270)	6.435(6)	3.978(6)
tiger	52184	✓	1173	99.286(1347)	16.073(210)	12.447(166)	5.274(6)	1.871(6)
tuebadz	8932	✓	293	6.894(465)	1.925(133)	6.878(461)	0.477(7)	0.341(7)
wsj	31170	✓	765	462.378(9787)	68.650(1439)	diverge	2.363(7)	3.616(8)

Table 5.2: Performance results for checking consistency of SCFGs derived from Penn Treebank. The columns from left to right denote the following: the name of the dataset, the number of grammar rules in the SCFG derived from such a dataset and next to it ✓ if the grammar was "consistent" and ✘ otherwise, the biggest SCC of variables after decomposition of the underlying equation system, and finally, for each numerical algorithm, the time in seconds it took to compute the approximation and in parentheses the number of iterations this algorithm performed for the biggest SCC. The stopping condition was to obtain an absolute tolerance of $\varepsilon = 10^{-12}$. SCFG was declared "consistent" if all nonterminals had termination probability $\geqslant (1 - 10^{-4})$. The SCFGs brown and swbd failed consistency by a wide margin having many nonterminals with termination probability as low as 0.3.

5.3.2 Randomly generated RMCs and 1-RSSGs

We wanted to compare the performance of all numerical algorithms implemented in PReMo on "random" instances of RMCs. However, it is hard to randomly generate an RMC or to define an unbiased distribution on them. Instead, we looked at random quadratic equation systems since they are simpler to generate.

We tested PReMo on a fairly representative sample of equation systems of different sizes, ranging from 10,000 to 500,000 variables. Random generation was done in two ways. We separately generated equation systems that can occur when analyzing 1-RMCs (equivalently SCFGs) and multi-exit RMCs. The first ones are a lot easier to generate. The nonlinear equations we generated were of three possible kinds: (1) $x_i = x_j x_k$, (2) $x_i = p x_j + (1-p) x_k$, and (3) $x_i = p_1 x_j + p_2 x_k + p_3$, where $p_1 + p_2 + p_3 = 1$. Type (1) corresponds to a function call, type (2) to a branch node and type (3) to a branch node for which one of the branches exits the current function while returning a value. We chose type (1) with 0.2 probability, type (2) with 0.6 probability ($p \in (0,1]$ was picked uniformly at random), and (3) with 0.2 probability (and p_1, p_2, p_3, were chosen uniformly in $(0,1]$ and then normalized, dividing by $p_1 + p_2 + p_3$, so they sum to 1). These probability values were chosen more or less arbitrarily. We can see the running times in Figure 5.7. Size was measured in the number of variables, n. The number of random instances generated for each size n, was $(2 \times 10^6)/n$, and running time was averaged over all instances of size n.

On these random large instances, with very high probability most nodes are in one huge SCC with small diameter (by the so-called "small world phenomenon"). Dense Newton's method did not work at all on these huge SCCs, because inverting such large matrices is too expensive. On the other hand, we can see that both Gauss-Seidel and the Sparse Newton's method did very well. In particular, Sparse Newton handled instances with 500,000 variables in ~ 45 seconds. It is worth noting that by using Gauss-Seidel instead of basic iteration, we speed up by a factor of 2 on average.

Newton's method does not apply directly to 1-RSSGs, with nonlinear min-max equations, but Gauss-Seidel does. We applied a similar random generation technique to generate 1-RSSGs (this time with min and max nodes as well) and obtained the results in Figure 5.8. See Section 5.3.5 for a comparison of the Gauss-Seidel method and a method being a combination of the strategy improvement algorithm and a Sparse Linear solver (i.e., a single step of the Sparse Newton's method) for 1-RSSGs with rewards. (Strategy improvement is described in Section 5.2.6, Sparse Linear solvers in Section 5.2.4.)

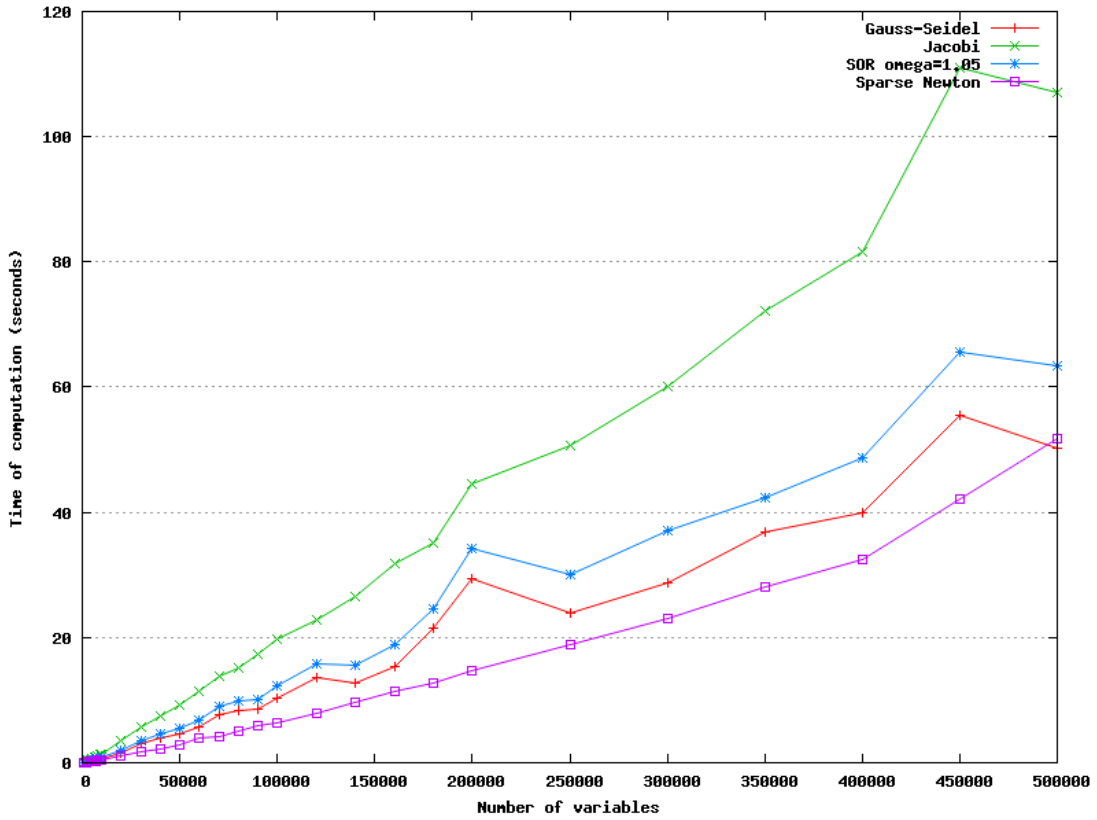

Figure 5.7: Running times of various numerical algorithm for randomly generated 1-RMCs. For each size n that was tried, $(2 \times 10^6)/n$ instances of size n were generated and the running time was averaged. The termination condition was absolute tolerance $\varepsilon = 10^{-12}$.

Finally, for multi-exit RMCs, we had a difficult time finding a direct random generation scheme that was simple to define. We instead chose to randomly generate more general monotone nonlinear polynomial equations, where equations can be of the form $x_i = x_j x_j + x_r$, in addition to the possible equations we generated for 1-RMCs. These equation systems could potentially have no finite LFP solution, in which case the methods diverge to infinity, and create overflow error (or may not be defined, in the case of Newton's method, because the Jacobians may not be invertible). But the equations generated form a superset of the equations for multi-exit RMCs. We generated these equations and tested to see if the methods do converge. Two generated instances were discarded because the results diverged to infinity. The results for the instances that did converge are in Figure 5.9.

To sum up, even though we do not have polynomial time algorithms for calculating the termination probability of any of the tested models, these basic numerical

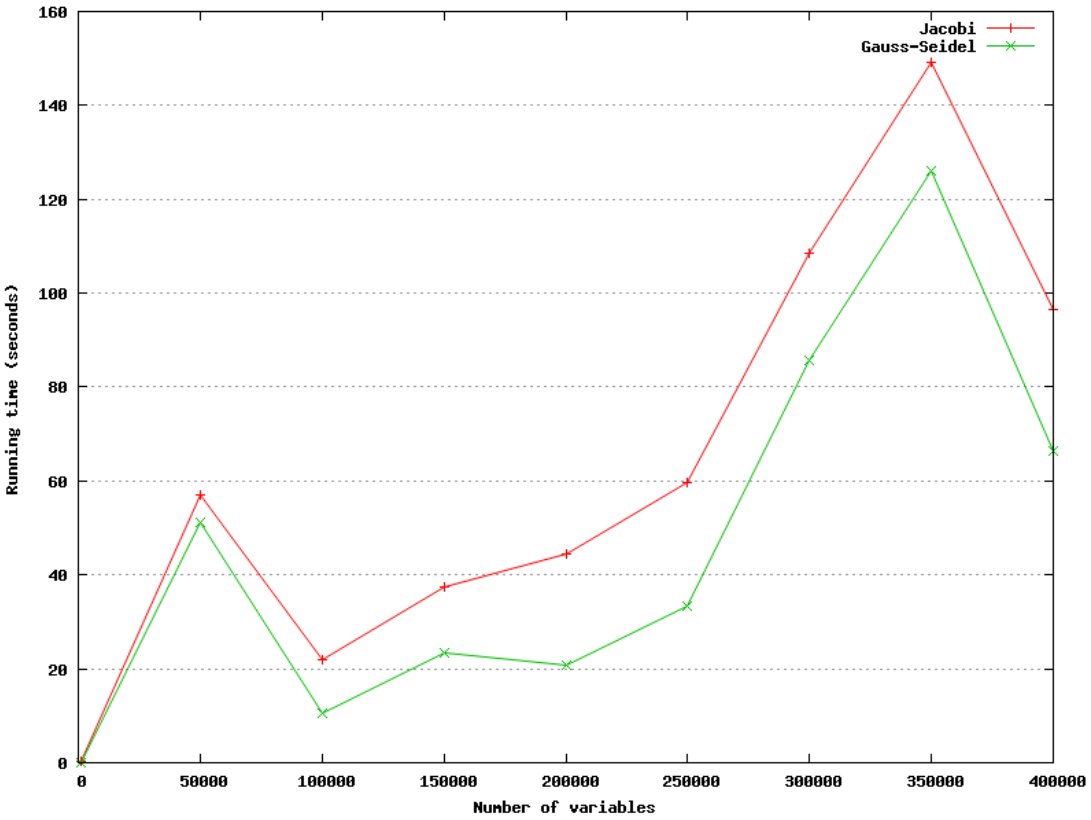

Figure 5.8: Random 1-RSSGs. Average running time for computing termination probability of 2 instances for each size n, $n = 50{,}000 * i$, $i = 1,\ldots,7$, with absolute tolerance $\varepsilon = 10^{-12}$.

iteration methods work reasonably well for relatively large (albeit random) instances.

5.3.3 Quicksort

For expected termination time analysis, we considered a toy model of randomized Quicksort, using a simple hierarchical 1-RMC. In our model we have n components, Q_i, $i = 1,\ldots,n$, corresponding to invocations of Quicksort on arrays of size i. Each such component takes time i to pick the pivot and split the entries. We modeled this as "expected" time i, by having a self-loop at the entry node of component i with probability $(1 - 1/i)$ and transitioning to the "pivot choice" node with probability $\frac{1}{i}$. After this transition is taken, the pivot d is chosen uniformly at random and then we have to recursively solve two instances of Quicksort, of sizes d and $i - d$. We modeled this by random transitions of probability $1/i$, for each $d \in \{1,\ldots,i-1\}$, to a sequence of two boxes labeled by Q_d and Q_{i-d}, and then to the terminal exit of the component.

We computed the *expected time for termination* for these models, to see whether

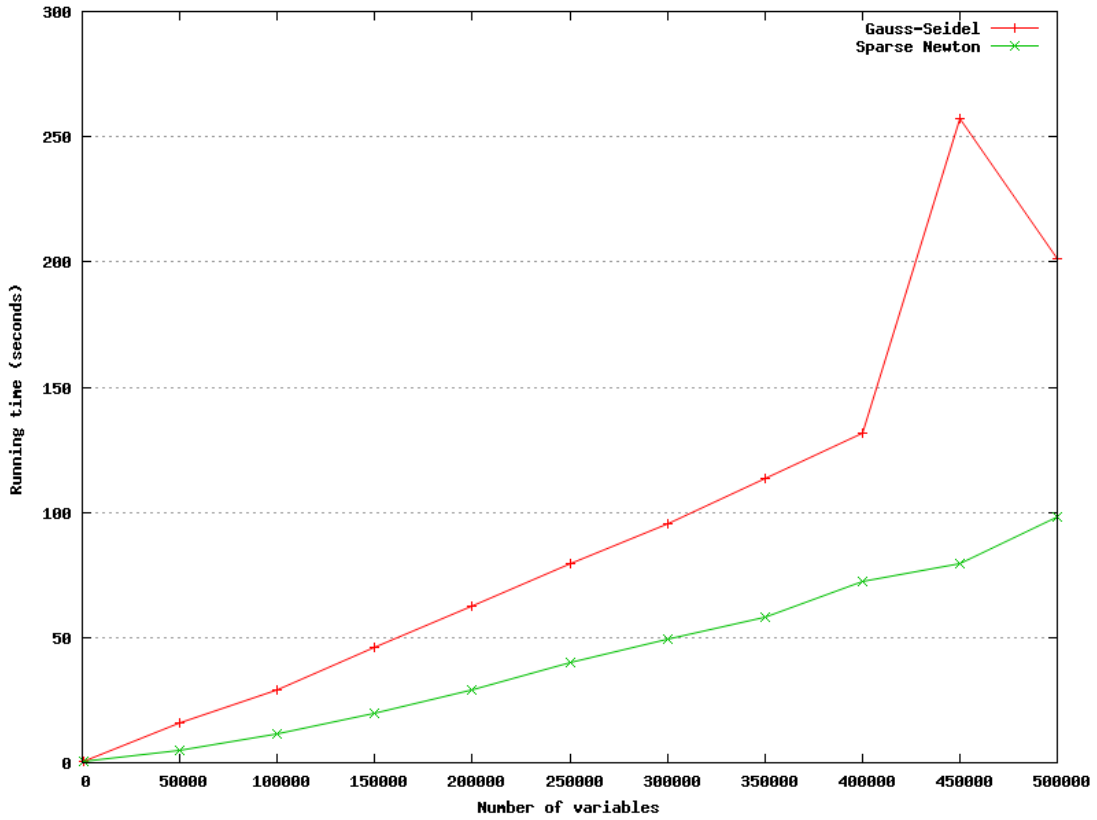

Figure 5.9: Random RMCs and general monotone polynomial systems. Average running time for computing termination probability of 3 instances for each size n, $n = 50,000 * i$, $i = 1,\ldots,10$, with absolute tolerance $\varepsilon = 10^{-12}$. Two generated instances that diverged were discarded.

the expected running time of the algorithm matches the known theoretical average-case analysis of $\Theta(n \log n)$. We also tried letting the pivot node be controlled by the *minimizer* or *maximizer*, and generated the corresponding linear min-max equations for expected running time for such 1-RMDPs, in order to consider best-case and worst-case running times of Quicksort. Our models, as would be expected, matched the known theoretical analysis of running times for (randomized) Quicksort. Namely, the expected running time for random pivot choices is $cn\log_2 n$, the expected running time best pivot choice is $c'n\log_2 n$, and the expected time for the worst pivot choice is $c''n^2$. In our model we found for $n = 500$: $c = 1.85$, $c' = 1.55$ and $c'' = 0.51$. This would suggest that random pivot choice runs 1.19 times slower than the optimal possible choice of pivots.

5.3.4 Long chains

It is easy to construct examples of simple, even finite state Markov chains, where the behavior of Newton's method can in principle be exponentially better than Gauss-Seidel iteration. This occurs, for example, when a finite Markov chain consists of a "long chain" with n nodes $v_1, \ldots, v_n$, where v_n is the terminal state, and where there are transitions of the form $v_i \stackrel{1/2}{\rightarrow} v_{i+1}$ and $v_i \stackrel{1/2}{\rightarrow} v_1$, for $i = 1, \ldots, (n-1)$. In these examples, clearly the probability of termination from all nodes v_i is 1. Let x_i^j be the value of basic value iteration (Jacobi) after j iterations, starting at 0, for a variable x_i representing the probability of termination from v_i. It is easy to show that, in order x_1^j to be $\geqslant 1/2$, it must be the case that $j \in 2^{\Omega(n)}$. On the other hand, Newton's iteration on Markov chains converges in one iteration, because the Jacobian is a constant, invertible, matrix. We ran Jacobi, Gauss-Seidel, and both Dense and Sparse Newton iteration on such long chains. Interestingly, although Dense Newton performed as expected, solving the required single iteration in a very short time (for the sizes where it could handle the matrix inversion), the Sparse Newton's iteration encountered numerical problems with all five sparse linear solvers available for use in PReMo. It appears that the small probabilities that arise in solving this linear system causes problems for the available sparse linear solvers. The results of the running times for Jacobi, Gauss-Seidel, and Dense Newton, are in Figure 5.10.

5.3.5 Simultaneous strategy improvement vs value iteration

We tested the performance of the strategy improvement with the Sparse Newton's method vs the Gauss-Seidel method on max-linear equation systems with 10 000, 20 000, ..., 100 000 variables. For each of the sizes, we randomly generated one hundred tests and ran both algorithms on all of them. About one third of the equations in each single test had a max operator in it. Such equation systems can arise in the context of calculating the worst expected running time of a program modeled as a maximizing 1-RMDP with positive rewards. We created all equation systems in a way that eliminates the possibility of any of the variables having an infinite value in the Least Fixed Point (LFP) solution. This was necessary since none of our numerical procedures checks first whether the values to be computed are infinite or not. Such a check can be performed for linear equation systems but it would be more costly than the main step, i.e., finding an approximate LFP of such an equation system (see Section 5.2.4). For a max-linear system of equations, a possible solution could use an upper

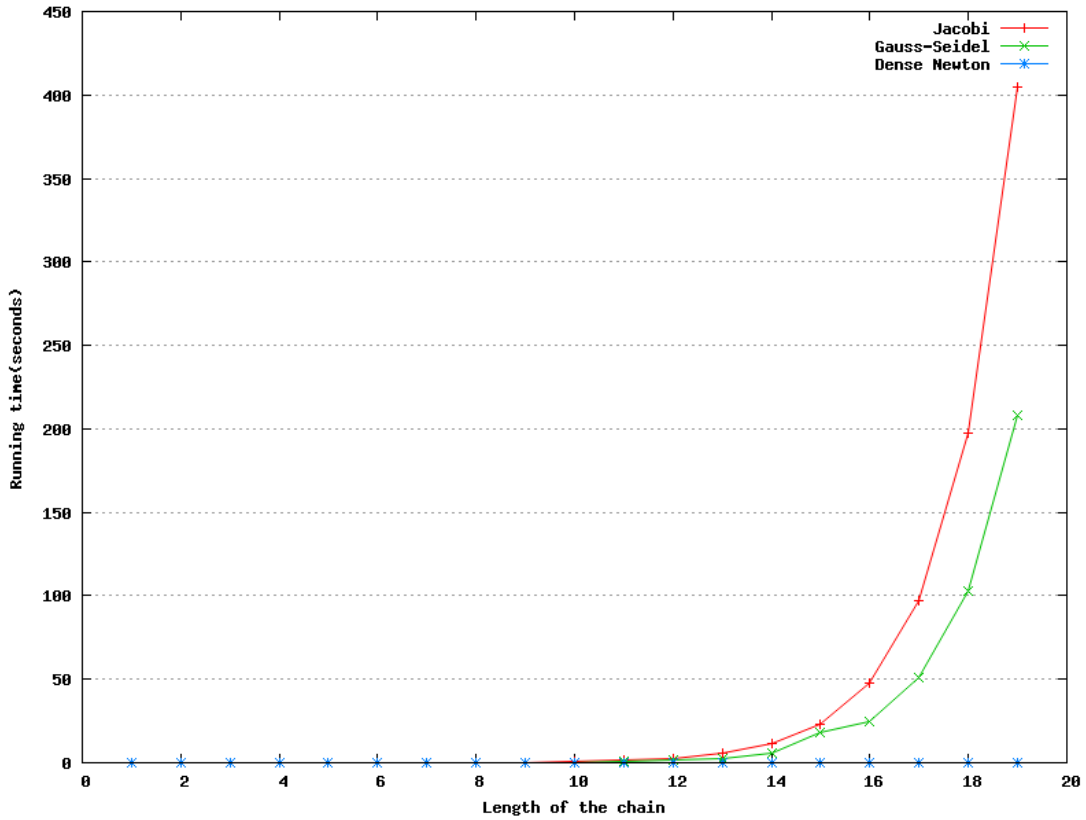

Figure 5.10: Running times for long chains examples, with target absolute tolerance $\varepsilon = 10^{-12}$. Note that the dense Newton's method computes the approximations almost instantly.

bound, which depends on the input, on the biggest finite value that can occur in its LFP solution. If a variable exceeds such a maximal value during the approximation process, its value has to be ∞ in the LFP. However, the just mentioned bound is exponential in the size of the equation system and for the sizes we are considering, this bound would exceed the biggest value that can be stored in double precision, making it useless for our purposes. Furthermore, since such a check would be the same for both tested methods, we had focused instead on examples whose values are all finite.

The Gauss-Seidel method returned the correct LFP solution for every single equation system examined. On the other hand, the strategy improvement in about 10% of the cases did not converge. It was due to a failure in convergence of the underlying sparse linear equation solver – Biconjugate Gradient, and it was despite the fact that all the matrices encountered during that computations were invertible. The average running time of the successful runs only for each algorithm is plotted in Figure 5.11. Notice the big variability of the running times for the Gauss-Seidel method. This is the

result of random and occasional generation of an instance that is very hard to solve for a simple iterative algorithm such as Gauss-Seidel. However, even these "hard" instances are not any harder to solve for the strategy improvement algorithm. The same phenomenon was observed for random 1-RMCs (Section 5.3.2 and [WE07]). Notice also that the average running time of the strategy improvement algorithm looks as if it is linear in the size of the equation system.

In order to evaluate both methods more precisely, we plotted in Figure 5.12 the minimum, maximum and the median running time for the same examples. Also, the average running time is plotted in the same graph for comparison. Notice that the Gauss-Seidel method has a huge variability in its running time among different equations of the same size. For instance, it can solve some examples with 40 000 variables in a time as little as one second and for some others it requires as much as 225 seconds. The most surprising fact is that the median running time for the Gauss-Seidel method turned out to be much lower than for the strategy improvement. For the examined sizes (apart from the size 40 000) the median is about three times lower than the median for Newton's method. On the other hand, stable performance of strategy improvement is very impressive. Its median running time for any size is very close to the minimum running time for the same size and the variability in the running time is really small. That allows us to predict more accurately the time needed to find the solution for a given instance. We should also remember that the average running time of strategy improvement is much lower than for the Gauss-Seidel method, hence although Gauss-Seidel's median running time is lower, it is rather more beneficial to use a strategy improvement algorithm with a linear sparse solver whenever we can.

5.4 Comparison of PReMo with SMCSolver

We performed a number of experiments to compare the performance of our tool PReMo with the state of the art tool for QBDs — SMCSolver [BMSH06] (Structured Markov Chains solver). These two tools in fact are very different. They differ in how the equation systems are represented, the implementation language and implemented numerical algorithms. PReMo is implemented entirely in Java and each equation system is written down in an explicit algebraic formula representation (which allows handling arbitrary monotone systems of nonlinear equations) while SMCSolver makes use of a concise matrix representation for the entire equation system and is implemented in Fortran, a programming language this is geared towards numerical computation. The

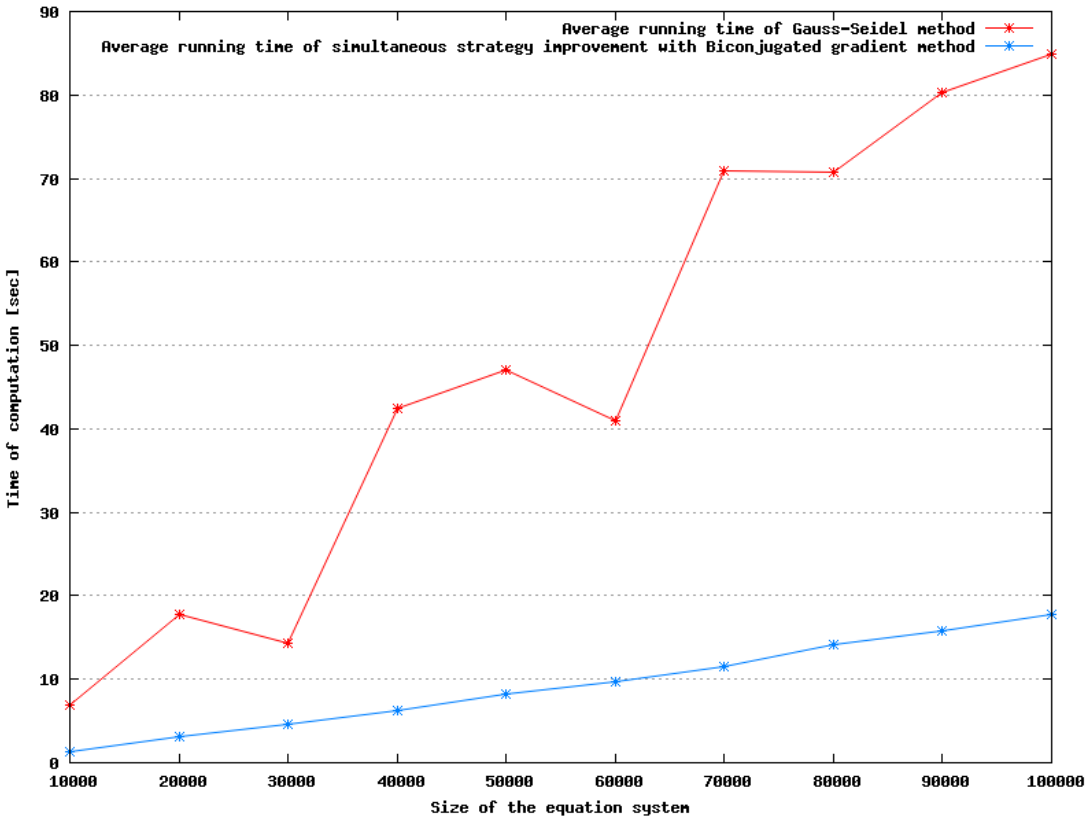

Figure 5.11: The blue line shows the average running time of the successful runs of the simultaneous strategy improvement method that uses the Biconjugate gradient solver as its sparse linear equations solver. The red line is the average running time of the Gauss-Seidel method.

numerical approximation algorithms, apart from the most basic ones, are also different. PReMo's fastest numerical algorithm in practice is a sparse version of Newton's method, which is not yet implemented in SMCSolver (not even in an undecomposed way). Instead, it implements two highly efficient numerical algorithms: *Cyclic Reduction* and *Logarithmic Reduction*, that are designed specifically to solve equation systems arising in the context of QBDs. These algorithms were further speeded up by using a ***shifting technique*** to achieve "quadratic" convergence even in the case of null recurrent QBDs ([Guo07]). For details of these algorithms and a shifting technique see [BMSH06]. Cyclic and Logarithmic Reduction were later modified and applied to Tree-like QBDs (see, e.g. [BLM05]), which are equivalent to RMCs (see Chapter 2) and so they could potentially be used in their analysis. However, the benefit of using them in the context of analyzing probabilistic procedural programs is questionable, since intuitively most programs have a sparse transition matrix which, as we will see later,

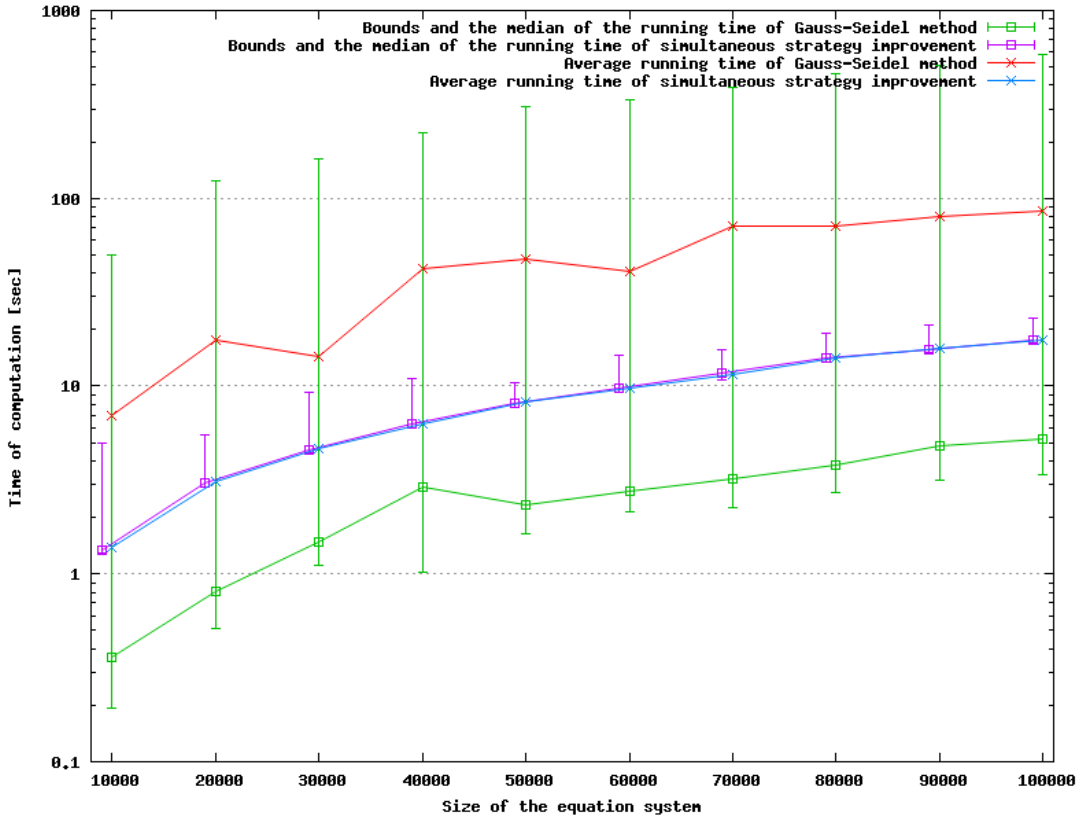

Figure 5.12: The minimum, maximum, median and average running times of each of the tested numerical approximation algorithms are plotted. (The magenta lines were slightly shifted to the left (by 1000), since otherwise the vertical green lines and vertical magenta lines would overlap, as well as the “horizontal” blue line and the “horizontal” magenta line.) Notice that the scale of the X-axis is linear, but the scale of the Y-axis (the running time) is logarithmic. The green bars shows the minimum and maximum running time of the Gauss-Seidel method for each of the sizes with small squares showing its median, and the red line representing its average running time. The magenta bars shows the minimum and maximum running time of the simultaneous strategy improvement that uses the Biconjugate gradient method as its sparse linear equations solver and, as before, small squares showing its median. The blue line represents its average running time (only successful runs were taken into account). Notice that the average running time of Newton’s method combined with a strategy improvement is almost identical to its median time. However, there is a huge difference between the median and the average running time of Gauss-Seidel method. Interestingly enough, the median running time of Gauss-Seidel is a few times lower than the median and average running time of strategy improvement.

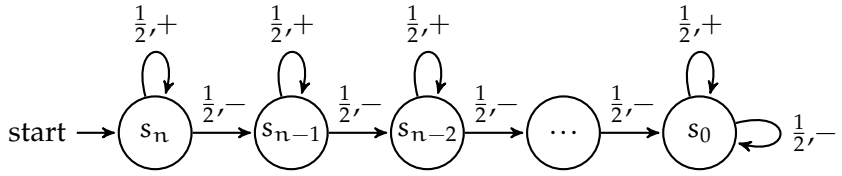

Figure 5.13: An example p1CA (equivalently a QBD) with n control states used in our performance tests.

is better handled by PReMo's sparse representation and sparse version of Newton's method.

In our case study, we measured the impact of the difference in the implementation language and the difference in the equation representation on the performance of PReMo as compared to SMCSolver. Our working example of a p1CA is presented in Figure 5.13. Since we found that for any positive recurrent QBD/p1CA of a reasonable size, its analysis takes less than a fraction of a second, our example is a null recurrent QBD/p1CA. This makes the running times of all the numerical algorithms much longer, hence allowing us to make a more precise comparison. For the comparison to be impartial, we do not include in the running time of PReMo the parsing time of the input equation system. This is because of the fact that when any iterative solution method in SMCSolver is initiated, the whole equation system is stored already in the main memory and does not have to be further preprocessed.

Our first experiment was performed on an instance of the p1CA from Figure 5.13 with 10 control states. We compared the execution time of 150 thousands iterations of the *basic Jacobi* iterative method with no decomposition into SCCs in PReMo with the same number of iterations of the *natural iteration* in SMCSolver. Both of these methods perform exactly the same standard fixed point computation however they do it in a different way. Basic Jacobi method does it step by step for each variable while natural iteration does it for all variables at once via standard matrix operations applied to the whole matrix of variables. Both methods returned solutions not much further apart than the standard floating-point error in double precision. As it turned out, SMCSolver needed only 2.3 seconds to execute that amount of iterations while PReMo needed over 58 seconds. This means that SMCSolver can execute 25 times faster thanks to its matrix equation representation resulting in a far lower cost per iteration and its implementation in a programming language that is geared towards numerical computations. Next, we checked how decomposition into SCCs can improve PReMo's

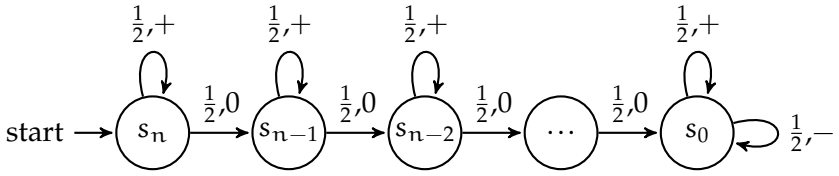

Figure 5.14: A slightly modified version of the example p1CA with n control states from Figure 5.13.

performance. We set an absolute error tolerance to 10^{-10} and ran SMCSolver first. It obtained $2.22 \cdot 10^{-10}$ error after about 95000 steps and took 1.5 second. We then ran PReMo with the same error bound of $2.22 \cdot 10^{-10}$ and it returned a solution within 0.7 seconds. As we can see, decomposition into SCCs increased PReMo's performance over 80 times.

Next, we compared the most efficient solvers of SMCSolver and PReMo, Cyclic and Logarithmic Reduction (with and without the shift acceleration) and the Sparse Newton's method, respectively. The Sparse Newton's method was set to use the Biconjugate Gradients method to solve a sparse linear system of equations that occurs on each step of Newton's method iteration. For the same p1CA with 200 control states (instead of 10), Cyclic Reduction took 14 seconds to find a solution within 10^{-8} tolerance while Logarithmic Reduction was not able to converge at all. When using the shift acceleration, the execution times were reduced to 2.5 and 3 seconds, respectively. On the other hand, our Sparse Newton's method with decomposition of the equation system into SCCs took just 0.814 seconds. (Note that, as we already mentioned before, this value does not include the significant amount of time needed to parse the 30MB big input equation system.)

The previous examples suggest that the decomposition into SCCs gives a huge performance boost, so we also tested PReMo's performance on SMCSolver's built-in examples whose transition matrices are dense and all the variables form just one big SCC. For SMCSolver's Example 1 with 100 control states, it took for Cyclic and Logarithmic Reduction about 0.2 seconds to find a solution and 0.1 second when a shift technique was applied. On the other hand, Gauss-Seidel in PReMo needed 57 seconds and the Sparse Newton's method (using the Biconjugate Gradients method) needed 83 seconds to converge to the same solution with the same tolerance. For SMCSolver's Example 3 with 100 control states the running times for SMCSolver were about the same, 0.2 second, while for Gauss-Seidel in PReMo the running time was about 18 sec-

onds and the Sparse Newton's method did not manage to finish within 3 minutes. To explain this, it has to be noted that in Example 3 from each control state there is a transition of all three possible types to any other control state and this results in a huge equation system. More precisely, if k is the number of control states then the equation representation in a symbolic form grows like $\mathcal{O}(k^3)$ compared to $\mathcal{O}(k^2)$ when represented in a matrix form. As we can see, the good performance of the Sparse Newton's method in the previous example was mainly due to its decomposition into SCCs. Notice, however, that the returned G matrix in this example is a dense lower triangular matrix, so although the transition system is sparse, the returned solution is not.

In order to see how sparsity of the G matrix can further improve PReMo's performance, we changed in our example all the transitions that were decreasing the counter (apart from the last one at s_0) to ones that retain the value of the counter (see Figure 5.14, and compare it with the previous example in Figure 5.13). Any instance of this example is still null recurrent, but now starting at any control state, the system can only terminate at control state s_0 with full probability 1. Now, we were able to generate the input equations for much bigger examples, e.g., with five thousands control states. SMCSolver crashed when run on such a big example and when the number of control states was reduced to 500, its most efficient method (shifted Cyclic Reduction) took 41 seconds to find a solution with a 10^{-8} tolerance. At the same time, PReMo was able to find the solution in 0.624 second even for an example with five thousand control states and could easily handle much bigger examples.

In conclusion, we can see that PReMo can be faster than SMCSolver for examples that are highly decomposable and a lot faster for examples for which a lot of entries of the G matrix are equal to zero. On the other hand, SMCSolver can easily outperform PReMo on dense examples with a high number of transitions thanks to its concise non-symbolic matrix representation of the underlying equation system and by using highly optimized linear matrix algebra packages written for Fortran. This gives rise to a question whether it is possible to combine algorithms operating on the matrix formulation of the equation system with decomposition into SCCs. Newton's method can also be carried out directly over $O(n^2)$ sized matrix equations for QBDs, with low cost per iteration ($O(n^3)$ operations), using known efficient methods for solving the concise linear matrix equations that arise in each iteration of Newton's method over QBDs (certain generalized Sylvester matrix equations, see [BLM05]). However, while TL-QBDs and RMCs also have nonlinear equations with $O(n^2)$ matrix representations, no such efficient solution method is known for the more general linear matrix equa-

tions that arise in iterations of Newton's method on them. Finding such a method would make Newton's method more practical on large "dense" TL-QBDs, RMCs, and pPDSs. But even with such an efficient method, it remains a challenge to combine it well with decomposition, because in general decomposition destroys the matrix form of the equations.

5.5 Conclusions

In this chapter, we presented capabilities of the new tool PReMo that can be used for numerically computing important basic quantities for various classes of (finitely-presentable) countable-state probabilistic models and their controlled extensions. We reported on some of the experiments we conducted with PReMo and the performance of the numerical algorithms implemented. In particular, on random equation systems, all solvers performed well, despite the fact that the best complexity upper bound known for the underlying computational problems is PSPACE ([EY05s]). The decomposed Sparse Newton's method did particulary well for all (not only random) polynomial equation systems, up to quite large sizes, and Linear Sparse Solver (a tailored version of the Sparse Newton's method to linear systems) paired with the simultaneous strategy improvement algorithm performed best on (random) max-linear equation systems. Surprisingly, the median running time (as opposed to the average time) of a simple iterative method, Gauss-Seidel, is much lower than any other method for random max-linear equation systems. Furthermore, PReMo helped to discover some annotation errors in the Penn Treebank corpora, which is well-known in the NLP community, while testing PReMo's performance on a collection of SCFGs derived from it by our colleagues.

We also compared PReMo to SMCSolver, a tool that is tailored to analyzing QBDs. According to our study, PReMo is significantly slower for models with a dense transition system and the main reason being the use of a concise matrix representation by SMCSolver for the underlying equation system. On the other hand, PReMo outperforms SMCSolver on highly decomposable examples, which gives rise to the question whether it is possible to combine algorithms operating on the matrix formulation of the equation system with decomposition into SCCs. Unfortunately, in general decomposition destroys the matrix form of the equation system and it is a major challenge how to tackle this problem.

PReMo is a valuable tool that already attracted attention of the probabilistic veri-

fication community. However, there is still a lot that can be improved and new features that can be added. Of course PReMo could be made faster by reimplementing its core computational procedures like symbolic expression evaluator or using a machine-optimized linear algebra library, e.g., ATLAS. A simple extension would be to add to PReMo a support for even more input models, e.g., pPDSs. However, to make describing more complex models easier, PReMo could be equipped with a more expressive "high-level" language that would allow the user to use integer variables and conditional branching, i.e., a probabilistic version of Boolean Programs [BR00]. Boolean Programs have proved to be highly successful in SLAM ([BR02]), a tool developed by Microsoft Research for verifying Windows's device drivers. However, allowing for integer variables leads to a state explosion problem, since each boolean variable can in principle double the state space, thus doubling in our case the size of the equation system. In order to address this issue we could potentially use Multi-Terminal BDDs ([CFM+93]) that were already successfully applied in verification of finite-state probabilistic systems in the tool PRISM ([KNP02]). Another technique to tackle this problem is *Counterexample Guided Abstraction* (CEGAR) [CGJ+03], used, e.g., in SLAM. However, in probabilistic settings it is even hard to define what a counterexample is and how to represent it. Just recently this topic attracted some attention and results about finding counterexamples in probabilistic finite systems have emerged ([HK07]). Whether any of these techniques could be applied to our setting of infinite-state probabilistic systems is not clear yet.

Furthermore, integrating into PReMo a full-fledged linear-time model checker for RMCs would be very useful for reasoning about such programs. However, this poses a major challenge because there are very difficult numerical issues that have to be overcome in order to enable general model checking for multi-exit RMCs. For instance, to even approximate the probability of visiting a given node infinity often we need to decide whether we terminate for a given multi-exit RMC with the exact probability 1 or strictly less than 1. The answer to the infinitely-many visits question could be 1 in the former and 0 in the later. Such properties cannot be checked in PReMo since all the computations are done with some floating-point error and the only available solution at the moment is to use (extremely impractical) decision procedures for the existential theory of the reals. Of course, such a sensitivity may not occur in practice, and PReMo can always at least provide some lower bounds on the satisfiability probability of a given Linear Time Logic formula.

On the other hand, there are plenty of quantitative properties that could be com-

puted using PReMo about a given model in order to help to reason about it. For instance, using the results from [EKM05], PReMo can quite easily be extended to compute the long-time average reward per transition and the expected total reward for multi-exit RMCs (under the assumption that they terminate) for rewards that depend not only on the control state and the top stack symbol, but also on the size of the stack. Furthermore, the probability of the set of runs for which the size of the stack is bounded can be computed. On the other hand, if for 1-RMDPs we were able to compute the long-time average reward per transition or the variance of the expected total reward then we could easily answer questions about reachability 1-RMDPs, which are not known to be decidable.

From the direct connection between RMCs and TL-QBDs, we can take advantage of new numerical methods for finding the LFP of the underlying set of equations. The two most efficient, Logarithmic and Cyclic reduction, could be implemented in PReMo, but in order to do it efficiently we would need to allow the equation systems to be defined as equations on matrices. Such a representation would also increase the speed and memory usage of the basic iterative methods (at least for dense equation systems). Each step of Newton's method can be represented as a linear matrix equation system for QBDs, as well as for general RMCs. However, for QBDs we can find the solution of such an equation system more efficiently than usual (in $\mathcal{O}(n^3)$ as opposed to $\mathcal{O}(n^6)$ where n is the number of control states and n^2 is thus the size of the input transition matrix) by observing that it is a certain generalized Sylvester matrix equation system ([BLM05]). For RMCs no such efficient solution method is known. Moreover, none of these methods was so far combined with a decomposition into SCCs, and it is questionable whether substantial gain would be achieved by using them in the setting of probabilistic procedural programs, since it is hard to imagine a program whose transition system is dense (i.e., it branches to almost any other instruction at each step).

Appendix Proof of the theorem mentioned in Section 5.2.4

We would like to prove the following:

Proposition 5.5.1. *For an irreducible matrix* $A \geqslant 0$ *and a vector* $b > 0$ *(i.e.,* $b \geqslant 0$ *and* $b \neq 0$*), the least nonnegative solution* x^* *of an equation system* $x = Ax + b$ *is finite iff* $I - A$ *is nonsingular and* $(I-A)^{-1}b \geqslant 0$.

Proof. It can be proved by an easy induction that $x^* \geqslant (\sum_{i=0}^{k} A^i)b$ for any $k \geqslant 0$. If $(\sum_{i=0}^{\infty} A^i)$ diverges then, since A is irreducible and $b > 0$, x^* cannot be finite and in fact all its entries are ∞. On the other hand, if it does converge then it can easily be shown that $x^* = (\sum_{i=0}^{\infty} A^i)b$ satisfies $x = Ax + b$. Furthermore, in such a case we have that in fact $\sum_{i=0}^{\infty} A^i$ is the inverse of $I - A$. This proves ($\Rightarrow$). The other direction is trivial since $(I - A)^{-1}b$, if exists, is a solution of $x = Ax + b$ and since it is $\geqslant 0$ and finite, then the least nonnegative solution has to be finite as well. (In fact, $(I - A)^{-1}b$ is the only solution of $x = Ax + b$ when $I - A$ is nonsingular.) □

Bibliography

[ABKM06] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. In *Proc. of 21st IEEE Computational Complexity Conference*, 2006.

[ABE+05] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.*, 27(4):786–818, 2005.

[BR00] T. Ball and S. K. Rajamani. Bebop: A symbolic model checker for Boolean programs. In *Proc. of 7th SPIN*, 2000.

[BR02] T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In *Proc. of 29th POPL*, 2002.

[BP94] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Classics in Applied Mathematics: SIAM, 1994.

[Bel57] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Bil79] P. Billingsley. *Probability and Measure*. J. Wiley & Sons, 1979.

[BLM03] D. A. Bini, G. Latouche, and B. Meini. Solving nonlinear matrix equations arising in tree-like stochastic processes. *Linear Algebra Appl.*, 366:39–64, 2003.

[BLM05] D. Bini, G. Latouche, and B. Meini. *Numerical methods for Structured Markov Chains*. Oxford Press, 2005.

[BMSH06] D. Bini, B. Meini, S. Steffe, and B. Van Houdt. Structured Markov chains solver: algorithms/software tools. In *Proc. of 1st SMCTools*, 2006.

[BC03] V. Blondel and V. Canterini. Undecidable problems for probabilistic automata of fixed dimension. *Theory of Computing Systems*, 36:231–245, 2003.

[BCSS98] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer, 1998.

[Bra07] T. Brázdil. Verification of probabilistic recursive sequential programs. PhD thesis, Masaryk University, 2007.

[BBE+09] T. Brázdil, V. Brožek, K. Etessami, A. Kučera, and D. Wojtczak. One-Counter Markov Decision Processes. *submitted for publication*, 2009.

[BBFK06] T. Brázdil, V. Brozek, V. Forejt, and A. Kucera. Reachability in recursive Markov decision processes. In *Proc. of 17th CONCUR*, 2006.

[BKS05] T. Brázdil, A. Kučera, and O. Stražovský. Decidability of temporal properties of probabilistic pushdown automata. In *Proc. of 22nd STACS*, 2005.

[Bro65] C. G. Broyden. A Class of Methods for Solving Nonlinear Simultaneous Equations. *Mathematics of Computation*, 19(92):577–593, 1965.

[Can88] J. Canny. Some algebraic and geometric computations in PSPACE. In *Proc. of 20th STOC*, 1988.

[CFM+93] E. Clarke, M. Fujita, P. McGeer, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In *Proc. of 2nd IWLS*, 1993.

[CGJ+03] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of ACM*, 50(5):752–794, 2003.

[CGP99] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[Con92] A. Condon. The complexity of stochastic games. *Inf. & Comp.*, 96(2):203–224, 1992.

[CL89] A. Condon and R. J. Lipton. On the complexity of space bounded interactive proofs. In *Proc. of 29th FOCS*, 1989.

[CM94] A. Condon and M. Melekopoglou. On the complexity of the policy iteration algorithm for stochastic games. *ORSA Journal on Computing*, 6(2), 1994.

[DK06] A. Dubey and F. Keller. *personal communication*, 2006.

[DEKM99] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic models of Proteins and Nucleic Acids*. Cambridge U. Press, 1999.

[Eme90] E. A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, Chapter 16, pages 995–1072, 1990.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proc. of 12th CAV*, 2000.

[EKL08] J. Esparza, S. Kiefer, and M. Luttenberger. Convergence thresholds of Newton's method for monotone polynomial equations. In *Proc. of 25th STACS*, 2008.

[EKM04] J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. In *Proc. of 19th LICS*, 2004.

[EKM05] J. Esparza, A. Kučera, and R. Mayr. Quantitative analysis of probabilistic pushdown automata: expectations and variances. In *Proc. of 20th LICS*, 2005.

[EWY08i] K. Etessami, D. Wojtczak and M. Yannakakis. Recursive Stochastic Games with Positive Rewards. In *Proc. of 35th ICALP(1)*, 2008.

[EWY08q] K. Etessami, D. Wojtczak and M. Yannakakis. Quasi-Birth-Death Processes, Tree-Like QBDs, Probabilistic 1-Counter Automata, and Pushdown Systems. In *Proc. of 5th QEST*, 2008.

[EY05s] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of non-linear equations. In *Proc. of 22nd STACS*, 2005. Full journal version to appear in *Journal of ACM* and is available at: `http://homepages.inf.ed.ac.uk/kousha/rmc_j_version_final.pdf`.

[EY05t] K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic state machines. In *Proc. of 11th TACAS*, 2005.

[EY05i] K. Etessami and M. Yannakakis. Recursive Markov decision processes and recursive stochastic games. In *Proc. of 32nd ICALP*, 2005.

[EY06s] K. Etessami and M. Yannakakis. Efficient qualitative analysis of classes of recursive Markov decision processes and simple stochastic games. In *Proc. of 23rd STACS*, 2006.

[EY06i] K. Etessami and M. Yannakakis. Recursive Concurrent Stochastic Games. In *Proc. of 33nd ICALP*, 2006. Journal version to appear in *Logical Methods in Computer Science*.

[EY07] K. Etessami and M. Yannakakis. On the complexity of Nash equilibria and other fixed points. In *Proc. of 48th FOCS*, 2007.

[EU48] C. Everett and S. Ulam. Multiplicative systems in several variables. Technical report, Part I (LA-683), Part II (LA-690), Part III (LA-707), Los Alamos Scientific Laboratory, 1948.

[FKK$^+$00] R. Fagin, A. Karlin, J. Kleinberg, P. Raghavan, S. Rajagopalan, R. Rubinfeld, M. Sudan, and A. Tomkins. Random walks with "back buttons" (extended abstract). In Proc. of *32nd STOC*, 2000.

[FV97] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.

[GN99] E. Gansner and S. North. An Open Graph Visualization System and Its Applications. *Software - Practice and Experience*, 30:1203–1233, 1999.

[GGJ76] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In *Proc. of 8th STOC*, 1976.

[Gaw08] T. Gawlitza. Personal communication. April, 2008.

[GS07] T. Gawlitza and H. Seidl. Precise relational invariants through strategy iteration. In *Proc. of 16th CSL*, 2007.

[GV88] D. Grigorev and N. Vorobjov. Solving systems of polynomial inequalities in subexponential time. *Journal of Symbolic Computation*, 5(1-2):37–64, 1988.

[Guo07] C.-H. Guo. Comments on a shifted cyclic reduction algorithm for quasi-birth-death problems. In *SIAM J. Matrix Anal. Appl.*, 24:1161–1166, 2003.

[HK07] T. Han and J.-P. Katoen. Counterexamples in probabilistic model checking. In Proc. of 13th TACAS, 2007.

[Har63] T. E. Harris. *The Theory of Branching Processes*. Springer-Verlag, 1963.

[HK66] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Sci.*, 12:359–370, 1966.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Formal Languages and Computation*. Addison-Wesley, 1979.

[IK66] E. Isaacson and H. B. Keller. *Analysis of Numerical Methods*. J. Wiley & Sons, 1966.

[JPY88] D. S. Johnson, C. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

[Jub06] B. Juba. On the hardness of simple stochastic games. Master's thesis, CMU, 2006.

[JE95] X. Jungong and J. Erxiong. Entrywise relative perturbation theory for nonsingular M-matrices and applications. BIT Numerical Mathematics, 35:417-427, 1995.

[KLE07] S. Kiefer, M. Luttenberger, and J. Esparza. On the convergence of Newton's method for monotone systems of polynomial equations. In *Proc. of 39th STOC*, 2007.

[KA02] M. Kimmel and D. E. Axelrod. *Branching processes in biology*. Springer, 2002.

[KS47] A. N. Kolmogorov and B. A. Sevastyanov. The calculation of final probabilities for branching random processes. *Dokaldy*, 56:783–786, 1947.

[Kuc03] A. Kucera. The complexity of bisimilarity checking for one-counter processes. *Theoretical Computer Science*, 304:157–183, 2003.

[KJ02] A. Kucera and P. Jancar. Equivalence-checking with infinite-state systems: Techniques and results. In *Proc. of 29th SOFSEM*, 2002.

[KNP02] M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In *Proc. of 12th Computer Performance Evaluations, Modelling Techniques and Tools*, 2002.

[Lat94] G. Latouche. Newton's iteration for non-linear equations in Markov chains. *IMA J. Numer. Anal.*, 14(4):583–598, 1994.

[LR99] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM series on statistics and applied probability, 1999.

[LR93] G. Latouche and V. Ramaswami. A logarithmic reduction algorithm for quasi-birth-death processes. *J. of Applied Prob.*, 30(3):650–674, 1993.

[MS98] A. Maitra and W. Sudderth. Finitely additive stochastic games with Borel measurable payoffs. *International J. of Game Theory*, 27(2):257–267, 1998.

[MS99] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[Mar98] D. A. Martin. Determinacy of Blackwell games. *J. Symb. Logic*, 63(4):1565–1581, 1998.

[NS06] M. J. Neiderhof and G. Satta. Using Newton's method to compute the partition function of a PCFG, 2006. *unpublished draft manuscript*.

[Neu81] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models:an algorithmic approach*. Dover, 1981.

[Neu89] M. F. Neuts. *Stuctured Stochastic Matrices of M/G/1 Type and their applications*. Marcel Dekker, 1989.

[NS03] A. Neyman and S. Sorin, editors. *Stochastic Games and Applications*. NATO ASI Series, Kluwer, 2003.

[OR70] J. M. Ortega and W.C. Rheinbolt. *Iterative solution of nonlinear equations in several variables*. Academic Press, 1970.

[Ost01] A. Ost. *Performance of Communication Systems. A Model-Based Approach with Matrix-Geometric Methods*. PhD thesis, RWTH Aachen, 2001.

[Pap94] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. In *Journal of Computer and System Sciences*, 48(3):498-532, 1994.

[Pli76] S. Pliska. Optimization of multitype branching processes. *Management Sci.*, 23(2):117–124, 1976/77.

[Put94] M. L. Puterman. *Markov Decision Processes*. Wiley, 1994.

[RHC07] A. Remke, B. R. Haverkort, and L. Cloth. CSL model checking algorithms for QBDs. *Theoretical Computer Science*, 382(1):24–41, 2007.

[Ren92] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals, parts I-III. *J. Symb. Comp.*, 13(3):255–352, 1992.

[RW82] U. Rothblum and P. Whittle. Growth optimality for branching Markov decision chains. *Math. Oper. Res.*, 7(4):582–601, 1982.

[Ser06] O. Serre. Parity Games Played on Transition Graphs of One-Counter Processes. In *Proc. of 9th FOSSACS*, 2006.

[Sha53] L.S. Shapley. Stochastic games. *Nat. Acad. Science*, 39:1095–1100, 1953.

[Sch86] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1986.

[TSY95] T. Takine, B. Sengupta, and R. W. Yeung. A generalization of the matrix M/G/1 paradigm for Markov chains with a tree structure. *Comm. Statist. Stochastic Models*, 11(3):411–421, 1995.

[Tho97] W. Thomas. Languages, automata, and logic. *Handbook of Formal Languages, Vol. 3*, Springer, 1997.

[Tiw92] P. Tiwari. A problem that is easier to solve on the unit-cost algebraic RAM. *Journal of Complexity*, 8(4):393–397, 1992.

[HB03] B. van Houdt and C. Blondia. Tree structured QBD Markov chains and tree-like QBD processes. *Stochastic Models*, 19(4):467–482, 2003.

[Val79] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[VP75] L. G. Valiant and M. Paterson. Deterministic one-counter automata. In *Journal of Computer and System Sciences*, 10:340–350, 1975.

[VHB06] J. Van Velthoven, B. Van Houdt, and C. Blondia. Transient analysis of tree-like processes and its application to random access systems. In *SIGMETRICS/Performance 2006*, 2006.

[Var62] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, 1962.

[Vei69] A. F. Veinott. Discrete dynamic programming with sensitive discount optimality criteria. *Ann. Math. Statist.*, 40:1635–1660, 1969.

[Woj06] D. Wojtczak. PReMo – User's Manual. 2006. `http://groups.inf.ed.ac.uk/premo/documentation.pdf`

[WE07] D. Wojtczak and K. Etessami. PReMo: an analyzer for probabilistic recursive models. In *Proc. of 13th TACAS*, 2007. Tool's web page: `http://groups.inf.ed.ac.uk/premo/`.

[YE05] M. Yannakakis and K. Etessami. Checking LTL properties of Recursive Markov Chains. In *Proc. 2nd QEST*, 2005.

[YA99] R. W. Yeung and A. Alfa. The quasi-birth-death type Markov chain with a tree structure. *Comm. Statist. Stochastic Models*, 15(4):639–659, 1999.

[YS94] R. W. Yeung and B. Sengupta. Matrix product-form solutions for Markov chains with a tree structure. *Adv. in Appl. Probab.*, 26(4):965–987, 1994.