TECHNISCHE UNIVERSITÄT MÜNCHEN
Lehrstuhl für Informatik VII

Verification of Reachability Properties and Termination for Probabilistic Systems

Andreas Gaiser

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. H. Seidl

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Dr. h.c. F.J. Esparza Estaun
2. Prof. A. Kučera, Ph.D.
   Masaryk Univ. / Tschechien

Die Dissertation wurde am 10.10.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 20.02.2013 angenommen.

Abstract

In this thesis we study reachability problems for Markov chains and Markov decision processes originating from probabilistic programs and multi-type finite branching processes. Manual analysis of such systems is challenging and error-prone; its automation has been widely investigated, but mainly for systems with finite state spaces. Apart from the problem of state explosion (and infinite state spaces) one has to deal with the quantitative nature of probabilistic programs.

In the first part of the thesis we consider the problem of computing lower and upper bounds for reachability probabilities of programs. We extend abstraction approaches to arbitrary domains from the abstract interpretation framework, making them more flexible and efficient. In the second part we present a novel algorithm for proving termination with probability one of probabilistic programs. Our algorithm exploits the power of state-of-the-art termination provers for nonprobabilistic programs. In the last part we develop robust numerical algorithms for the analysis of termination properties of multi-type finite branching processes and simple recursive programs.

Zusammenfassung

Diese Arbeit beschäftigt sich mit Erreichbarkeitsproblemen für Markov-Ketten und Markov-Entscheidungsprozesse mit großen und unendlichen Zustandsräumen, die von probabilistischen Programmen und Verzweigungsprozessen stammen. Neben der Zustandsexplosion treten hierbei auch Herausforderungen auf, die von der quantitativen Art der Erreichbarkeitsprobleme herrühren.

Im ersten Teil der Arbeit wird das Problem behandelt, obere und untere Schranken für Erreichbarkeitswahrscheinlichkeiten von Programmen zu errechnen. Bestehende Abstraktionsansätze werden um die Möglichkeit erweitert, beliebige abstrakte Domänen zu benutzen, was die Ansätze flexibler und effizienter macht. Im zweiten Teil wird ein neuer Algorithmus vorgestellt, der fast sichere Terminierung von Programmen beweist. Er benutzt dafür kürzlich entwickelte Terminierungsbeweiser für nichtprobabilistische Programme. Außerdem werden robuste numerische Algorithmen für die Analyse von Terminierungseigenschaften von Verzweigungsprozessen und einfachen rekursiven Programmen entwickelt.

Acknowledgements

Zuallererst möchte ich meinem Betreuer Prof. Javier Esparza danken. Während meiner Zeit als Doktorand lernte ich ihn als sorgfältigen und unendlich geduldigen Betreuer, Ratgeber und Lehrer kennen, der nach Rückschlägen aufbauen konnte, und mit sicherem Gespür meine Forschungsversuche immer wieder in die richtigen Bahnen lenkte. Ich habe viel von ihm gelernt, in vielen Bereichen.

Javier machte mich auch auf das Graduiertenkolleg PUMA (Nr. 1480) der Deutschen Forschungsgemeinschaft aufmerksam, dessen Stipendiat ich sein durfte. Ich danke sowohl der DFG als auch der Technischen Universität München für ihre Unterstützung. Dank gebührt auch meinem Zweitbetreuer Helmut Seidl, der als Leiter des Graduiertenkollegs immer für seine PUMAs eintrat, und Stefan Schwoon, der meinen akademischen Werdegang vom ersten Semester an begleitet hat.

Noch einen Stefan möchte ich ganz besonders hervorheben: Stefan Kiefer. Viele Ergebnisse dieser Arbeit sind in Kooperation mit ihm entstanden. Ich schätze mich glücklich, mit einem solch scharfsinnigen und gütigen Kollegen und Freund zusammenarbeiten zu dürfen, erst in München, dann während meines Aufenthaltes in Oxford.

Am Lehrstuhl I7 herrscht eine ganz besondere Arbeitsatmosphäre. Ich denke an inner- und außeruniversitäre Aktivitäten mit (dem dritten) Stefan und Christian, Laufen und Heimfahrten mit Michael, und an meinen Zimmerkollegen Jan, unsere Gespräche über Deutsche, Tschechen, Sprache, Religion und den ganzen Rest; sie alle sind mir Freunde geworden. Danke auch an alle neuen und alten Kollegen: Jörg, Remy, René, Maximilian, Juan, Frau Leber, Frau Auer etc., und besonders an Andreas Reuß: es war immer schön, mit ihm Kriegsrat zu halten.

I am grateful for the opportunity to work with the colleagues of our new research group: Andrey, Corneliu, Ruslan, and Ashuthosh - I want to mention in particular the disussions with Ruslan, and Corneliu's and Andrey's help with taming ARMC and lots of other things.

Von Björn Wachter habe ich viel gelernt über probabilistisches Model Checking und Abstraktion, vielen Dank dafür.

I also thank Prof. Antonín Kučera for agreeing to review this thesis.

Meiner Schwester Diana und Simone möchte ich danken für ihren Beistand in den vergangenen Jahren. Meine Eltern haben mich in unaufhörlicher Liebe in jeder Situation meines Lebens unterstützt und begleitet. Meine Dankbarkeit ihnen gegenüber lässt sich schwerlich in Worte fassen. Ich bin unendlich froh, sie haben zu dürfen.

S · D · G

Contents

List of Figures

Chapter 1

Introduction

Everything existing in the universe is the fruit of chance.
$\Delta\eta\mu\acute{o}\kappa\rho\iota\tau o\varsigma$

Probability theory finds applications in all areas of modern science. Probabilities are used to model the development of animal populations in systems biology (Haccou et al. [51]), for RNA and DNA structure prediction in molecular biology (Anderson et al. [1]), to represent cascade reactions in nuclear physics (Harris [54]), and to investigate reactive systems and randomized algorithms in computer science, to name just a few examples.
Important mathematical models for describing such systems are discrete *Markov models*, in particular *Markov chains* and *discrete Markov decision processes.* They represent simple but nevertheless powerful formalisms and are used in all example applications that we mentioned. Markov chains and Markov decision processes have finite or countable state spaces. States change in discrete time steps, and successor states are determined probabilistically. Additionally, Markov decision processes allow to model nondeterminism: in some states a successor state is picked by a player, who can have different objectives, and might e.g. represent an “unpredictable” environment.
In the investigation of systems with probabilistic behaviour it is often important

to deduce with which probability certain events occur, like the maximal probability of reaching an error location in a computer program, the probability that such a program terminates, or the probability of a nuclear explosion in a container filled with radioactive material. These questions can be transferred to the problem of reaching a certain set of states in the Markov model representing the system. More precisely: in Markov chains we like to compute the probability of reaching a given set of states; in Markov decision processes (where such a probability depends on the behaviour of the player) we are interested in both the minimum and maximum probability of reaching a given set of states, taken over all behaviours of the player. We refer to these questions as *reachability problems* in the following.

In this work we mainly study reachability problems for Markov models that originate from *probabilistic programs.* They are equipped with a random number generator having a finite number of outcomes. Additionally they can exhibit nondeterministic behaviour. Manual analysis of probabilistic programs is a very challenging and error-prone task; its automation has therefore been widely investigated in the verification community, mainly for systems with finite state spaces. In addition to the problem of *state explosion*, which is well-known from the analysis of nonprobabilistic programs, one has to deal with the *quantitative nature* of reachability problems for Markov models, since it is often required to compute a numerical value (a probability) instead of just giving a 0/1-answer as in nonprobabilistic verification (like: is the state reachable or not). However, *qualitative* reachability problems, like proving that a program reaches a state with probability 1, pose a lot of new challenges as well. In particular they often become undecidable if one considers infinite state spaces. In this case one has to rely on incomplete approaches.

Verification of probabilistic programs. Quite early, probabilistic programs became a popular research topic in the verification community, caused by their growing importance in areas like reactive systems and concurrent programs; see e.g. Baier and Katoen [7] for a detailed introduction, and Baier et al. [10] for a survey of applying analysis techniques for probabilistic systems to performance evaluation. After the semantical foundations were laid in the 1980s', e.g. by the work of Kozen (Kozen [65]), approaches evolved for model checking of mostly finite-state systems against specifications in temporal logics like LTL or PCTL, for qualitative as well as quantitative properties (see e.g. Bianco and Alfaro [14], Courcoubetis and Yannakakis [24], Sharir et al. [94], Vardi [97] and Lehmann and Shelah [67]). In particular, the problem of proving termination of programs with probability 1 received special attention, e.g. in Arons et al. [3]. We note that apart from modeling systems with Markov decision processes and Markov chains, which have a discrete-time semantics, also *continuous-time Markov chains* and *continuous-time Markov decision processes* have become widely used for determining system performance (see e.g. Baier et al. [9]). However we do not study them in this thesis. Quite often techniques from nonprobabilistic verification could be applied successfully to the probabilistic setting. For example, one of the approaches for fighting the state explosion in probabilistic programs consists of representing Markov decision processes and Markov chains with the help of symbolic decision diagrams as in Parker [82], similar to the use of BDDs in "classical" verification (Esparza and Schwoon [34], Hun et al. [58]). These techniques were used in one of the first and most widely spread tools for the verification of probabilistic systems, the model checker PRISM (Kwiatkowska et al. [66]). It supports finite-state probabilistic programs with nondeterminism and concurrency and provides a wide range of properties to be checked. The use of abstraction techniques has been identified quite early as a good solution for coping with large and especially infinite state spaces. Besides techniques from the abstract interpretation framework (Cousot and Cousot [25]) in work like Monniaux [76] and Di Pierro et al. [31], the counterexample-guided abstraction refinement approach (CEGAR, see Clarke et al. [19]) has been adapted to the probabilistic setting, e.g. in Han et al. [53] and Hermanns et al. [55] and the approaches of Wachter and Zhang [99] and Kattenbelt et al. [62]. In particular the latter two are of particular importance in this thesis. For probabilistic programs with recursion or dynamic process creation, analysis tools like PReMo (Etessami and Yannakakis [42], Wojtczak and Etessami [102]) have been developed, which can be used to compute e.g. termination probabilities using sophisticated numerical methods.

Contributions of the thesis. In this thesis we develop algorithms for solving reachability problems for classes of probabilistic programs with potentially infinite state space in an automatic or semi-automatic way. We show that it is useful to employ *different techniques*, specifically tailored to the requirements of the considered systems and problems:

- Instead of constructing the concrete semantics of a program, we use *abstraction* (Cousot and Cousot [25]) to obtain small abstract systems from which we can still obtain useful information about reachability probabilities of the program.
- We use tools from nonprobabilistic verification to reduce special cases of reachability problems to purely nonprobabilistic problems.
- We develop numerical techniques to analyze special classes of probabilistic systems. For example, we avoid numerical imprecision by combining inexact and exact arithmetic and obtain efficient algorithms.

We focus on two important and closely related verification problems: Computing bounds for the probability of reaching a fixed program point and proving that a final state of a program (or other system) is reached with probability one; the latter property is called *almost-sure termination*.

Outline. In **Chapter 2** we introduce basic notational conventions and mathematical results that are used throughout the thesis. **Chapter 3** recalls basics from probability theory and then defines Markov chains and Markov decision processes, together with some important properties of these stochastic models.
Chapter 4 describes our approach for computing bounds for reachability probabilities of probabilistic programs with possibly infinite state space, using arbitrary domains from the abstract interpretation framework (Cousot and Cousot [25]). Our approach is an extension of the game-based abstraction approaches from Kattenbelt et al. [62] and Wachter and Zhang [99]. These approaches rely on predicate abstraction. We introduce concepts from abstract interpretation and game theory for defining so-called *abstract game arenas*, and explain how to obtain lower and upper bounds for reachability probabilities from them. We

give an algorithm for constructing abstract game arenas using arbitrary domains, and discuss possible refinement techniques. With the help of examples we show differences and advantages of our extension compared to the approaches based solely on predicate abstraction.

The techniques from Chapter 4 are often not efficient when we want to prove that an (infinite-state) program is almost-surely terminating (i.e., is reaching a terminating state with probability one). Problems arise which are similar to situations in nonprobabilistic verification for proving termination: the approaches tend to unroll loops, which results in large game arenas. We therefore present a new method for proving almost-sure termination of an important class of infinite state programs, called weakly finite programs, in **Chapter 5**. Let P be a program. Our approach is based on *patterns*: a pattern Φ is a simple expression describing a set S_Φ of runs of P with S_Φ having probability one. Φ is called terminating if every run in S_Φ eventually reaches a terminating state. The existence of a terminating pattern therefore proves almost-sure termination of P. We give a method for computing terminating patterns that exploits the power of state-of-the-art model checkers and termination provers for nonprobabilistic programs. In the case of finite state programs it finds terminating patterns automatically, for general weakly finite programs it requires an extrapolation step. We give a completeness result of our proof method for the class of weakly finite programs. We also show how to use termination information to improve algorithms for quantitative computation of reachability probabilities.

For the programs we study in Chapter 5, almost-sure termination can be proved without taking the actual probabilities occurring in the program into account. In **Chapter 6** we consider a class of infinite-state Markov chains that originate from multi-type finite branching processes. These processes are important in many branches of science and in particular in the termination analysis of simple recursive programs. For these systems, the actual probabilities play an important role in their termination properties. It turns out that almost-sure termination of such systems can be characterized by a sequence of probabilities called extinction probabilities. We develop a strongly-polynomial algorithm for deciding whether the extinction probabilities are all equal to one, which gives us an algorithm for deciding almost-sure termination of special classes of programs based on solving

linear equation systems in the size of the branching process. We show that our algorithm is more efficient in practice than existing ones based on exact Linear Programming. We also study the problem of how to obtain reliable bounds for extinction probabilities. We show that existing approaches might exhibit numerical problems, and until now only methods for computing *lower* bounds were known. We give a robust algorithm that computes arbitrarily precise lower and upper bounds for extinction probabilities. Although we use inexact arithmetic for efficiency in the algorithm, we do so in a controlled way and always obtain reliable bounds.
We end in **Chapter 7** with a short summary and an outlook: we propose several directions for future research, which on particular aim at combining the proposed techniques.

Chapter 2

Preliminaries

In this section we establish some basic notational conventions and recall fundamental structures and results that are needed throughout the thesis.
We denote by $\mathbb{N}$ the set of natural numbers $1, 2, \ldots$, by $\mathbb{Z}$ the integers, by $\mathbb{Q}$ the rational numbers and by $\mathbb{R}$ the real numbers. We denote the power set of a set A by 2^A. We write infinite sequences $a^{(1)}, a^{(2)}, \ldots$ respectively $a_1, a_2, \ldots$ as $(a^{(i)})_{i\in\mathbb{N}}$ respectively $(a_i)_{i\in\mathbb{N}}$.

2.1 Lattices and Fixed Points

In different parts of the thesis we investigate how to compute or approximate fixed points of functions defined over *partially ordered sets*:

Definition 1. (Partially ordered set)
Let A be a set. A pair $(A, \sqsubseteq)$ with $\sqsubseteq \subseteq A \times A$ is a partially ordered set if $\sqsubseteq$ is anti-symmetric (i.e. for all elements a and b in A with $a \sqsubseteq b$ and $b \sqsubseteq a$ it follows that $a = b$), reflexive and transitive. $\square$

If $\sqsubseteq$ is clear we just write A for $(A, \sqsubseteq)$. We often study partially ordered sets with additional properties, so-called *complete lattices*:

Definition 2. (Complete lattice)
Let A be a set. A partially ordered set $(A, \sqsubseteq)$ is a *complete lattice* if for every subset $S \subseteq A$

- there exists an element $a \in A$, written $a = \bigsqcap S$ and called the *infimum* of S, such that (i) for every $x \in S$ it holds that $a \sqsubseteq x$ and (ii) for every $b \in A$ such that $b \sqsubseteq x$ for every $x \in S$, $b \sqsubseteq a$ holds.

- there exists an element $a \in A$, written $a = \bigsqcup S$ and called the *supremum* of S, such that (i) for every $x \in S$ it holds that $x \sqsubseteq a$ and (ii) for every $b \in A$ such that $x \sqsubseteq b$ for every $x \in S$, $a \sqsubseteq b$ holds.

$\bigsqcap \emptyset$ is denoted by $\top$ (the maximal or top element), and $\bigsqcup \emptyset$ by $\bot$ (the least or bottom element).
We often write a complete lattice as tuple $(A, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$. □

If $(A, \sqsubseteq)$ is a complete lattice, suprema and infima are unique, as are the bottom and top element. An easy example of a complete lattice is $(2^A, \subseteq, \bigcup, \bigcap, A, \emptyset)$ for an arbitrary set A: the supremum of two lattice elements a and b is their union, the infimum their intersection; the bottom element is $\emptyset$, the top element A.
Another example is the lattice $([0,1], \leq, \sup, \inf, 1, 0)$: for every subset $S \subseteq [0,1]$ both supremum and infimum are again contained in $[0,1]$.
An important class of functions defined on complete lattices are *monotone* and *continuous functions*[1]:

Definition 3. (Monotone and continuous functions, fixed points)
Let $L = (A, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$ be a complete lattice, and $f : A \to A$ a function. f is

- *monotone* with respect to L if for all elements a and b in A with $a \sqsubseteq b$ it holds that $f(a) \sqsubseteq f(b)$,

- f is *continuous* with respect to L if it is monotone and for all $S \subseteq A$ it holds that $f(\bigsqcup S) = \bigsqcup f(S)$.[2]

An element $x \in A$ is a *fixed point* of f if $f(x) = x$. □

A fundamental result from lattice theory for monotone functions is the Knaster-Tarski theorem (see e.g. Winskel [101]): it gives a characterization of the fixed points of a monotone function f defined over a complete lattice.

[1]We do not introduce CPOs here, therefore we give a somewhat stronger definition of continuous functions etc., that is sufficient for our needs.

[2]We use the standard notation $f(S) := \{f(s) \mid s \in S\}$.

Theorem 1. *(Knaster-Tarski theorem)*
Let $(A, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$ *be a complete lattice, and* $f : A \to A$ *a monotone function. Let*

$$\mu_f = \bigsqcap\{x \in A \mid f(x) \sqsubseteq x\}.$$

Then μ_f *is the* least fixed point *of* f*, i.e. for every fixed point* x *of* f *it holds that* $x \sqsubseteq \mu_f$*. Now let*

$$\zeta_f = \bigsqcup\{x \in A \mid x \sqsubseteq f(x)\}.$$

Then ζ_f *is the* greatest *fixed point of* f*, i.e. for every fixed point* x *of* f *it holds that* $x \sqsubseteq \zeta_f$*.* □

A famous theorem due to Kleene characterizes least fixed points of continuous functions (see e.g. Winskel [101]):

Theorem 2. *(Kleene's fixed point theorem for complete lattices)*
Let $(A, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$ *be a complete lattice, and* $f : A \to A$ *a continuous function. Then the least fixed point* μ_f *of* f *is given by*

$$\mu_f = \bigsqcup\{f^i(\bot) \mid i \in \mathbb{N}\} = \lim_{k\to\infty} f^k(\bot),$$

i.e. μ_f *is the limit of the sequence* $\bot, f(\bot), f(f(\bot)), \ldots$*.* □

Consider for example a function $f : [0,1] \to [0,1]$ with

$$f(x) = \sum_{i=0}^{m} a_i \cdot x^i,$$

with $a_i \in [0,1]$ for $0 \le i \le m$, and $\sum_{i=0}^{m} a_i \le 1$. f is continuous with respect to $([0,1], \le, \sup, \inf, 1, 0)$ (here both in the lattice-theoretic as in the calculus sense). By Kleene's Theorem every such f has a least fixed point, given by $\lim_{n\to\infty} f^n(0)$.

2.2 Languages

Let Σ be a finite non-empty set. We denote by Σ^* and Σ^ω the sets of finite and infinite words over Σ, and set $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. For $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$, we write L_1L_2 for the set $\{uv \mid u \in L_1 \land v \in L_2\}$.

Sometimes we require the elements of an alphabet to be ordered:

Definition 4. (Ordered alphabet)
Let $n \in \mathbb{N}$. Let Σ be a finite nonempty set with $|\Sigma| \geq n$. An ordered alphabet A is a finite sequence $(a_1, \ldots, a_n) \in \Sigma^n$ with $a_i \neq a_j$ for $1 \leq i < j \leq n$. We define $a_i < a_j$ iff $1 \leq i < j \leq n$. $\square$

We often write A as $\{a_1, \ldots, a_n\}$ (abusing the notation slightly to encode the order of the elements) and treat it as a set.

2.3 Vectors

We use bold letters for designating (column) vectors, e.g. $\mathbf{v} \in \mathbb{R}^n$. We write $\overline{s}$ with $s \in \mathbb{R}$ for the vector $(s, \ldots, s)^\top \in \mathbb{R}^n$ (where $^\top$ indicates transpose), if the dimension n is clear from the context. The i-th component of $\mathbf{v} \in \mathbb{R}^n$ will be denoted by $\mathbf{v}_i$. We write $\mathbf{x} = \mathbf{y}$ (resp. $\mathbf{x} \leq \mathbf{y}$ resp. $\mathbf{x} \prec \mathbf{y}$) if $\mathbf{x}_i = \mathbf{y}_i$ (resp. $\mathbf{x}_i \leq \mathbf{y}_i$ resp. $\mathbf{x}_i < \mathbf{y}_i$) holds for all $i \in \{1, \ldots, n\}$. By $\mathbf{x} < \mathbf{y}$ we mean $\mathbf{x} \leq \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$. The (n-dimensional) i-th unit vector is denoted by $\mathbf{e}^i \in \mathbb{R}^n$, i.e. $\mathbf{e}^i_j = 0$ for $i \neq j$ and $\mathbf{e}^i_j = 1$.
We note that the partial order $([0,1]^n, \leq)$ forms a complete lattice, (as an n-times product of $([0,1], \leq)$), with $\overline{0}$ as the bottom element and $\overline{1}$ the top element.

Chapter 3

Probability Theory and Markov Models

To give a precise semantics for the probabilistic systems studied in this thesis we use *Markov chains* and *Markov decision processes*, which are simple but fundamental models for representing probabilistic systems. We introduce them as labeled transition systems.

Markov chains are *purely probabilistic*: in every state, a successor state is determined by a probabilistic choice like e.g. a coin toss, and every transition is weighted by a probability. However, each transition also has an additional label that can carry further information.

Markov decision processes, an extension of Markov chains, additionally allow *nondeterministic decisions* by introducing a *player* who is allowed to choose a successor state in special situations. His choices are represented by *action transitions*. There are in general many different possible behaviours for the player, which influence the properties of the resulting process.

We also introduce *reachability problems* for both models, the central questions tackled throughout this thesis. We require some basic facts from probability theory. More extensive treatments can be found in standard probability theory literature. Our notation is similar to the one in Rosenthal [91] and Baier and Katoen [7].

3.1 Probability Spaces

Let us consider a random experiment where exactly one event out of a set S occurs. We then call S the set of *elementary events* of the experiment. σ-algebras represent a collection of experiment outcomes, i.e. subsets of S:

Definition 5. (σ-algebra)
Let S be a set. A σ-algebra $\mathfrak{S}$ is a subset of 2^S over S such that:

- $\{\emptyset, S\} \subseteq \mathfrak{S}$,
- for every sequence $(S_i)_{i\in\mathbb{N}}$ in $\mathfrak{S}$, $\bigcup_{i\in\mathbb{N}} S_i \in \mathfrak{S}$ and $\bigcap_{i\in\mathbb{N}} S_i \in \mathfrak{S}$, and
- if $T \in \mathfrak{S}$, then also $S \setminus T \in \mathfrak{S}$.

$\square$

A *probability measure* is used to assign probabilities to elements of a σ-algebra (in general it is not possible to assign a probability to *arbitrary* subsets of S):

Definition 6. (Probability measure and probability space)
Given a σ-algebra $\mathfrak{S} \subseteq 2^S$ over S, a function $\Pr[\cdot] : \mathfrak{S} \to [0,1]$ is a *probability measure* over $\mathfrak{S}$ if

- $\Pr[\emptyset] = 0, \Pr[S] = 1$, and
- $\Pr$ is countably additive, i.e. for every sequence $(S_i)_{i\in\mathbb{N}}$ in $\mathfrak{S}$ such that $S_i \cap S_j = \emptyset$ for all $i \neq j$,

$$\Pr[\bigcup_{i\in\mathbb{N}} S_i] = \sum_{i\in\mathbb{N}} \Pr[S_i].$$

The tuple $(S, \mathfrak{S}, \Pr)$ is called a *probability space.* $\square$

Often we will deal with simple probability spaces that describe the probability that one out of countably many events occurs:

Definition 7. (Discrete probability distribution)
Let S be a finite or countable set. We call a function $P : S \to [0,1]$ a *discrete*

probability distribution, or discrete distribution for short, if $\sum_{s \in S} P(s) = 1$. Further we call P having *finite support* if $P(s) = 0$ for all but finitely many $s \in S$. We denote by $\mathrm{Dist}(S)$ the set of all discrete distributions, and by $\mathrm{Dist}_F(S)$ the set of all discrete distributions with finite support. $\square$

A discrete probability distribution P over a set S can be used to form a probability measure $\hat{P} : 2^S \to 2^S$ for the σ-algebra 2^S, by setting for every $M \subseteq S$

$$\hat{P}(M) := \sum_{s \in M} P(s).$$

The tuple $(S, \mathfrak{S}, \hat{P})$ then forms a probability space.
We mention the usual convention of conditional probability: we define for every S, T elements in $\mathfrak{S}$ with $\Pr[T] \neq 0$ the probability $\Pr[S \mid T]$ by $\frac{\Pr[S \cap T]}{\Pr[T]}$ ($S \cap T \in \mathfrak{S}$ due to the definition of σ-algebras). We call a set $S \in \mathfrak{S}$ a null set if $\Pr[S] = 0$. Note that then $\Pr[T] = \Pr[S \cup T]$ for every $T \in \mathfrak{S}$.

3.2 Markov Chains and Markov Decision Processes

Both Markov chains as well as Markov decision processes are transition systems with finite or countable state space.[1] In a Markov chain, each transition is labeled and weighted with a probability.

Definition 8. (Markov chain)
A *Markov Chain* $\mathcal{M}$ is a tuple $(Q, \to, \mathrm{Lab})$, where

- Q is a finite or countable set of states,
- Lab is a finite or countable set of labels,
- $\to \subseteq Q \times (0,1] \times \mathrm{Lab} \times Q$ is a transition relation.

The relation $\to$ has to satisfy the following conditions:

1. if (q, p, ℓ, q') and (q, p', ℓ, q') are in $\to$, then $p = p'$;

[1]We do not study uncountable state spaces in this work.

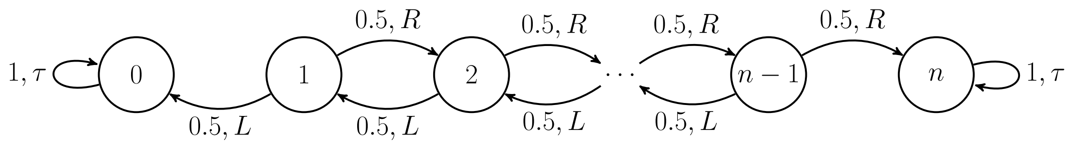


Figure 3.1: Example of a finite-state Markov chain: the bounded random walk. The labels R resp. L mean going one step to the right resp. to the left.

2. the probabilities of the outgoing transitions of every $q \in Q$ add up to 1, i.e. for every fixed $q \in Q$, $\sum_{(q,p,\ell,q') \in \rightarrow} p = 1$.

□

For $(q, p, \ell, q') \in \rightarrow$ we write $q \xrightarrow{p,\ell} q'$. The outgoing transitions of a state q of a Markov chain can be used to define a transition distribution for q:

Definition 9. (Transition distributions)
Let $\mathcal{M} = (Q, \rightarrow, \text{Lab})$ be a Markov chain. For every $q \in Q$ we define the distribution $\delta_q : Q \rightarrow [0, 1]$ by setting $\delta_q(q') := \sum_{(q,p,\ell,q') \in \rightarrow} p$.

A Markov chain can be visualized as a graph with labeled edges. A run in a Markov chain corresponds to a play on its graph: a token is placed on a state $q \in Q$. Then an outgoing transition of q is chosen, according to the probability weights, i.e. a transition (q, p, ℓ, q') is chosen with probability q. The token is moved to q' and the play continues in the same manner.
The Markov chain $\mathcal{M} = (\{0, 1, \ldots, n\}, \rightarrow, \{L, R, \tau\})$ drawn in Fig. 3.1 represents a standard example from Markov chain theory, the *finite random walk*, often also called *Drunkard's Walk*: a heavily drunk pub visitor starts at a position $pos > 0$ (the pub) and staggers randomly either one step to the left or one to the right. If she reaches 0 or n (0 might be her home, n a police station), she stays there.
Fig. 3.2 represents an *infinite random walk*, which essentially removes the "sink state" on the right side of the walk.
For transitions that do not require a label we use the *empty label* τ (and we often omit it, especially in figures).
Markov decision processes, MDPs for short, are an extension of Markov chains. A state of an MDP can be either an action state or a probabilistic state, the latter having essentially the same behaviour as a state in a Markov chain.

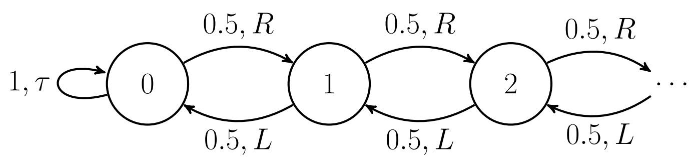


Figure 3.2: The right-unbounded random walk, an infinite-state Markov chain.

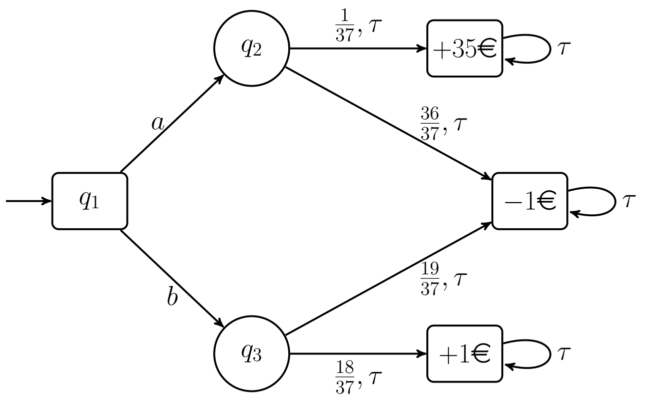


Figure 3.3: Example MDP. Probabilistic states are drawn as circle-shaped nodes, action states as rectangle-shaped nodes.

Definition 10. (Markov Decision Process)
A *Markov Decision Process* (MDP) is a tuple $\mathcal{M} = (Q_A, Q_P, \mathrm{Init}, \rightarrow, \mathrm{Lab}_A, \mathrm{Lab}_P)$, where

- Q_A and Q_P are countable or finite sets of *action states* and *probabilistic states*; we set $Q := Q_A \cup Q_P$,
- $\mathrm{Init} \subseteq Q_A \cup Q_P$ is a set of *initial states*, and
- Lab_A respectively Lab_P is a countable set of *action labels* respectively *probabilistic labels*, and
- the relation $\rightarrow$ is equal to $\rightarrow_A \cup \rightarrow_P$, where $\rightarrow_A \subseteq Q_A \times \mathrm{Lab}_A \times Q$ is a set of *action transitions*, and $\rightarrow_P \subseteq Q_P \times (0,1] \times \mathrm{Lab}_P \times Q$ is a set of *probabilistic transitions.*

It further satisfies the following conditions:

- if (q, p, ℓ, q') and (q, p', ℓ, q') are probabilistic transitions, then $p = p'$,

- if (q, ℓ, q_1) and (q, ℓ, q_2) are action transitions, then $q_1 = q_2$,
- the probabilities of the outgoing transitions of a probabilistic state add up to 1, i.e. for every $q \in Q_P$, $\sum_{(q,p,\ell,q') \in \rightarrow} p = 1$,
- every state of Q_A has at least one successor in $\rightarrow_A$.

□

We write $q \xrightarrow{\ell} q'$ for $(q, \ell, q') \in \rightarrow_A$, and again $q \xrightarrow{p,\ell} q'$ for $(q, p, \ell, q') \in \rightarrow_P$ (we omit p if it is irrelevant).
If for a Markov Decision Process $\mathcal{M} = (Q_A, Q_P, \text{Init}, \rightarrow, \text{Lab}_A, \text{Lab}_P)$ it holds that $Q_A = \emptyset$, then we will interpret $\mathcal{M}$ as the Markov chain $(Q_P, \rightarrow, \text{Lab}_P)$. Therefore we can treat Markov chains as a special case of MDPs from now on. In particular, each definition in the following given in terms of MDPs can be used for Markov chains as well. Note however that we include initial states only in the definition of MDPs, not in the definition of Markov chains.
Similar to Markov chains we can interpret a run in an MDP as a play where a token is moved on its graph: if the token is located in a probabilistic state, a probabilistic choice takes place to obtain a successor state as in Markov chains; however, in an action state, a player chooses the successor state. Consider the example MDP in Fig. 3.3. As a convention we draw action states as rectangle-shaped nodes and probabilistic states as ellipse or circle-shaped nodes. It models a game where the player starts in state q_1 and has to decide whether he chooses label a or label b. The game ends up in one of the three states on the right, symbolizing the player winning 35€, 2€, or losing 1€. The label a represents a more adventurous choice of the player (apparently).

3.2.1 Runs and Paths

Let us fix an MDP $\mathcal{M} = (Q_A, Q_P, \text{Init}, \rightarrow, \text{Lab}_A, \text{Lab}_P)$ for this section (by interpreting Markov chains as MDPs, we can use the following notations for both Markov chains and MDPs). We set $Q = Q_A \cup Q_P$ and $\text{Lab} = \text{Lab}_A \cup \text{Lab}_P$. A *run* of an MDP $\mathcal{M}$ is an infinite word $r = q_0\ell_0q_1\ell_1 \ldots \in (Q\text{Lab})^\omega$ such that for all $i \geq 0$ either $q_i \xrightarrow{p,\ell_i} q_{i+1}$ for some $p \in (0, 1]$ or $q_i \xrightarrow{\ell_i} q_{i+1}$. We call r *initial*

if $q_0 \in \mathrm{Init}$. We denote the set of runs starting at a state q by $\mathrm{Runs}^{\mathcal{M}}(q)$. For MDPs we additionally denote the set of all initial runs of $\mathcal{M}$ by $\mathrm{Runs}^{\mathcal{M}}$.
A *path* π is a proper prefix of a run r, and we call π initial if r is initial. We denote by $\mathrm{Paths}^{\mathcal{M}}(q)$ the set of all paths starting at q, and by $\mathrm{Paths}^{\mathcal{M}}$ all initial paths of $\mathcal{M}$ (again if $\mathcal{M}$ is an MDP). We denote the *last* state occuring in a path π by $\mathrm{last}(\pi)$.
We often write $r = q_0 \xrightarrow{\ell_0} q_1 \xrightarrow{\ell_1} q_2 \xrightarrow{\ell_2} \ldots$ instead of $r = q_0\ell_0q_1\ell_1q_2\ldots$ for runs (similarly for paths), and omit the superscripts of Runs and Paths if the context is clear. If necessary we also add the probabilities of transitions, i.e. we write $r = q_0 \xrightarrow{p_0,\ell_0} q_1 \xrightarrow{p_1,\ell_1} q_2 \xrightarrow{p_2,\ell_2} \ldots$ if $(q_0, p_0, \ell_0, q_1) \in \rightarrow, (q_1, p_1, \ell_1, q_2) \in \rightarrow, \ldots$.
For $w \in \mathrm{Lab}^*$ with $w = \ell_1 \ldots \ell_k$ we write $q \xrightarrow{w} q'$ if there exists $q_1, \ldots, q_{k-1}$ such that

$$q \xrightarrow{\ell_1} q_1 \xrightarrow{\ell_2} q_2 \xrightarrow{\ell_3} \ldots \xrightarrow{\ell_{k-1}} q_{k-1} \xrightarrow{\ell_k} q' \in \mathrm{Paths}^{\mathcal{M}}.$$

We can extend this notation to regular expressions over Lab^*: e.g. $q \xrightarrow{\ell_1^*} q'$ holds iff there exists a path from q to q' where only the label ℓ_1 is visited.
For each path $\pi \in \mathrm{Paths}^{\mathcal{M}}(q)$ with $q \in Q$, we define

$$\mathrm{Cyl}(\pi, \mathcal{M}) = \{r \in \mathrm{Runs}^{\mathcal{M}}(q) \,|\, \pi \text{ is prefix of } r\},$$

the *cylinder set* of π. We often abbreviate $\mathrm{Cyl}(\pi, \mathcal{M})$ to $\mathrm{Cyl}(\pi)$ if the context is clear.
We will sometimes use a "mixed" notation for paths and runs. For example, assume two paths $\pi_1 = q_1\ell_1q_2\ell_1 \ldots \ell_{n-1}q_n$ and $\pi_2 = s_1\ell_1s_2\ell_2 \ldots \ell_{m-1}s_m$ with $q_n \xrightarrow{\ell} s_1$; then we write $\pi_1\ell\pi_2$ as $\pi_1 \xrightarrow{\ell} \pi_2$.

3.2.2 Probability Measures for Markov Chains and MDPs

For Markov chains we define probability measures over sets of runs: first we define probabilities for cylinder sets, and then we extend the definition to obtain a probability measure for the smallest σ-algebra containing these sets.

Definition 11. (Probability measures for Markov chains)
Let $\mathcal{M} = (Q, \rightarrow, \mathrm{Lab})$ be a Markov chain. For every $q_0 \in Q$, we define a function M_{q_0} that maps cylinders $\mathrm{Cyl}(\pi, \mathcal{M})$, $\pi \in \mathrm{Paths}^{\mathcal{M}}$, to $[0, 1]$:

- if $\pi = q_0$ then $M_{q_0}(\mathrm{Cyl}(\pi, \mathcal{M})) := 1$,
- if $\pi = q_0 \xrightarrow{p_0,\ell_0} q_1 \xrightarrow{p_1,\ell_1} \dots \xrightarrow{p_{n-1},\ell_{n-1}} q_n$, then

$$M_{q_0}(\mathrm{Cyl}(\pi, \mathcal{M})) := p_0 \cdot M_{q_1}[\mathrm{Cyl}(q_1 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_{n-1}} q_n, \mathcal{M})],$$

- otherwise $M_{q_0}(\mathrm{Cyl}(\pi, \mathcal{M})) := 0$.

We denote by $\mathfrak{S}^{\mathcal{M}}$ the smallest σ-algebra that contains all sets $\mathrm{Cyl}(\pi)$ for $\pi \in \mathrm{Paths}(\mathcal{M})$. For each $q \in Q_P$, we define $\mathrm{Pr}_q^{\mathcal{M}} : \mathfrak{S}^{\mathcal{M}} \to [0, 1]$ as the unique extension of M_q to $\mathfrak{S}^{\mathcal{M}}$ that forms a probability measure over $\mathfrak{S}^{\mathcal{M}}$. □

For the construction of $\mathfrak{S}^{\mathcal{M}}$ and a proof of the uniqueness of $\mathrm{Pr}_q^{\mathcal{M}}$ we refer the reader to standard probability theory literature (e.g. Rosenthal [91]). The triple $(\mathrm{Runs}^{\mathcal{M}}, \mathfrak{S}^{\mathcal{M}}, \mathrm{Pr}_q^{\mathcal{M}})$ forms a probability space for every $q \in Q$. We often skip the superscript of $\mathrm{Pr}_q^{\mathcal{M}}$ if it is understood.
Let us again fix an MDP $\mathcal{M} = (Q_A, Q_P, \mathrm{Init}, \rightarrow, \mathrm{Lab}_A, \mathrm{Lab}_P)$ with $Q := Q_A \cup Q_P$. Before we can define probability measures for runs of MDPs we have to resolve the choices of the player. This is the task of a *strategy* (often also called policy):

Definition 12. A *strategy for* $\mathcal{M}$ is a function S that maps

- the empty path ϵ to a state $q_S \in \mathrm{Init}$,
- every path $\pi = q_0 \xrightarrow{\ell_0} q_1 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_{n-2}} q_{n-1} \xrightarrow{\ell_{n-1}} q_n \in \mathrm{Paths}^{\mathcal{M}}$ with $q_n \in Q_A$, to a distribution $d \in \mathrm{Dist}(\mathrm{Lab})$, such that for $\ell \in \mathrm{Lab}$, if $d(\ell) > 0$ then there exists $q \in Q$ with $q_n \xrightarrow{\ell} q$.

We denote the set of all strategies for $\mathcal{M}$ by $\mathcal{S}(\mathcal{M})$. □

A strategy represents the behaviour of the player: it inspects the history of a play and chooses a successor (maybe probabilistically). This behaviour allows us to construct a Markov chain representing the possible plays:

Definition 13. (Strategy-induced Markov chains)
Let S be a strategy for $\mathcal{M}$. We define $\mathcal{M}[S] := (\mathrm{Paths}^{\mathcal{M}}, \rightarrow_S, \mathrm{Lab})$, the Markov chain induced by S: states of $\mathcal{M}[S]$ are paths in $\mathcal{M}$, and for every state $\pi = q_0 \xrightarrow{\ell_0} q_1 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_{n-2}} q_{n-1} \xrightarrow{\ell_{n-1}} q_n \in \mathrm{Paths}^{\mathcal{M}}$ of $\mathcal{M}[S]$,

- if $q_n \in Q_A$ and $q_n \xrightarrow{\ell} q'$ with $S(\pi)(\ell) = p > 0$, then $\pi \xrightarrow{p,\ell}_S (\pi\ell q')$,
- if $q_n \in Q_P$ and $q_n \xrightarrow{p,\ell} q'$, then $\pi \xrightarrow{p,\ell}_S (\pi\ell q')$.

We define for every state $q \in Q$ a probability measure $\mathrm{Pr}_q^{\mathcal{M},S}$ over $\mathfrak{S}^{\mathcal{M}}$: for $\pi = q \xrightarrow{\ell_0} q_1 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_{n-2}} q_{n-1} \xrightarrow{\ell_{n-1}} q_n \in \mathrm{Paths}^{\mathcal{M}}$, let

$$\pi_0 = q, \pi_1 = q \xrightarrow{\ell_0} q_1, \pi_2 = q \xrightarrow{\ell_0} q_1 \xrightarrow{\ell_1} q_2, \dots, \pi_n = \pi$$

be the sequence of all prefixes of π. We set

$$\overline{\pi} := \pi_0 \xrightarrow{\ell_0} \pi_1 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_{n-1}} \pi_n \in \mathrm{Paths}^{\mathcal{M}[S]}(q_S)$$

and define

$$\mathrm{Pr}_{q_S}^{\mathcal{M},S}[\mathrm{Cyl}(\pi, \mathcal{M})] := \mathrm{Pr}_{q_S}^{\mathcal{M}[S]}[\mathrm{Cyl}(\overline{\pi}, \mathcal{M}[S])].$$

We write $\mathrm{Pr}^{\mathcal{M},S}$ for $\mathrm{Pr}_{q_S}^{\mathcal{M},S}$; If $\mathcal{M}$ is clear from the context we further abbreviate $\mathrm{Pr}^{\mathcal{M},S}$ to Pr^S. We call a path $\pi \in \mathrm{Runs}^{\mathcal{M}}$ *S-possible* in $\mathcal{M}$ iff $\mathrm{Pr}^S[\mathrm{Cyl}(\pi, \mathcal{M})] > 0$. Finally we write $\mathrm{Paths}^{\mathcal{M},S}$ (or Paths^S) for the set of all S-possible paths in $\mathcal{M}$. $\square$

3.3 Reachability Problems

We are now ready to formulate the problem of computing reachability probabilities, both for Markov chains and Markov decision processes.

Definition 14. (Reaching a state set)
Let $\mathcal{M} = (Q_A, Q_P, \mathrm{Init}, \rightarrow, \mathrm{Lab}_A, \mathrm{Lab}_P)$ be an MDP. Let $F \subseteq Q_A \cup Q_P$. A run $r = q_0 \xrightarrow{\ell_0} q_1 \xrightarrow{\ell_1} q_2 \xrightarrow{\ell_2} \dots \in \mathrm{Runs}^{\mathcal{M}}$ *reaches* F if there is a $k \in \mathbb{N}$ such that $q_k \in F$. The set of all initial runs reaching F is denoted by $\mathrm{Reach}(\mathcal{M}, F)$. $\square$

The following lemma implies that for every set of states F the set of runs of reaching F is *measurable*, i.e. an element of $\mathfrak{S}^{\mathcal{M}}$:

Lemma 1. *(Reaching a state set is measurable)*
Let $\mathcal{M}$ be an MDP. For every $F \subseteq V$, Reach$(\mathcal{M}, F) \in \mathfrak{S}^{\mathcal{M}}$. $\square$

Proof. Reach($\mathcal{M}, F$) can be written as a countable union of cylinders:

$$\text{Reach}(\mathcal{M}, F) = \bigcup_{\pi = q_0 \xrightarrow{\ell_0} q_1 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_{i-1}} q_i \in F} \text{Cyl}(\pi, \mathcal{M}).$$

□

We can now define the reachability problem in Markov chains.

Problem 1. *(Reachability in Markov chains)*
Given: *Markov chain* $\mathcal{M} = (Q, \rightarrow, Lab)$*, a state* $q_0 \in Q$*, and a set* $F \subseteq Q$*.*
Wanted: *The probability* $Pr^{\mathcal{M}}_{q_0}[Reach(\mathcal{M}, F)]$ *of reaching a state in* F*, the* reachability probability *of* F *in* $\mathcal{M}$ *starting at* q_0*.* □

In the case that Q is finite we can solve the reachability problem by computing the least nonnegative solution of a linear equation system with at most $|Q|$ unknowns. The structure of the equation system is captured by the equations occuring in the following lemma:

Lemma 2.
Let $\mathcal{M} = (Q, \rightarrow, Lab)$ *be a Markov chain, let* $F \subseteq Q$ *and* $q \in Q$*. Then*

$$Pr_q[Reach(\mathcal{M}, F)] = \begin{cases} 1 & \text{if } q \in F \\ \sum_{q \xrightarrow{p,\ell} q'} p \cdot Pr_{q'}[Reach(\mathcal{M}, F)] & \text{otherwise.} \end{cases} \tag{3.1}$$

□

Proof.
For $q \in F$, we get $\text{Pr}_q[\text{Reach}(\mathcal{M}, F)] \geq \text{Pr}_q[\text{Cyl}(q, \mathcal{M})] = 1$.

If $q \notin F$,

$$\begin{aligned}
&\Pr_q[\mathrm{Reach}(\mathcal{M}, F)] \\
&= \Pr_q\left[\bigcup_{\pi = q \xrightarrow{\ell_0} q_1 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_{i-1}} q_i \in F} \mathrm{Cyl}(\pi, \mathcal{M})\right] && \text{(Def. 14)} \\
&= \sum_{q \xrightarrow{p,\ell_0} q_1} p \cdot \Pr_{q_1}\left[\bigcup_{\pi = q_1 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_{i-1}} q_i \in F} \mathrm{Cyl}(\pi, \mathcal{M})\right] && \text{(Def. 11)} \\
&= \sum_{q \xrightarrow{p,\ell_0} q'} p \cdot \Pr_{q'}[\mathrm{Reach}(\mathcal{M}, F)].
\end{aligned}$$

□

For MDPs we have to provide a player strategy to obtain a probability measure. Different strategies can cause different probabilities of reaching a given state set. For applications we are in particular interested in the *minimum and maximum* of the reachability probabilities over all possible strategies:

Problem 2. *(Reachability in MDPs)*
Given: An MDP $\mathcal{M} = (Q_A, Q_P, Init, \to, Lab_A, Lab_P)$, and a state set $F \subseteq Q_A \cup Q_P$.
Wanted: The values

$$MinReach(\mathcal{M}, F) := \inf_{S \in \mathcal{S}(\mathcal{M})} Pr^{\mathcal{M},S}[Reach(\mathcal{M}, F)]$$

$$MaxReach(\mathcal{M}, F) := \sup_{S \in \mathcal{S}(\mathcal{M})} Pr^{\mathcal{M},S}[Reach(\mathcal{M}, F)],$$

the extremal reachability values *of F in $\mathcal{M}$. We call $MinReach(\mathcal{M}, F)$ the* minimal *and $MaxReach(\mathcal{M}, F)$ the* maximal *reachability value of F in $\mathcal{M}$.* □

If $\mathcal{M}$ has finitely many states, the computation of extremal reachability values can be carried out iteratively, e.g. using value iteration or policy iteration, or by solving a Linear Programming problem constructed from $\mathcal{M}$. For further details see e.g. Baier and Katoen [7], Puterman [89].
If $\mathcal{M}$ has no action states, i.e. it is a Markov chain, both extremal reachability values of Problem 2 coincide, and Problem 2 reduces to Problem 1.

Problem variants. Often it is not feasible to solve Problem 1 and Problem 2 exactly for the probabilistic systems we are investigating. Besides more practical reasons like too costly computations, quite fundamental obstacles arise, especially if we deal with infinite-state Markov chains and MDPs. For example, the reachability probabilities of systems we study in Chapter 6 might not even be representable by radicals.
Using the notation of Problem 1, instead of computing a reachability probability $Pr_q[\text{Reach}(\mathcal{M}, F)]$ [1] exactly, we therefore propose to compute solutions for the following two slightly weaker *problem variants*, which suffice for most applications:

1. Computing arbitrary precise *bounds*: given $\epsilon \in \mathbb{Q} \setminus \{0\}$, compute values $\{a, b\} \subseteq [0,1] \cap \mathbb{Q}$ such that $a \leq \Pr_q[\text{Reach}(\mathcal{M}, F)] \leq b$ and $b - a \leq \epsilon$.

2. *Almost-sure* reachability: Decide whether $\Pr_q[\text{Reach}(\mathcal{M}, F)] = 1$.

[1] or minimal and maximal reachability values of MDPs of a given set of states

Chapter 4

Reachability in Probabilistic Programs

In the current and the following chapter we study *probabilistic programs*. We use this term in the thesis for computer programs that can make use of a *probabilistic choice operator*. Such an operator chooses an alternative from a given finite set of commands or values according to a fixed probability distribution. Probabilistic programs are also equipped with a *nondeterministic choice operator*. It similarly chooses an alternative from a finite set, but here no additional information is given on how such an alternative is selected by the program. The semantics of a probabilistic program P can be given by an MDP $\mathcal{M}_P$ in which, roughly spoken, action states correspond to program states of P, nondeterministic choices are represented by action transitions, and probabilistic choices are represented by probabilistic transitions.

We study the problem of automatically computing minimal and maximal reachability probabilities for probabilistic programs. Often these programs are used to represent *reactive systems*, i.e. they interact with an *environment* by *receiving* input data and *reacting* to this data by creating output signals. The inputs received from the environment might be sensor signals or status messages from other programs or hardware etc. Often information about the behaviour of the environment is available in the form of probability distributions that describe the frequency with which specific input values occur. We can embed this information

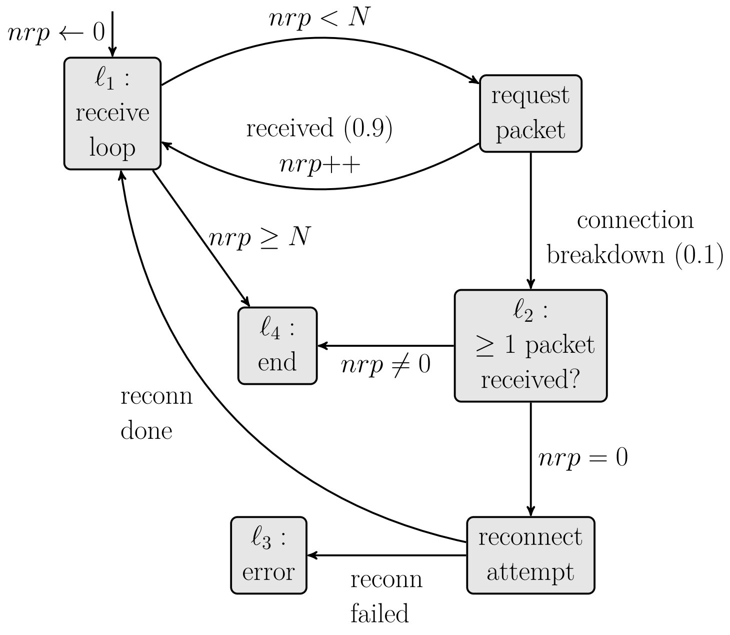


Figure 4.1: Packet transmission example, given as control flow graph.

into the program with the help of probabilistic choice operators. However, we might have no additional information about other aspects of the environment, e.g. in some situations we only know the possible input values, but have no information about the frequencies of their occurrence. This uncertainty can be modeled by introducing nondeterminism into the program.

We often want to know the probability of a fail situation, e.g. the probability of unsuccessfully trying to send a message over a network within at most k retry attempts. Since nondeterminism might occur in the program, this probability depends on the behaviour of the environment. Therefore we are interested in *minimal* and *maximal* probabilities of a fail situation to occur, hereby taking all the possible behaviours of the environment into account.

As a running example throughout the chapter we use the program in Fig. 4.1, which is given in form of a control flow graph. The goal of the program is to receive up to $N > 0$ packets through a network connection. The number of received packets is counted by the variable `nrp` (number of packets).

The transmission is handled in location ℓ_1. During each attempt of receiving a packet, the connection can break down with probability 0.1, in which case no further packet can be received; the program then exits the transmission loop and jumps to location ℓ_2. If N packets have been received in the transmission loop, the program jumps to ℓ_4, signaling successful termination.

If at least one packet is received when the program reaches ℓ_2, it terminates successfully by entering location ℓ_4.[1] Otherwise, the program tries to repair the connection, which may fail or succeed. In the former case the program jumps to ℓ_3, signaling an error, in the latter case the program is started again (at ℓ_1). The choice between success and failure is nondeterministic, since we do not have any information about the "success rate" of repairing a connection.

For assessing the reliability of the system we want to deduce the *minimum and maximum* over the probabilities of reaching the error location ℓ_3. They are dependent on whether the connection can be repaired or not in case of a connection breakdown. For this simple program, the smallest probability clearly corresponds to the situation where the the connection always can be repaired; in this case, we never reach the error location. The largest probability corresponds to the situation where all repair attempts fail; we then reach ℓ_3 with probability 0.1.

A way of automatically solving the reachability problem for a probabilistic program P is e.g to construct $\mathcal{M}_P$ and then to use value iteration or LP-solvers for computing the probabilities, as already mentioned in Chapter 2. However, the main problem we are facing when we follow this approach is *state explosion*: the state space of $\mathcal{M}_P$ can be very large or even infinite, making an explicit construction of $\mathcal{M}_P$ infeasible or impossible, a phenomenon well-known from nonprobabilistic automatic software verification. For example, if we set $N = 100$, the MDP corresponding to our example program has more than 500 states (see Fig. 4.3). For general probabilistic programs with infinite state spaces, the problem of computing reachability probabilities is even undecidable: probabilistic programs with integer variables are already Turing-powerful, and e.g. the halting problem can easily be reduced to a reachability problem. The only solution in this case is to rely on incomplete methods.

In this chapter we therefore present an approach for tackling these problems

[1]It would be more realistic to set another bound, like 20 packets, but with one packet the probabilities are easy to compute.

that is based on *abstraction techniques*. The abstract interpretation framework provides the general, mathematical basis of abstraction (Cousot and Cousot [25] and Cousot and Cousot [26]).

Our approach is an extension of frameworks developed in Kattenbelt et al. [62] and Wachter and Zhang [99]: instead of constructing the concrete MDP $\mathcal{M}_P$ we build a *stochastic 2-Player game arena* $\mathcal{G}$ that we call an *abstract game arena*. For the construction we use techniques from abstract interpretation. The structure of game arenas is similar to that of an MDP; however, the action states of $\mathcal{G}$ are controlled by one of *two* players instead of a single player. In $\mathcal{G}$ we can distinguish between nondeterminism inherent to the probabilistic program (modeled by the choices of Player 1) and nondeterminism introduced by abstraction effects (modeled by Player 2). States belonging to the first player in abstract arenas are *abstract states*, i.e. represent a *set of program states*. $\mathcal{G}$ can be considerably smaller than $\mathcal{M}_P$ and can be used to derive precise *lower and upper bounds* for the extremal reachability values of $\mathcal{M}_P$. The crucial idea is to consider *four different* games in $\mathcal{G}$ that differ in their winning objectives for the players (this was first considered in Kattenbelt et al. [62]). By computing the game values for these four games, we obtain an *upper* and a *lower* bound for the *minimal* reachability value as well as an *upper* and a *lower* bound for the *maximal* reachability value of $\mathcal{M}_P$. In the following we subsume all abstraction approaches using this idea under the term “game-based abstraction”.

Former approaches (D’Argenio et al. [29], Wachter et al. [100]) abstract probabilistic programs into MDPs and use them to compute a lower bound for the *minimal* reachability value and an upper bound for the *maximal* reachability value. Especially if the extremal reachability values of $\mathcal{M}_P$ differ considerably, these bounds alone are of little use: it is unclear in this situation whether the difference is caused by the abstraction or by nondeterminism of the considered program. With the additional bounds given by game-based abstraction approaches we obtain two intervals in which the probabilities are guaranteed to be contained; their width is a measure for the imprecision caused by the abstraction. Refinement techniques allow us to further improve these bounds, i.e., making the intervals tighter.

The game-based approaches of Kattenbelt et al. [62], Wachter and Zhang [99] rely on predicate abstraction, which partitions the state space into a finite number

of classes. In the context of general abstract interpretation theory this means that only domains with finitely many elements are used to build abstractions, and that the concretization of two distinct abstract states is always disjoint (the disjointness property). While predicate abstraction has proved very successful, it is known to have a number of shortcomings, for example potentially expensive equality and inclusion checks for abstract states, and “predicate explosion”. Our contribution consists in extending this approach to the general abstract interpretation framework, which allows us to use arbitrary domains (also such with infinitely many elements) and non-disjoint abstract states to construct abstract game arenas. This makes the approach more flexible and powerful, as we show by several examples. In particular we show how suitable widening operators can deduce loop invariants difficult to find for predicate abstraction. The proofs of Wachter and Zhang [99] use the disjointness property to easily define a Galois connection between the sets of functions assigning values to the abstract and the concrete states. There seems to be no easy way to adapt their proof to our construction. The framework from Kattenbelt et al. [62] builds game arenas that are structurally different from ours, and moreover they also depend on the disjointness property in their proof. Therefore we show the soundness of our approach by a new proof that uses additional techniques. We also provide refinements specifically tailored to the more general setting.
This chapter is partly based on joint work with Javier Esparza. Parts of the material have been published in Esparza and Gaiser [32].

Structure of the chapter. We first define probabilistic guarded programs, the program model that we use throughout the chapter. Then we will introduce some basic concepts from abstract interpretation theory and stochastic 2-Player games. Both are required prerequisites for building abstract game arenas of a probabilistic program. We sketch the structure of abstract game arenas and their construction using an example in Section 4.4, and also show how we can obtain bounds for reachability probabilities.
We give a formal definition of abstract game arenas for probabilistic programs in Section 4.5 and a precise explanation of our construction algorithm in Section 4.6. After that we discuss strategies for refining abstractions, and give several smaller

case studies in Section 4.8, pointing out the advantages of using the general abstract interpretation framework for building abstract game arenas. We close by comparing our approach with related work and give a short conclusion.

4.1 Probabilistic Programs

As program model we use *probabilistic guarded command programs* (*PGP*s for short). PGPs are a subset of the input language of the well-known finite-state probabilistic model checker PRISM (Kwiatkowska et al. [66]), and many protocols and randomized algorithms have been modeled as PGPs.[1] However, as an important extension we also allow variables with infinite range in our programs. This can lead to systems with infinitely many reachable states which are not supported by PRISM. PGPs are also essentially equivalent to the program model of concurrent probabilistic programs (Wachter [98]).

PGPs form a rich class of programs, supporting probabilistic choices, nondeterminism, and concurrency (through interleaving semantics). A PGP is given as a collection of guarded commands; every guarded command consists of a guard and a probabilistic update operation. Every PGP manipulates a set of program variables, which have a possibly infinite value range:

Definition 15. (Variables, states, and guards)
Let $\mathcal{V}$ be a finite set of variables, where $x \in \mathcal{V}$ has a finite or countable range $\mathrm{rng}(x)$. A *state* of $\mathcal{V}$ is a map $\sigma\colon \mathcal{V} \to \bigcup_{x\in\mathcal{V}} \mathrm{rng}(x)$ such that $\sigma(x) \in \mathrm{rng}(x)$ for all $x \in \mathcal{V}$. The sets of states is denoted by $\Sigma_{\mathcal{V}}$. $\square$

State changes are realized using *transitions*:

Definition 16. (Transitions, guards, and assignments)
A *transition* is a map $f \in 2^{\Sigma_{\mathcal{V}}} \to 2^{\Sigma_{\mathcal{V}}}$ such that $|f(\{\sigma\})| \leq 1$ for all $\sigma \in \Sigma_{\mathcal{V}}$ (i.e., a transition maps a single state to the empty set or to a singleton again), and

$$\bigcup_{\sigma\in M} f(\{\sigma\}) = f(M) \text{ for all } M \subseteq \Sigma_{\mathcal{V}}.$$

[1]see e.g. `http://www.prismmodelchecker.org/`.

A transition g is a *guard* if $g(\{\sigma\}) \in \{\{\sigma\}, \emptyset\}$ for every state σ. We say that σ *enables* g if $g(\{\sigma\}) = \{\sigma\}$.
A transition c is an *assignment* if $|c(\{\sigma\})| = 1$ for all $\sigma \in \Sigma_{\mathcal{V}}$. The semantics of an assignment c is the map $[\![c]\!] : \Sigma_{\mathcal{V}} \to \Sigma_{\mathcal{V}}$ given by $[\![c]\!](\sigma) := \sigma'$ if $c(\{\sigma\}) = \{\sigma'\}$.
The set of transitions is denoted by $\text{Trans}_{\mathcal{V}}$. □

Now we can give a complete definition of a PGP:

Definition 17. (Probabilistic Guarded Command Program)
A *probabilistic guarded command program* (PGP for short) is a triple $P = (\mathcal{V}, \sigma_0, \mathcal{C})$ where $\mathcal{V}$ is a finite set of program *variables*, $\sigma_0 \in \Sigma_{\mathcal{V}}$ is the *initial state*, and $\mathcal{C}$ is a finite set of *guarded commands*. A *guarded command* a has the form

$$a = g \to p_1 : c_1 + \ldots + p_m : c_m,$$

where $m \geq 1$, g is a guard, $p_1, \ldots, p_m$ are probabilities adding up to 1, and $c_1, \ldots, c_m$ are assignments. We call a tuple $\langle p_i, c_i \rangle$ an *update*. We denote the guard of a by g_a, the sequence of *updates* $\langle\langle p_1, c_1\rangle, \ldots, \langle p_m, c_m\rangle\rangle$ of a by up_a, and the set $\{up_a \mid a \in \mathcal{C}\}$ by $up_{\mathcal{C}}$.
We denote the probability of the i-th update of a (i.e., the first component of the i-th pair in up_a) by $p_a(i)$, and denote the i-th assignment by $c_a(i)$. □

We give an example of how we represent guarded commands in code listings:

```
ex: (x >= 0) & (y = 0) -> 0.25: (x' = x+1) & (y' = 0)
                        + 0.75: (x' = 0)
```

The name of the guarded command is ex, and the the value $p_{\text{ex}}(2)$ is equal to 0.75, for example. We write assignments in quote notation; in the first update, the value of `x` in the next step (denoted by `x'`) will be the old value of `x` plus one, and `y` will be set to 0. In the second update, no assignment for `y'` is given; we assume in this case that the value of the variable does not change.

4.1.1 PGP Semantics

A PGP can be seen as a description of an MPD with finite or countable state space:

Definition 18. (Semantics of PGPs)
The *MDP associated to* a PGP $P = (\mathcal{V}, \sigma_0, \mathcal{C})$ is

$$\mathcal{M}_P = (Q_A, Q_P, \{\sigma_0\}, \rightarrow, \mathcal{C} \cup \{\tau\}, \mathbb{N}),$$

where the set of action states Q_A is a subset of $\Sigma_{\mathcal{V}}$, and the set of probabilistic states Q_P is a subset of $Q_A \times \mathcal{C}$, and $\rightarrow \subseteq (Q_A \cup Q_P) \times (Q_A \cup Q_P)$.
Q_A, Q_P and $\rightarrow$ the are the smallest sets (respectively relation) such that $\sigma_0 \in Q_A$ and for every $\sigma \in Q_A$ the following holds:

1. If σ enables g_a, then $\langle \sigma, a \rangle \in Q_P$ and $\sigma \xrightarrow{a} \langle \sigma, a \rangle$. If σ does not enable *any* $a \in \mathcal{C}$, $\sigma \xrightarrow{\tau} \sigma$ holds.

2. If $\sigma \rightarrow \langle \sigma, a \rangle$ and $|up_a| = n$, then for every i with $1 \leq i \leq n$ it holds that $[\![c_a(i)]\!](\sigma) \in Q_A$, and

$$\langle \sigma, a \rangle \xrightarrow{p_a(i), i} [\![c_a(i)]\!](\sigma).$$

□

Reachability problem for PGPs. For a given PGP P and a guard f we want to know the minimum and maximum of the probabilities of reaching a state that enables f. Hereby we assume that no reachable program state σ enables both a guard of a guarded command in the program as well as f. More precisely:

Problem 3. *(Reachability problem for PGPs)*
Let $P = (\mathcal{V}, \sigma_0, \mathcal{C})$ be a PGP and f a guard, such that for all guarded commands $a \in \mathcal{C}$ holds that no $\sigma \in \Sigma_{\mathcal{V}}$ enables both f and g_a. The reachability problem *for P and f is then Problem 2 applied to $\mathcal{M}_P$ and the state set $F = \{\sigma \in Q_A \mid \sigma$ enables $f\}$, i.e. to compute both MinReach$(\mathcal{M}_P, F)$ and MaxReach$(\mathcal{M}_P, F)$. We call F the set of final states and f the* final guard. □

Assumption: We assume in the following that for every run $\pi \rightarrow \sigma \in \text{Paths}^{\mathcal{M}_P}$ either $\sigma \in F$ or σ enables the guard of at least one command (i.e., we do not "get stuck" during the computation: either a guard g_a of a guarded command

a is enabled or f is enabled). This can easily be achieved by adding a suitable guarded command to the program that simulates a self loop.
Recall our introductory example from Fig. 4.1. In Fig. 4.2 we give the example program in form of a PGP. The control flow is modeled by the variable `loc`. As we already pointed out, the probability of reaching ℓ_3 (i.e. loc=3 in the PGP) depends on the behaviour of the environment. Transferring our considerations from before to the PGP, the smallest respectively largest probability clearly corresponds to the environment choosing the guarded command `l3a`, respectively `l3b`, whenever possible, and its value is 0, respectively 0.1, independent of the concrete value of N. A brute-force automatic technique however will construct the MDP in Fig. 4.3. Note that the size of the MDP depends linearly on the size of N. We could also imagine setting N to ∞, which leads to an MDP with infinitely many states. In the following sections we introduce our abstraction technique, which does better and is able to infer tight intervals for both the smallest and the largest probability.

```
int nrp = 0, loc = 1;
l1a: (loc = 1) & (nrp < N)
      -> 0.9: (nrp' = nrp+1)
       + 0.1: (loc' = 2);
l1b: (loc = 1) & (nrp >= N) -> 1: (loc' = 4);
l2a: (loc = 2) & (nrp <> 0) -> 1: (loc' = 4);
l2b: (loc = 2) & (nrp = 0)  -> 1: (loc' = 1);
l2c: (loc = 2) & (nrp = 0)  -> 1: (loc' = 3);
l4:  (loc = 4) -> 1: (loc' = 4); // successful termination
reach: (loc = 3)   // error location
```

Figure 4.2: Packet transmission example, given as a PGP.

A note about conccurrency. PRISM possesses syntactic constructs for supporting concurrency; PGPs do not offer such operations explicitly. However, concurrent PRISM models can be converted to PGPs by a simple syntactic transformation (see *The PRISM Language - Semantics* [96]) that preserves the properties we are interested in.

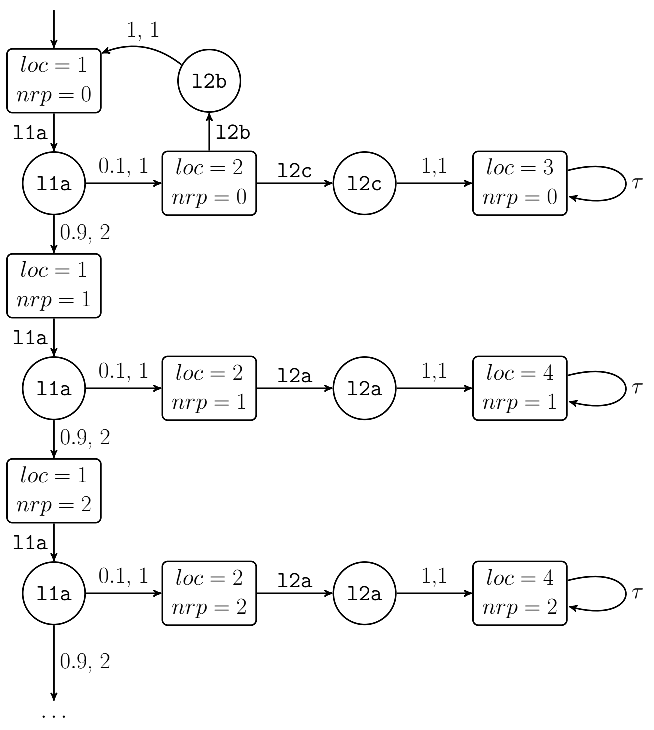


Figure 4.3: Prefix of the MDP from the packet transmission example. In probabilistic states $\langle \sigma, a \rangle$, σ is not displayed for lucidity. The single initial state is $\langle loc = 1, nrp = 0 \rangle$.

4.2 Abstracting Program States

In this section we introduce standard techniques from abstract interpretation (Cousot and Cousot [25],Cousot and Cousot [26]) for representing the state space of abstract game arenas and constructing their transitions. In the following section we define stochastic game arenas; using these prerequisites we can finally give concrete examples of abstract game arenas in Section 4.4. The abstract interpretation framework allows to design program analyses in a systematic and flexible way. The literature about abstract interpretation is vast, as are the techniques and tools developed within the framework. In particular one can rely on powerful libraries which facilitate constructing abstractions.

We fix a PGP $P = (\mathcal{V}, \sigma_0, \mathcal{C})$ for this section, and abbreviate $\Sigma_{\mathcal{V}}$ by Σ. We explain how to apply abstraction to program states of P. The main idea is not to use *concrete* program states as states of abstract game arenas, but rather *abstract states*. An abstract state can be interpreted as a set of (concrete) program states. We introduce all concepts tightly connected to our application; however, almost all material in this section is rather standard abstract interpretation theory, and readers familiar with these topics might safely omit it. For a general treatment see Cousot and Cousot [25, 26], Nielson et al. [81].

4.2.1 Domains

In abstract interpretation, abstract states are elements of completes lattices which are called *domains*:

Definition 19. (Domains)
A domain $D = (A, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$ is a complete lattice; the relation $\sqsubseteq$ is called the *safeness relation*. □

A particular important domain for us is $C = (2^{\Sigma}, \subseteq, \bigcup, \bigcap, \Sigma, \emptyset)$, which we call the *concrete domain*[1]. It contains all possible abstract states, since every subset of Σ is contained in it.
For an arbitrary domain $D = (A, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$, $\sqsubseteq$ is used to define *safe approximations*: for states s and s' from A, s is a safe approximation of s' whenever $s' \sqsubseteq s$. The choice of $\sqsubseteq$ depends on the property we are interested in. Roughly spoken, if $s' \sqsubseteq s$, then replacing s' by s in an abstraction yields a less precise, but still valid result.
C contains every possible abstract state, therefore it is in some sense the most precise domain. However, elements $M \in 2^{\Sigma}$ can be infinite and without any obvious regular structure. If we want to carry out analyses with finite-state computing devices, representing arbitrary M is therefore difficult or even impossible.
Instead of using the concrete domain for representing abstract states the abstract interpretation framework offers *abstract domains* $(D^{\sharp}, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$. We often abbreviate such a domain to $D^{\sharp}$, if $\sqsubseteq$ is clear from the context. Roughly spoken, every element in $D^{\sharp}$ corresponds also to an abstract state (or sets of abstract states), but can be represented efficiently by a finite-size description.

[1]In our case, this is the only concrete domain we are considering.

The downside of using an abstract domain $D^\sharp$ instead of C in analyses and constructions is that not every abstract state $M \subseteq 2^\Sigma$ is contained in $D^\sharp$, and so we might have to choose another state $d \in D^\sharp$ to represent M. The interplay between the elements of the concrete domain C and an abstract domain $D^\sharp$, i.e. what sets of states does an element $d \in D^\sharp$ correspond to and vice versa, can be described elegantly by a *Galois connection*, which consists of two functions, an *abstraction function* $\alpha : 2^\Sigma \to D^\sharp$ and a *concretization function* $\gamma : D^\sharp \to 2^\Sigma$:

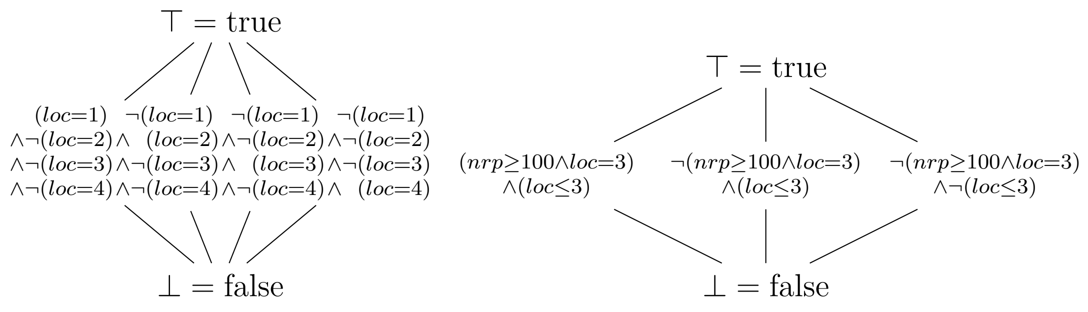
Definition 20. (Galois connection)
Let $(A, \sqsubseteq_A, \bigsqcup_A, \bigsqcap_A, \top_A, \bot_A)$ and $(B, \sqsubseteq_B, \bigsqcup_B, \bigsqcap_B, \top_B, \bot_B)$ be domains. A pair (α, γ) of monotone maps $\alpha : A \to B$ and $\gamma : B \to A$ forms a *Galois connection* between $(A, \sqsubseteq_A)$ and $(B, \sqsubseteq_B)$ iff the following two conditions hold:

1. $a \sqsubseteq_A \gamma(\alpha(a))$ for all $a \in A$,
2. $\alpha(\gamma(b)) \sqsubseteq_B b$ for all $b \in B$.

$\square$

Assume we have a Galois connection (α, γ) between C and $D^\sharp$. Condition (1) assures that when we map an element a from C to $D^\sharp$ by α and then back again by γ, the resulting element is always a safe approximation of a: We might lose information, but we stay safe. On the other hand, Condition (2) is a precision statement: Whenever we concretize an element b in C by γ, and abstract it afterwards by α, the resulting element is at least as precise as b. We assume for every abstract domain $D^\sharp$ the existence of a Galois connection (α, γ) between C and $D^\sharp$.

An example: predicate domains. As a first example we introduce predicate domains, which form the basis of program analyses based on predicate abstraction and counter-example guided abstraction refinement (Clarke et al. [19]), and essentially also of the game-based approaches described in Kattenbelt et al. [62], Wachter and Zhang [99].
Each element of the domain (except $\bot$ and $\top$, the minimal and maximal elements) is a class of a finite partition of the state space Σ. The partition is constructed



⊤ = true
(loc=1) ∧¬(loc=2) ∧¬(loc=3) ∧¬(loc=4)
¬(loc=1) ∧ (loc=2) ∧¬(loc=3) ∧¬(loc=4)
¬(loc=1) ∧¬(loc=2) ∧ (loc=3) ∧¬(loc=4)
¬(loc=1) ∧¬(loc=2) ∧¬(loc=3) ∧ (loc=4)
⊥ = false
⊤ = true
(nrp≥100∧loc=3) ∧(loc≤3)
¬(nrp≥100∧loc=3) ∧(loc≤3)
¬(nrp≥100∧loc=3) ∧¬(loc≤3)
⊥ = false

Figure 4.4: Hasse diagrams of predicate domains. The elements are represented as logical formulas. Note that e.g. on the right diagram no element $(nrp \geq 100 \wedge loc = 3) \wedge \neg(loc \leq 3)$ is drawn: the formula is a contradiction, and hence the element corresponding to it is $\perp$.

using a set $\phi_1, \ldots, \phi_k$ of predicates over Σ. Let $b_i \in \{\text{true}, \text{false}\}$ for $1 \leq i \leq k$. We define the sets

$$S(b_1, b_2, \ldots, b_k) := \{\sigma \in \Sigma \mid \phi_1(\sigma) \Leftrightarrow b_1 \wedge \phi_2(\sigma) \Leftrightarrow b_2 \wedge \ldots \wedge \phi_k(\sigma) \Leftrightarrow b_k\},$$

Let $\text{Part}(\phi_1, \ldots, \phi_k)$ be the set of elements $S(b_1, b_2, \ldots, b_k) \neq \emptyset$. $\text{Part}(\phi_1, \ldots, \phi_k)$ then forms a partition of Σ. We define the abstract domain

$$\mathbb{PRED}(\phi_1, \ldots, \phi_k) = (\{\top, \perp\} \cup \text{Part}(\phi_1, \ldots, \phi_k), \sqsubseteq, \bigsqcup, \bigsqcap, \top, \perp)$$

with $\top = \Sigma$ and $\perp = \emptyset$, by setting $a \sqsubseteq b$ iff $(a = \perp) \vee (a = b) \vee (b = \top)$ holds ($\bigsqcap, \bigsqcup$ follow from the definition of $\sqsubseteq$). It contains up to $2^k + 2$ elements.

As a concrete example we use the predicates $\phi_i := (loc = i)$ for $1 \leq i \leq 4$ to partition the state space of our introductory example in Fig. 4.2 and obtain the domain

$$L := \mathbb{PRED}(loc = 1, loc = 2, loc = 3, loc = 4).$$

This specific partition "distinguishes" only between control locations and does not preserve any information about `nrp`. We can represent its order structure by the Hasse diagram in Fig. 4.4 on the left. As another example we also give the Hasse diagram of the domain $\mathbb{PRED}(nrp \geq 100 \wedge loc = 3, loc \leq 3)$ in Fig. 4.4 on the right.

A Galois connection (α, γ) between C and $\mathbb{PRED}(\phi_1, \ldots, \phi_k)$ is given by defining

$$\gamma(\bot) := \emptyset, \gamma(\top) := \Sigma, \text{ and } \gamma(S) := S \text{ for } S \in \mathrm{Part}(\phi_1, \ldots, \phi_k).$$

and for $M \subseteq \Sigma$

$$\alpha(M) := \begin{cases} \bot & \text{if } M = \emptyset \\ S & \text{if there is } S \in Part(\phi_1, \ldots, \phi_k) \text{ such that } M \subseteq S \\ \top & \text{otherwise.} \end{cases}$$

It is easy to see (α, γ) indeed forms a Galois connection.
Note that for an arbitrary predicate domain $\mathbb{PRED}(\phi_1, \ldots, \phi_k)$ we can give a finite representation of every element $S(b_1, b_2, \ldots, b_k)$ as a k-bitvector. The "special cases“ $\bot$ and $\top$ can be represented by two additional symbols. The entire representation space is then $\{0,1\}^k \cup \{\bot, \top\}$. □

Another example: the interval domain. We describe the *interval* domain introduced in Cousot and Cousot [25], a classical example of an abstract domain. Let us assume that x is an arbitrary integer variable in P. We define for every $M \in \Sigma$ the set $M_x = \{\sigma(x) \mid \sigma \in M\}$. An element $M \subseteq \Sigma$ is represented in the interval domain (relative to x) by the smallest interval that contains every element in M_x. For example a set $M \subseteq \Sigma$ with $M_x = \{-2, 0, 10, 120\}$ is abstracted to the interval $[-2, 120]$. Note that e.g. a set $N \subseteq \Sigma$ with $N_x = \{-2, 0, 2, 4, 5, 6, \ldots\}$ is abstracted to $[-2, \infty)$, hence there is no one-to-one bijection between elements in C and corresponding intervals in the abstract domain. We set

$$\mathbb{INT}_x = (\mathrm{Int}, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot),$$

with

$$\mathrm{Int} = \{\emptyset, (-\infty, \infty)\} \cup \{[a, b] \mid a, b \in \mathbb{Z} \wedge a \leq b\} \cup \{(-\infty, b] \mid b \in \mathbb{Z}\} \cup \{[a, \infty) \mid a \in \mathbb{Z}\}$$

for the partial order that forms the interval domain. Hereby $\bot = \emptyset$ and $\top = (-\infty, \infty) = \mathbb{Z}$.

We choose the relation $\sqsubseteq$ as follows: $\bot \sqsubseteq x$ for all $x \in \text{Int}$, and $[a, b] \sqsubseteq [a', b']$ iff $a \geq a'$ and $b \leq b'$ for all $\{[a, b], [a', b']\} \subseteq \text{Int}$ (we use the usual conventions for comparing numbers with $\pm\infty$).

We can transfer the notion of safe approximation to $\mathbb{INT}_x$: the interval $[2, 10]$ is e.g. a safe approximation of $[3, 8]$, since $[3, 8] \sqsubseteq [2, 10]$ holds.

It remains to connect C and the abstract domain $\mathbb{INT}_x$ by a Galois connection. Note that $\mathbb{INT}_x$ itself does not contain any information about which variable it abstracts, i.e. that its elements represent values of x; this is the task of the Galois connection.

We define the *abstraction function* $\alpha : 2^\Sigma \to \text{Int}$ for every $M \in 2^\Sigma$ as follows:

$$\alpha(M) := \begin{cases} \bot & \text{if } M = \emptyset \\ [\min M_x, \max M_x] & \text{if } M_x \neq \emptyset \text{ and finite} \\ (-\infty, \max M_x] & \text{if } M_x \cap -\mathbb{N} \text{ infinite and } M_x \cap \mathbb{N} \text{ finite} \\ [\min M_x, \infty) & \text{if } M_x \cap -\mathbb{N} \text{ finite and } M_x \cap \mathbb{N} \text{ infinite} \\ \mathbb{Z} & \text{otherwise.} \end{cases}$$

We give a *concretization function* $\gamma : \text{Int} \to 2^\Sigma$ that maps an element I of Int to the largest set of states that itself gets abstracted to I:

$$\gamma(I) := \begin{cases} \emptyset & \text{if } I = \bot \\ \{\sigma \in \Sigma \mid a \leq \sigma(x) \leq b\} & \text{if } I = [a, b], a, b \in \mathbb{Z} \\ \{\sigma \in \Sigma \mid \sigma(x) \leq b\} & \text{if } I = (-\infty, b], b \in \mathbb{Z} \\ \{\sigma \in \Sigma \mid a \leq \sigma(x)\} & \text{if } I = [a, \infty), a \in \mathbb{Z} \\ \mathbb{Z} & \text{otherwise.} \end{cases}$$

It is easy to see that (α, γ) is a Galois connection between C and $\mathbb{INT}_x$. A finite representation of an element in Int can be given by two values a and b with $\{a, b\} \in \mathbb{Z} \cup \{\infty, -\infty\}$, representing the bounds of the interval. $\bot$ can e.g. be expressed by any tuple (a, b) with $a > b$. The representation for $\mathbb{INT}_x$ is therefore $(\mathbb{Z} \cup \{\infty, -\infty\})^2$. $\square$

```
int x = 0;
a: (x >= 0) -> 0.5: (x'=x+2) + 0.5: (x'= -1);
reach: (x = -1)
```

Figure 4.5: PGP example for illustrating the use of widenings.

4.2.2 Abstract Transitions

Besides fixing a domain we also have to build abstract versions of transitions for computing successors of abstract states:

Definition 21. (Abstract transitions)
Let $D = (A, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$ be a domain, and (α, γ) a Galois connection between C and D. Let further $f : C \to C$ be a transition. A monotone function $f^\sharp : A \to A$ is an *abstract transition for f via* (α, γ) if

$$\text{for all } a \in A, f(\gamma(a)) \subseteq \gamma(f^\sharp(a)) \text{ holds.}$$

The *best abstract transition* of f is $\alpha \circ f \circ \gamma$. $\square$

Intuitively, results obtained by evaluating an abstract transition $f^\sharp$ of f provide (over-)approximations for results of f (note that $f^\sharp$ does not necessarily represent the *best* abstract transition).
Computing the best abstract transition for a complex transition is often expensive. However, we can give abstractions for simple assignments and guards and define abstractions for more complicated guards by composition.
For illustrating abstract transitions we use the simple PGP in Fig. 4.5. The guard of a is given by $g_a : 2^{\mathbb{Z}} \to 2^{\mathbb{Z}}$ with $g(M) = \{\sigma \in M \mid \sigma(x) \geq 0\}$. The first update of the guarded command can be given by the transition $u_1 : 2^{\mathbb{Z}} \to 2^{\mathbb{Z}}$ with $u_1(M) = \{\sigma \mid \exists \sigma' \in M : \sigma(x) = \sigma'(x) + 2\}$ for $M \subseteq \mathbb{Z}$. The effect of taking the first update of a is given by the transition t with $t := u_1 \circ g$. We choose the interval domain $\mathbb{INT}_x$; an abstract transition $g_A^\sharp : \mathrm{Int} \to \mathrm{Int}$ for g_A is given by $g_A^\sharp(M) = M \sqcap [0, \infty)$. An abstract transition of u_1 is given by $u_1^\sharp : \mathrm{Int} \to \mathrm{Int}$ such that for $a \leq b$ with $\{a, b\} \in \mathbb{Z} \cup \{-\infty, \infty\}$, $u_1^\sharp([a, b]) = [a + 2, b + 2]$ and $u_1^\sharp(\bot) = \bot$. By combining the two functions we obtain the abstract transition $t^\sharp = u_1^\sharp \circ g_A^\sharp$ for t.

4.2.3 Fixed Points and Widenings

Computing fixed points is an essential task in abstract interpretation (and program analysis in general). We denote by σ_0 the initial state of the example PGP in Fig. 4.5, i.e. $\sigma_0(x) = 0$, and set $s_0 = \alpha(\{s_0\}) = [0, 0]$. We can use $t^\sharp$ to build (abstract) successor states of s_0 by applying $t^\sharp$, and so approximate a part of the reachable state space of $\mathcal{M}_P$. However, using a naive approach, we run into problems: starting with s_0, we generate

$$s_0, t^\sharp(s_0), t^\sharp(t^\sharp(s_0)), t^\sharp(t^\sharp(t^\sharp(s_0))), \ldots = [0, 0], [2, 2], [4, 4], [6, 6], \ldots,$$

which are the successor states if we always take the first update of A. This sequence however contains infinitely many elements, and therefore our computation would never stop. Note that this effect occurs although we are using the interval domain (every abstract state of the sequence in fact represents exactly one concrete state). Let R_t be the set of all program states reachable from σ_0 by applying t repeatedly. A first idea to tackle this problem is, as a rough over-approximation, to compute *one* abstract state that subsumes all states in R_t. We can write $R_t = \bigcup_{i \geq 0} t^i(\{\sigma_0\})$, and can also characterize it as the least solution of the equation system

$$X = \{\sigma_0\} \cup X \cup t(X). \tag{4.1}$$

We now build an "abstract version" of this equation:

$$X = s_0 \sqcup X \sqcup t^\sharp(X). \tag{4.2}$$

Every solution of Eq. 4.2 is also a fixed point of the monotone function $f(X) = s_0 \sqcup X \sqcup t^\sharp(X)$. By the Knaster-Tarski theorem (Theorem 1) f has a least fixed point μ_f, which is also the smallest solution of Eq. 4.2. Using the properties of Galois connections and abstract transitions one can prove that for μ_f it holds that $R_t \subseteq \gamma(\mu_f)$. Since in this case f is even continuous with respect to our chosen domain, we can also apply Kleene's theorem (Theorem 2) and get that $\mu_f = \lim_{i\to\infty} f^i(\bot)$. We conclude that instead of solving Eq. 4.1 it is sufficient to find an element m in our domain such that $\mu_f \sqsubseteq m$: $\gamma(m)$ is then a safe over-approximation of R_t.

Unfortunately, we cannot compute μ_f by keeping computing iterates $f^i(\bot)$ for increasing i in general, since the sequence might strictly increase as in our example and never reach R_t.
The approach often used in abstract interpretation in such situations is to apply a *widening operator*: It overapproximates the iterates $f^i(\bot)$, and we obtain a new sequence that is guaranteed to be stationary, and whose limit is still a safe over-approximation of μ_f.

Definition 22. (Widening Operator)
Let $D = (A, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$ be a domain. A function $\nabla : A \times A \to A$ is a *widening operator* for D if

- $a \sqsubseteq a \nabla b$ and $b \sqsubseteq a \nabla b$ for all $a, b \in A$,
- for every strictly increasing sequence $a^{(1)} \sqsubset a^{(2)} \sqsubset \ldots$ in A the sequence $(b^{(i)})_{i \in \mathbb{N}}$ defined by $b^{(1)} = a^{(1)}$ and $b^{(i+1)} = b^{(i)} \nabla a^{(i+1)}$ for $i \geq 1$ is stationary, i.e. there exists a $j \in \mathbb{N}$ such that $b^{(j)} = b^{(j+1)} = b^{(j+2)} = \ldots$.

$\square$

The classical widening for the interval domain (Cousot and Cousot [25]) is given by $\nabla : \text{Int} \times \text{Int} \to \text{Int}$ with $[a, b] \nabla [a', b'] = [s, t]$, with $s = -\infty$ if $a < a'$, otherwise $s = a$, and $t = \infty$ if $a' < a$, otherwise $t = b$. Note that for predicate domains the usual $\sqcup$-operator is trivially a widening, since every monotone sequence in a domain with finitely many elements is stationary. The following theorem characterizes the over-approximating sequence and its convergence properties. A proof can be found e.g. in Nielson et al. [81] (Prop. 4.13), where a slightly stronger result is shown.

Theorem 3. *(Widening sequences)*
Let $D = (A, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$ be a domain, $\nabla : A \times A \to A$ a widening operator *for D, and $f : A \to A$ a monotone function with respect to D. The sequence $(w^{(i)})_{i \in \mathbb{N}}$ with $w^{(1)} = \bot$, $w^{(i+1)} = w^{(i)} \nabla f(w^{(i)})$ for $i \geq 1$ is stationary, i.e. there is a $j \in \mathbb{N}$ such that for every $k \geq j$ it holds that $w^{(k)} = w^{(j)}$. The limit $w(f, \nabla)$ of the sequence satisfies $\mu_f \sqsubseteq w(f, \nabla)$ with μ_f the least fixed point of f.* $\square$

For our example we compute the sequence $(w^{(i)})_{i \in \mathbb{N}}$ in $\mathbb{INT}_{\mathtt{x}}$:

- $w^{(1)} = \bot$,
- $w^{(2)} = \bot \nabla f(\bot) = [0,0]$,
- $w^{(3)} = [0,0] \nabla f([0,0]) = [0,0] \nabla [0,2] = [0,\infty)$,
- $w^{(4)} = [0,\infty) \nabla f([0,\infty)) = [0,\infty)$.

We know for sure that $\mu_f \sqsubseteq [0,\infty)$ (in this case, even equality holds). We can transfer this result back into the concrete domain C: $\gamma([0,\infty)) = \{v \in \mathbb{Z} \mid v \geq 0\}$ is a safe over-approximation of the exact solution $\{v \in \mathbb{Z} \mid v \geq 0 \wedge 2|v\}$ of states that are reachable by just taking the first update of a.

4.2.4 Direct Product of Domains

Abstract interpretation theory allows to combine domains for improving the precision of the abstractions. A simple combination operator is the *direct product* of domains (see e.g. Nielson et al. [81]).

Definition 23. (Direct product of domains)
Let $D_A = (A, \sqsubseteq_A, \bigsqcup_A, \bigsqcap_A, \top_A, \bot_A)$ and $D_B = (B, \sqsubseteq_B, \bigsqcup_B, \bigsqcap_B, \top_B, \bot_B)$ be domains, with (α_A, γ_A) a Galois connection between C and D_A and (α_B, γ_B) a Galois connection between C and D_B. We define the *direct product domain* $D_A \times D_B := (A \times B, \sqsubseteq)$ by setting for (a,b) and (a',b') in $A \times B$:

$$(a,b) \sqsubseteq (a',b') \text{ iff } a \sqsubseteq_A a' \wedge b \sqsubseteq_B b'.$$

We define a Galois connection (α, γ) between C and $(A \times B, \sqsubseteq)$ by setting for $M \in 2^\Sigma$

$$\alpha(M) = (\alpha_A(M), \alpha_B(M))$$

and for $(x,y) \in A \times B$

$$\gamma((x,y)) = \gamma_A(x) \cap \gamma_B(y).$$

□

It is easy to prove that $D_A \times D_B$ forms again a complete lattice and (α, γ) is a Galois connection (Nielson et al. [81]).

Partition domain $\times$ Interval domain. We revisit our example from Fig. 4.2. We already developed $L = \mathbb{PRED}(loc = 1, loc = 2, loc = 3, loc = 4)$, an abstract domain providing information about the value of the `loc`-variable.
By building the direct product of L and the interval domain $\mathbb{INT}_{\texttt{nrp}}$ for Fig. 4.2 we obtain $\text{PD} = L \times \mathbb{INT}_{\texttt{nrp}}$. Its elements can be represented as tuples $\langle x, y\rangle \in (\{0,1\}^4 \cup \{\bot, \top\}) \times (\mathbb{Z} \cup \{\infty, -\infty\})^2$. We give a widening ∇ for PD by defining for every (x_1, x_2) and (y_1, y_2) in PD

$$(x_1, x_2) \nabla (y_1, y_2) := (x_1 \sqcup x_2, y_1 \nabla y_2).$$

$\square$

4.2.5 Other Abstract Domains

A special feature of the abstract interpretation framework is its wealth of existing domains with different strengths regarding precision and performance. Program analyzers can be tailored to the needs of the considered analysis problem by choosing (or designing) a suitable domain.
We use in our case studies

- the polyhedra domain (Cousot and Halbwachs [27]), which is able to represent its elements by using arbitrary linear inequalities of program variables;
- the octagon domain (Miné [74]), which uses a restricted class of linear inequalities of the form $\pm x \pm y \leq c$ for x, y program variables and c a constant to represent its elements; therefore it is less expressive than the polyhedra domain, but abstract transitions can be computed faster and representation of elements is often cheaper;
- the already introduced interval domain, which uses only linear inequalities of the form $x \leq c$ for x a program variable and c a constant for representing its elements; and
- the integer grid domain (Bagnara et al. [5]), which represents elements by congruential equations.

4.3 Stochastic Games

After describing abstract states we introduce in this section basics of *stochastic 2-Player game arenas* and *stochastic 2-Player games.* Stochastic 2-player game arenas are the formal model of what we call abstract game arenas for PGPs; given an abstract game arena $\mathcal{G}$ for a PGP P, we can obtain bounds for extremal reachability values of $\mathcal{M}_P$ by investigating four different *games* (and their game values) that are all played within $\mathcal{G}$, but are differing in the winning objectives of both players. We explain the games and how to obtain bounds from them in detail in Section 4.4.3.
Stochastic 2-Player game arenas (game arenas for short) are similar to MDPs; however, action states are partitioned into two classes 1 and 2: states belonging to class 1 are controlled by player 1, those of class 2 by player 2. For a more thorough introduction into the subject and proofs for the theorems see e.g. Condon [21] and Condon [20].

Definition 24. A *stochastic 2-Player game arena* $\mathcal{G}$ (short game arena) is a tuple $(Q_1, Q_2, Q_P, s_0, \rightarrow, \mathrm{Lab}_A, \mathrm{Lab}_P)$, where

- Q_1, Q_2, Q_P are distinct, finite sets of states, we set $Q = Q_1 \cup Q_2 \cup Q_P$,
- $s_0 \in Q$ is the *initial state*,
- Lab_A respectively Lab_P is a countable set of *action labels* respectively *probabilistic labels*,
- the relation $\rightarrow$ is equal to $\rightarrow_1 \cup \rightarrow_2 \cup \rightarrow_P$, where
  1. $\rightarrow_1 \subseteq Q_1 \times \mathrm{Lab}_A \times Q$ is a set of *player 1 transitions*,
  2. $\rightarrow_2 \subseteq Q_2 \times \mathrm{Lab}_A \times Q$ is a set of *player 2 transitions*, and
  3. $\rightarrow_P \subseteq Q_P \times (0, 1] \times \mathrm{Lab}_P \times Q$ is a set of *probabilistic transitions.*

The relation $\rightarrow$ satisfies the following conditions:

- if (q, p, ℓ, q') and (q, p', ℓ, q') are probabilistic transitions, then $p = p'$,
- if (q, ℓ, q_1) and (q, ℓ, q_2) are player transitions, then $q_1 = q_2$,

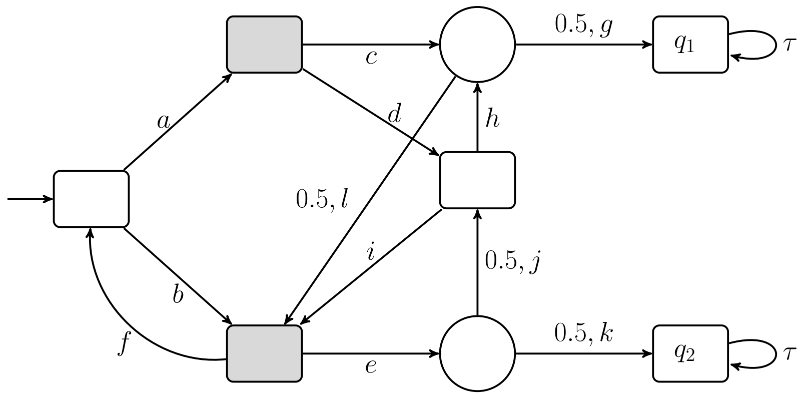


Figure 4.6: Example stochastic game arena. Probabilistic states are drawn as circle-shaped states, player states have rectangle shape; white states belong to Player 1, gray states belong to Player 2.

- the probabilities of the outgoing transitions of a probabilistic state add up to 1, i.e. for every $q \in Q_P$, $\sum_{(q,p,\ell,q') \in \rightarrow} p = 1$,
- every state of Q_1 resp. Q_2 has at least one successor in $\rightarrow_1$ resp. $\rightarrow_2$.

□

Note that, in contrast to MDPs, we have exactly one initial state in our definition of stochastic 2-Player game arenas. Fig. 4.6 shows an example: rectangle-shaped states represent action states; white ones are controlled by player 1, gray ones by player 2.

We fix a game arena $\mathcal{G} = (Q_1, Q_2, Q_P, s_0, \rightarrow, \mathrm{Lab}_A, \mathrm{Lab}_P)$ for the rest of the section. Similarly to MDPs, a *run* of $\mathcal{G}$ is an infinite word $r = q_0\ell_0q_1\ell_1 \ldots \in (Q\mathrm{Lab})^\omega$ such that for all $i \geq 0$ either $q_i \xrightarrow{p,\ell_i} q_{i+1}$ for some $p \in (0,1]$ or $q_i \xrightarrow{\ell_i} q_{i+1}$; r is initial if $q_0 = s_0$. We will often use the term *play* for runs in a game arena. The set of runs starting at a state q is $\mathrm{Runs}^{\mathcal{G}}(q)$, the set of all runs starting at s_0 is denoted by $\mathrm{Runs}^{\mathcal{G}}$.

A *path* is again a proper prefix of a run, $\mathrm{Paths}^{\mathcal{G}}(q)$ the set of all paths starting at q, and $\mathrm{Paths}^{\mathcal{G}}$ the set of all paths starting at q_0.

For each path $\pi \in \mathrm{Paths}^{\mathcal{G}}(q)$ with $q \in Q$, we again define

$$\mathrm{Cyl}(\pi, \mathcal{G}) = \{r \in \mathrm{Runs}^{\mathcal{G}}(q) \mid \pi \text{ is prefix of } r\},$$

the *cylinder set* of π, and abbreviate $\mathrm{Cyl}(\pi, \mathcal{G})$ to $\mathrm{Cyl}(\pi)$ if the context is clear. We define $\mathfrak{S}^{\mathcal{G}}$ as smallest σ-algebra containing the cylinder sets of $\mathcal{G}$.
The behaviours of Player 1 and 2 in $\mathcal{G}$ are described using a *pair* of strategies:

Definition 25. (Strategies for game arenas)
Let $i \in \{1, 2\}$. A *strategy for player i* in $\mathcal{G}$ is a function S that maps each path $\pi = q_0 \xrightarrow{\ell_0} q_1 \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_{n-2}} q_{n-1} \xrightarrow{\ell_{n-1}} q_n \in \mathrm{Paths}^{\mathcal{G}}$ with $q_n \in Q_i$ to a distribution $d \in \mathrm{Dist}_F(\mathrm{Lab})$, such that for $\ell \in \mathrm{Lab}_A$, if $d(\ell) > 0$ then there exists $q \in Q$ with $q_n \xrightarrow{\ell} q$.
We denote by $\mathcal{S}_i(\mathcal{G})$ the set of strategies for player i in $\mathcal{G}$. □

Since there is only one initial state in a game arena, strategies do not have to select an initial state. By fixing strategies for both player 1 and player 2, we can define their *induced* Markov chain, similarly to MDPs:

Definition 26. (Strategy-induced Markov chains for game arenas)
Let S_i be a strategy for Player i in $\mathcal{G}$, $i \in \{1, 2\}$. We define the *Markov chain* $\mathcal{G}[S_1, S_2] = (\mathrm{Paths}^{\mathcal{G}}, q_S, \rightarrow_S, \mathrm{Lab})$, i.e., states of the Markov chain are paths in $\mathcal{G}$. For every state $\pi = q_0 \xrightarrow{\ell_0} q_1 \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_{n-2}} q_{n-1} \xrightarrow{\ell_{n-1}} q_n \in \mathrm{Paths}^{\mathcal{G}}$ and $i \in \{1, 2\}$,

- if $q_n \in Q_i$ and $q_n \xrightarrow{\ell} q'$ with $S_i(\pi)(\ell) = p > 0$, then $\pi \xrightarrow{p,\ell}_S (\pi \ell q')$,
- if $q_n \in Q_P$ and $q_n \xrightarrow{p,\ell} q'$, then $\pi \xrightarrow{p,\ell}_S (\pi \ell q')$.

We define a probability measure $\mathrm{Pr}_{s_0}^{\mathcal{G},(S_1,S_2)}$ over $\mathfrak{S}^{\mathcal{G}}$ analogously to MDPs. We write $\mathrm{Pr}^{\mathcal{G},(S_1,S_2)}$ or $\mathrm{Pr}^{(S_1,S_2)}$ for $\mathrm{Pr}_{s_0}^{\mathcal{G},(S_1,S_2)}$ if the context is clear. We call a path $\pi \in \mathrm{Runs}^{\mathcal{G}}$ (S_1, S_2)*-possible* in $\mathcal{G}$ obeying (S_1, S_2) iff $\mathrm{Pr}^{(S_1,S_2)}[\mathrm{Cyl}(\pi, \mathcal{G})] > 0$. We write $\mathrm{Paths}^{\mathcal{G},(S_1,S_2)}$ (abbreviated $\mathrm{Paths}^{(S_1,S_2)}$) for the set of all possible paths in $\mathcal{G}$ obeying (S_1, S_2). □

Similarly to MDPs, for a given $F \subseteq Q$, we can also define the set $\mathrm{Reach}(\mathcal{G}, F)$ of runs reaching F in $\mathcal{G}$.
Before we are able to define games, we have to introduce *winning objectives.* They are used to decide whether a player has won a play:

Definition 27. (Winning objective)
A *winning objective* for a game arena $\mathcal{G}$ is a set $W \subseteq \mathrm{Runs}^{\mathcal{G}}(s_0)$ with $W \in \mathfrak{S}^{\mathcal{G}}$. □

A game combines an arena with winning objectives for both players:

Definition 28. (Stochastic 2-Player Game)
A *stochastic 2-Player game* (game for short) is a tuple $(\mathcal{G}, W_1, W_2)$, with $\mathcal{G}$ a stochastic 2-Player game arena and W_1, W_2 winning objectives for $\mathcal{G}$. We call W_1 (resp. W_2) the winning objective of Player 1 (resp. Player 2). We say player i *wins* a play r if $r \in W_i$, for $i \in \{1, 2\}$. □

Very often it is the case that $W_2 = \text{Runs}^{\mathcal{G}} \setminus W_1$ holds for a game $(\mathcal{G}, W_1, W_2)$, i.e. if one player loses, then the other one wins. However, we will also use games where $W_1 = W_2$, i.e. always both player win or both lose. If e.g. $W_1 = \text{Reach}(\mathcal{G}, F)$ for $F \subseteq Q$, Player 1 wins the play as soon as it *reaches* a state in F, and loses if that is never the case. If $W_2 = \text{Runs}^{\mathcal{G}} \setminus \text{Reach}(\mathcal{G}, F)$, the winning objective of Player 2 is to *avoid* any state in F. For example, a possible winning objective for Player 1 in the game arena of Fig. 4.6 is reaching q_1 and Player 2's objective might be to avoid reaching q_1.

4.4 The Approach in a Nutshell

We sketched in Section 4.2 how sets of concrete states of a PGP P can be subsumed into abstract states using the abstract interpretation framework, and introduced stochastic game arenas in Section 4.3.
In this section we combine both concepts and give the core contributions of the entire chapter in a nutshell: we explain the structure and intuition behind *abstract game arenas* for PGPs in Section 4.4. An abstract game arena is a stochastic 2-Player game arena whose states and transitions are constructed using abstract interpretation techniques. We illustrate the construction using an example in Section 4.4.2.
The technique of using stochastic 2-Player games (game-based approach) for abstracting probabilistic programs has first been developed in Kattenbelt et al. [62]. The structure of our abstract game arenas corresponds to a variant of the game-based approach called *parallel* or *menu-based* abstraction from Wachter [98], Wachter and Zhang [99]. We note that game arenas constructed e.g. by Kattenbelt et al. [62] have a somehow different structure (e.g. the roles of the players are

reversed); however, the underlying principle is the same. All these frameworks restrict themselves to predicate domains. Our contribution lies in extending their approaches to allow the use of arbitrary (in particular infinite) domains for abstract arenas, allowing in particular concretizations of abstract states to overlap. We sketch why using arbitrary domains can offer advantages compared to predicate abstraction in Section 4.4.4. After having introduced the most important concepts rather informally in this section, we give a formal definition of abstract game arenas and correctness proofs in Section 4.5; in Section 4.6 we give our construction algorithm and prove its correctness and termination.

We again fix a PGP $P = (\mathcal{V}, \sigma_0, \mathcal{C})$ and a final guard f. Let F be the final states corresponding to f. An *abstract game arena* $\mathcal{G}$ of P relative to f is a stochastic 2-Player game arena that captures behaviour of $\mathcal{M}_P$ as we shall point out in the following.

States of Player 1 in $\mathcal{G}$ are abstract states (representing a set of concrete states of program states), states of Player 2 in $\mathcal{G}$ are tuples of actions and abstract states. $\mathcal{G}$ also contains a "goal state", denoted by $\circledcirc$, and a "reject" state $\otimes$[1] (plus possibly additional states). The nondeterminism inherent to the system is modeled by the choices of Player 1, nondeterminism introduced by the abstraction is modeled by the choices of Player 2.

4.4.1 A Game Round

A play in an abstract game arena is partitioned into "game rounds" (see Fig. 4.8 for an example arena): A round starts at an abstract state n (we think of n as a set of concrete states) with Player 1 to move. Let n_i be the set of concrete states of n that enable the command a_i. Player 1 proposes an a_i such that $n_i \neq \emptyset$, modeled by a move from n to a state $\langle n, a_i \rangle$. If n contains some final state, then Player 1 can also propose to end the play (modeled by a move to $\langle n, \circledcirc \rangle$). Then it is Player 2's turn. If Player 1 proposes a_i, then Player 2 can accept the proposal (modeled by a move to a state determined below), or reject it and end the play (modeled by a move to $\otimes$), but only if $n_i \neq n$. Let $up_{a_i} = \langle \langle p_1, c_1 \rangle, \ldots, \langle p_k, c_k \rangle \rangle$ be the updates of a_i.

[1]$\otimes$ has a similar role as the $\star$-state in menu-based abstraction.

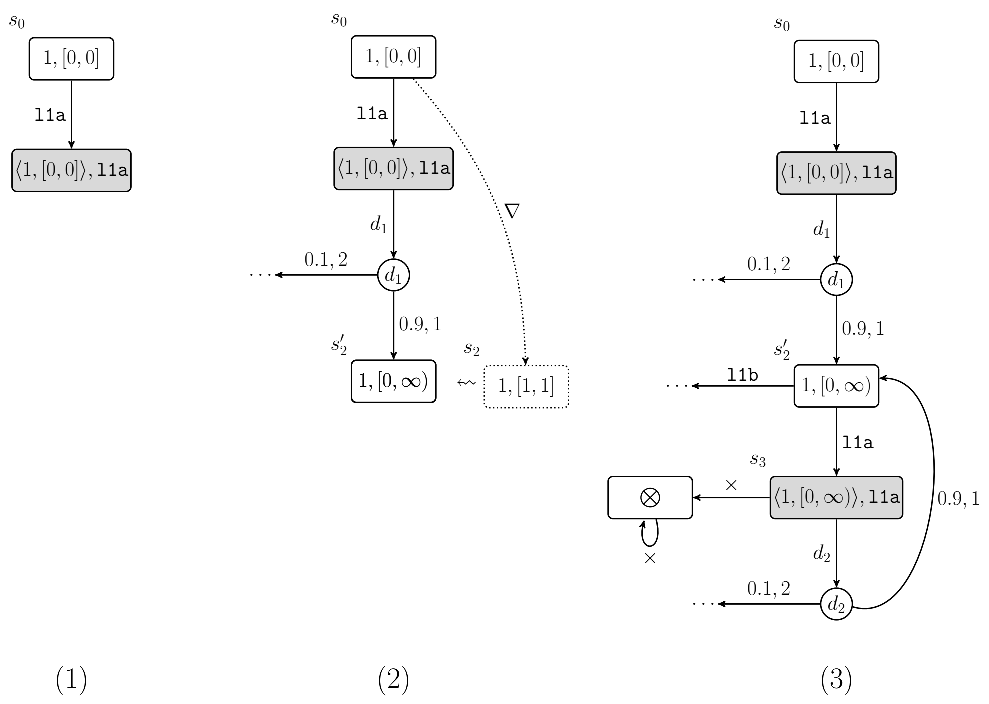


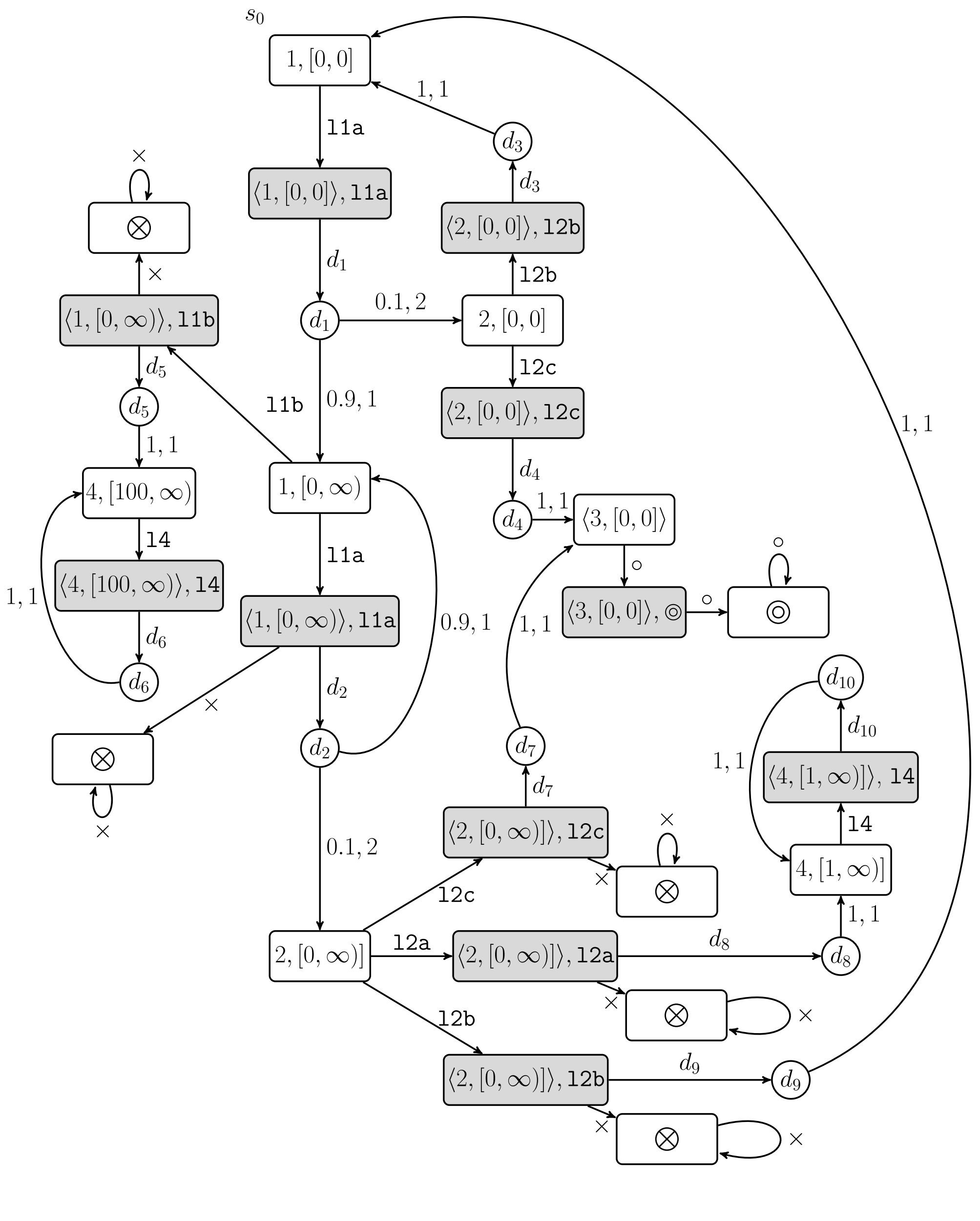
Figure 4.7: Construction of an abstract game arena for the example. The three steps are described in the text.

If Player 2 accepts a_i, she "picks" a concrete state $\sigma \in n_i$.

Then she selects a probabilistic state q with the following property: q has k successors $s_1, \ldots, s_k$, representing the outcome of performing the assignment c_j on σ, i.e. $[\![c_j]\!](\sigma)$ is contained in s_j for every $1 \leq j \leq k$.

The next state of the play is determined probabilistically: one of the updates j of a_i is selected randomly according to the probabilities, and the play moves to the abstract state s_j (recall that s_j contains $[\![c_j]\!](\sigma)$, and so the selection corresponds to taking the transition $\langle \sigma, a\rangle \xrightarrow{p_j, j} [\![c_j]\!](\sigma)$ in $\mathcal{M}_P$).

If Player 1 proposes $\circledcirc$ by choosing $\langle n, \circledcirc\rangle$, then Player 2 can accept the proposal, (modeled by a move to $\circledcirc$) or, if not all concrete states of n are final, reject it (modeled by a move $\langle n, \circledcirc\rangle \xrightarrow{\times} \langle n, \circledcirc\rangle$).



s_0
$1, [0, 0]$
l1a
$\langle 1, [0, 0]\rangle$, l1a
d_1
$0.1, 2$
$2, [0, 0]$
l2b
$\langle 2, [0, 0]\rangle$, l2b
d_3
$1, 1$
l2c
$\langle 2, [0, 0]\rangle$, l2c
d_4
$\langle 3, [0, 0]\rangle$
$\circ$
$\langle 3, [0, 0]\rangle, \circledcirc$
$\circledcirc$
$0.9, 1$
l1b
$\langle 1, [0, \infty)\rangle$, l1b
$\otimes$
$\times$
d_5
$4, [100, \infty)$
l4
$\langle 4, [100, \infty)\rangle$, l4
d_6
$1, [0, \infty)$
$\langle 1, [0, \infty)\rangle$, l1a
d_2
$0.1, 2$
$2, [0, \infty)]$
l2a
$\langle 2, [0, \infty)]\rangle$, l2a
d_8
l2c
$\langle 2, [0, \infty)]\rangle$, l2c
d_7
l2b
$\langle 2, [0, \infty)]\rangle$, l2b
d_9
$4, [1, \infty)]$
$\langle 4, [1, \infty)]\rangle$, l4
d_{10}

Figure 4.8: Complete abstraction without refinement (the reject state $\otimes$ is duplicated to keep the presentation clear; also we skipped labels from transitions between states of Player 2 to probabilistic states).

4.4.2 Constructing an Example Arena

We illustrate how to build abstract arenas using our example program from Fig. 4.2. Our algorithm performs an abstract exploration of the reachable state space of $\mathcal{M}_P$, similar to the approaches of Beyer et al. [13], Gulavani et al. [49], where abstract reachability trees are constructed for investigating the reachable state space of nonprobabilistic programs. However, to obtain bounds for extremal reachability values we also have to approximate the *structure* of the transition system of $\mathcal{M}_P$, by building game arenas that in particular might contain loops and have to obey additional requirements.

Before the construction we have to select an abstract domain that represents abstract states. We choose the domain

$$\mathrm{PD} = L \times \mathbb{INT}_{\texttt{nrp}} = \mathbb{PRED}(loc = 1, loc = 2, loc = 3, loc = 4) \times \mathbb{INT}_{\texttt{nrp}}$$

from Section 4.2.

For the sake of simplicity we write a concrete state of the example program as a pair $\langle \ell, x\rangle$, where ℓ is the value of loc and x is the value of nrp. An abstract state (i.e. an element of PD) is written as a pair $\langle v, [a, b]\rangle$: $[a, b]$ is an interval abstracting the values of `nrp` and either $v = i$ for an $i \in \{1, \ldots, 4\}$ (representing the abstract state not equal to $\top$ in which $\phi_i = (loc = i)$ holds) or $v \in \{\bot, \top\}$.

Our construction algorithm also requires us to provide a widening. We choose the widening ∇ for $D^\sharp$ that we defined in Section 4.2.

During the symbolic exploration of the state space, the widening is applied at special situations: after computing a successor state s, an abstract state s' already present in the arena is selected, and a widening is performed using s and s', resulting in $\hat{s}$ with $s' \sqsubseteq \hat{s}$; we then use $\hat{s}$ instead of s' in the construction. By a suitable widening strategy we can avoid (infinitely) long sequences of generated abstract states, since every sequence eventually becomes stationary. We apply the widening operator ∇ (in this example) as follows: if the abstract state $s = \langle b, [a, b]\rangle$ has an ancestor $s' = \langle b, [a', b']\rangle$ along the path between it and the initial state given by the construction, then we over-approximate s by $\hat{s}$ with $\hat{s} = s' \,\nabla\, (s \sqcup s')$. We will see a general example of a widening strategy for arbitrary domains in Section 4.6.

We illustrate the algorithm by showing the first steps of the construction (see Fig. 4.7):

1. The algorithm first creates the abstract state $s_0 = \alpha(\langle 1, 0\rangle) = \langle 1, [0, 0]\rangle$ and adds it to the game arena as a Player 1 state. It checks which guarded commands are *enabled* by some states in $\gamma(s_0)$[1], in this case only `l1a`. It adds a Player 2 state $\langle s_0, \texttt{l1a}\rangle$ to the arena, and a transition $s_0 \xrightarrow{\texttt{l1a}} \langle s_0, \texttt{l1a}\rangle$.

2. The algorithm adds a probabilistic state d_1 with $\langle s_0, \texttt{l1a}\rangle \xrightarrow{d_1} d_1$ to the arena (circle-shaped). Note that in general there might be more than one probabilistic state as a successor of a Player 2 state, corresponding to Player 2 "choosing" a concrete state. Two successors are created, representing the semantic effects of the two probabilistic updates of `l1a` on $\langle 1, [0, 0]\rangle$. Both are processed by the algorithm. We only explain how the algorithm handles s_2, the state according to the first update here: it applies the widening as explained above, and adds $s_2' = s_0 \nabla (s_0 \sqcup s_2) = \langle 1, [0, \infty)\rangle$ to the arena (not s_2 itself).

3. In $s_2' = \langle 1, [0, \infty)\rangle$ *two* guarded commands, namely `l1a` and `l1b`, are enabled (by different concrete states). The algorithm adds a Player 2 state $s_3 = \langle\langle 1, [0, \infty)\rangle, \texttt{l1a}\rangle$ (as well as $\langle\langle 1, [0, \infty)\rangle, \texttt{l1b}\rangle$) to the arena, and transitions from s_3 to the reject state $\otimes$, since not every state in $\gamma(s_2')$ enables the guard of `l1b`. A probabilistic state d_2 is added, and again the algorithm creates a successor for each probabilistic update. For the first update, the state $\langle 1, [1, \infty)\rangle$ is computed. The algorithm performs a widening and computes $\langle 1, [0, \infty)\rangle = s_2' \nabla (\langle 1, [1, \infty)\rangle \sqcup s_2')$. Since $\langle 1, [0, \infty)\rangle = s_2'$ it adds a transition $s_4 \xrightarrow{0.9,1} s_2'$.

After processing the other constructed states, we obtain the complete game arena, which is shown in Fig. 4.8. Note that states s, s' belonging to Player

[1]This check can always be performed conservatively: it might report guarded commands enabled for which actually no concrete state in $\gamma(s_0)$ is enabled. See Section 4.6 for details.

1 may "overlap", i.e. $\gamma(s) \cap \gamma(s') = \emptyset$ (the disjointness property) is not guaranteed in general. In Fig. 4.8 the states $\langle 2, [0,0] \rangle$ and $\langle 2, [0,\infty) \rangle$ do overlap, since $\gamma(\langle 2, [0,0] \rangle) \cap \gamma(\langle 2, [0,\infty) \rangle) = \{\sigma\}$ with $\sigma(loc) = 2$ and $\sigma(nrp) = 0$.
Furthermore if we did not apply any widenings during the construction, the abstract arena would be as large as the concrete MDP (at least in this example): Player 2 would never had a proper choice between two alternatives.

4.4.3 Reachability Information from Arenas

After building the arena $\mathcal{G}$, we compute lower and upper bounds for the minimal and maximal reachability values of $\mathcal{M}_P$. Let F be the set of final states in $\mathcal{M}_P$. We obtain bounds $\max^+, \max^-$ with $\max^- \leq \mathit{MaxReach}(\mathcal{M}_P, F) \leq \max^+$ and $\min^+, \min^-$ with $\min^- \leq \mathit{MinReach}(\mathcal{M}_P, F) \leq \min^+$ by considering four games that use $\mathcal{G}$ as arena, but have different winning objectives for both players.

4.4.3.1 Bounds $\max^+, \max^-$ for $\mathit{MaxReach}(\mathcal{M}_P, F)$

Here the winning objective of Player 1 is to reach $\circledcirc$. The bounds are given by the supremum over the probability of Player 1 winning, hereby taking two different objectives for Player 2 into account:

$\max^+$: Here Player 1 and Player 2 *cooperate*, i.e. Player 2's winning objective is also to reach $\circledcirc$. $\max^+$ is then the maximum over the probability of reaching $\circledcirc$, taken over all strategies of both Player 1 and Player 2. This value equals the maximal probability of reaching $\circledcirc$ in the MDP

$$\mathcal{M} = (Q_1 \cup Q_2, Q_P, \{s_0\}, \rightarrow, \mathrm{Lab}_A, \mathrm{Lab}_P)$$

(we merge the player states of $\mathcal{G}$ together), and we can write it as

$$\max^+ = \sup_{S_1 \in \mathcal{S}_1(\mathcal{G})} \sup_{S_2 \in \mathcal{S}_2(\mathcal{G})} \Pr^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, \{\circledcirc\})].$$

$\square$

max$^-$: Here Player 2's objective is to *avoid* ◎, i.e. the players are adversaries: for every strategy S_1 of Player 1, Player 2 plays optimally if Player 1's probability of winning is minimized. Let us denote this minimal probability that Player 2 can achieve dependent on S_1 by $r^-(S_1)$. max$^-$ can then be chosen as the supremum of $r^-(S_1)$ taken over all strategies S_1 of Player 1:

$$\max^- = \sup_{S_1 \in \mathcal{S}_1(\mathcal{G})} \inf_{S_2 \in \mathcal{S}_2(\mathcal{G})} \Pr^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, \{◎\})].$$

□

For the intuition behind these objectives, consider first the game described for obtaining max$^-$. Since Player 1 models the environment and wins by reaching ◎, the environment's goal is to reach a final state. Imagine first that the we use the concrete domain (and best abstract transitions), i.e., abstract and concrete states coincide. In our example this corresponds to not using widenings at all. Then Player 2 never has a choice, and the optimal strategy for Player 1 determines a set S of action sequences whose total probability is equal to the maximal probability of reaching a final state. Imagine now that the abstraction is coarser, i.e. we use an abstract domain or abstract transformers such that some abstract states in the arena represent more than one concrete state. In the arena for the abstract game the sequences of S are still possible, but now Player 2 may be able to prevent them, for instance by moving to ⊗ when an abstract state contains concrete states not enabling the next action in the sequence. Therefore, in the abstract game the probability that Player 1 wins can be at most equal to the maximal probability. In the case of max$^+$ the team formed by the two players can exploit the spurious paths introduced by the use of an abstract domain to find a strategy leading to a better set of paths; in any case, the probability of S is a lower bound for the winning probability of the team.

4.4.3.2 Bounds min$^-$, min$^+$ for *MinReach*$(\mathcal{M}_P, F)$

Here the objective of Player 1 is to *avoid* both ◎ and ⊗. The bounds are given by *minima* over the probability of Player 1 *losing*, again dependent on Player 2's objectives:

min$^-$: Player 2's objective is also to avoid ◎ and ⊗, i.e. the players cooperate. $\min^-$ is then given by the minimal probability of Player 1 *not* winning (i.e. of the game reaching ◎ or ⊗), with the minimum taken over all strategies of both Player 1 and Player 2. Dually to $\max^+$ it equals the minimal probability of reaching ◎ or ⊗ in the MDP $\mathcal{M}$ and can be written as

$$\min^- = \inf_{S_1 \in \mathcal{S}_1(\mathcal{G})} \inf_{S_2 \in \mathcal{S}_2(\mathcal{G})} \Pr^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, \{◎, \otimes\})].$$

□

min$^+$: The players are again adversaries: for every strategy S_1 of Player 1, Player 2 plays optimally if Player 1's winning probability is minimized. Let us denote this minimal probability dependent on S_1 by $s^-(S_1)$. By taking the maximum m of $s^-(S_1)$ over all strategies S_1 of Player 1 we can give $\min^+ = 1 - m$. Note that this again corresponds to the minimal probability of Player 1 *not* winning in the case where the players play against each other. Note also the duality: choosing a strategy that minimizes (maximizes) the probability of reaching a state set is equivalent to maximizing (minimizing) the probability of avoiding it. With this we can write

$$\begin{aligned} m &= \sup_{S_1 \in \mathcal{S}_1(\mathcal{G})} \inf_{S_2 \in \mathcal{S}_2(\mathcal{G})} \Pr^{(S_1,S_2)}[\mathrm{Runs}^{\mathcal{G}}(s_0) \setminus \mathrm{Reach}(\mathcal{G}, \{◎, \otimes\})] \\ &= 1 - \inf_{S_1 \in \mathcal{S}_1(\mathcal{G})} \sup_{S_2 \in \mathcal{S}_2(\mathcal{G})} \Pr^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, \{◎, \otimes\})] \end{aligned}$$

and so

$$\min^- = \inf_{S_1 \in \mathcal{S}_1(\mathcal{G})} \sup_{S_2 \in \mathcal{S}_2(\mathcal{G})} \Pr^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, \{◎, \otimes\})].$$

□

The intuition behind these game settings is similar to the one for $\max^-$ and $\max^+$. Note the somehow reversed role of ⊗ here, for example in the game setting for $\min^+$: if Player 2 is moving to ⊗ here, Player 1 again loses the play. $\min^-$ can therefore never be less than the minimal probability of reaching a final state, since here we choose $\min^-$ as the minimal probability of Player 1 *losing* the game.

Example strategies: Note that the action labels $\circ$ corresponds to choosing $\circledcirc$ by both players; $\times$ corresponds to Player 2 rejecting a move of Player 1. Optimal strategies for the game setting in $\max^+$ for Fig. 4.8 are: for Player 1, always play $\circ$ or `l2c` if possible; for Player 2, play $\circ$ if possible, otherwise avoid $\times$. The value of the game is 1. In the game for computing $\max^-$, the optimal strategy for Player 1 is the same, whereas Player 2 always plays $\otimes$ whenever possible. The value of the game is 0.1. We get $[0.1, 1]$ as lower and upper bound for the maximal probability. For the minimal probability we get the trivial bounds $[0, 1]$. $\square$

To get more precision, we can skip widenings at certain situations during the construction. If we skip widening operations at the successors of the initial state, the resulting abstract arena allows us to obtain the more precise bounds $[0, 0.1]$ and $[0.1, 0.1]$ for the minimal and maximal reachability values, respectively (roughly spoken, the "receive loop“ is unrolled one time if we skip the first widening in this case).

The values that we defined using the four different game settings give rise to the definition of *extremal game values*:

Definition 29. (Extremal game values)
Let $\mathcal{G} = (Q_1, Q_2, Q_P, s_0, \rightarrow, \mathrm{Lab}_A, \mathrm{Lab}_P)$ be a stochastic 2-Player game arena and $Q = Q_1 \cup Q_2 \cup Q_P$. Let $F \subseteq Q$. The *extremal game values*

$$\max{}^+(\mathcal{G}, F), \max{}^-(\mathcal{G}, F), \min{}^+(\mathcal{G}, F) \text{ and } \min{}^-(\mathcal{G}, F)$$

(relative to F) are defined as follows:

$$\begin{aligned}
\max{}^+(\mathcal{G}, F) &:= \sup_{S_1 \in \mathcal{S}_1(\mathcal{G})} \sup_{S_2 \in \mathcal{S}_2(\mathcal{G})} \Pr{}^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, F)] \\
\max{}^-(\mathcal{G}, F) &:= \sup_{S_1 \in \mathcal{S}_1(\mathcal{G})} \inf_{S_2 \in \mathcal{S}_2(\mathcal{G})} \Pr{}^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, F)] \\
\min{}^+(\mathcal{G}, F) &:= \inf_{S_1 \in \mathcal{S}_1(\mathcal{G})} \sup_{S_2 \in \mathcal{S}_2(\mathcal{G})} \Pr{}^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, F)] \\
\min{}^-(\mathcal{G}, F) &:= \inf_{S_1 \in \mathcal{S}_1(\mathcal{G})} \inf_{S_2 \in \mathcal{S}_2(\mathcal{G})} \Pr{}^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, F)].
\end{aligned}$$

$\square$

To make clear that extremal game values are in fact rather easy to compute we first define memoryless and non-randomized strategies:

Definition 30. (Memoryless and non-randomized strategies)
Let $\mathcal{G} = (Q_1, Q_2, Q_P, s_0, \rightarrow, \mathrm{Lab}_A, \mathrm{Lab}_P)$ be a stochastic 2-Player game arena. A strategy S for Player i ($i \in \{1, 2\}$) is called

- *memoryless* if for all paths $\{\pi_1, \pi_2\} \subseteq \mathrm{Paths}^{\mathcal{G}}$ with $last(\pi_1) = last(\pi_2) \in Q_i$ it holds that $S(\pi_1) = S(\pi_2)$, i.e. the strategy choice depends only on the currently visited state of the play,
- *non-randomized* if for every path $\pi \in \mathrm{Paths}^{\mathcal{G}}$ with $last(\pi) \in Q_i$ there exists an action $\ell \in \mathrm{Lab}_i$ such that $S(\pi)(\ell) = 1$.

$\square$

A strategy that is both memoryless and non-randomized can be represented by assigning a single action label to each state of the player. Luckily, in a game arena with finitely many states, it is sufficient to consider only such simple strategies for obtaining extremal reachability values. This is justified by the following well-known theorem (see Condon [21], Condon [20], and Chatterjee et al. [18]):

Theorem 4. *(Optimal strategies for 2-player games)*[1]
Let $\mathcal{G} = (Q_1, Q_2, Q_P, s_0, \rightarrow, Lab_A, Lab_P)$ be a stochastic 2-Player game arena. Let $F \subseteq Q_1 \cup Q_2 \cup Q_P$. For each $\kappa \in \{+, -\}$ there exist non-randomized and memoryless *strategies S_1^κ, T_1^κ in $\mathcal{S}_1(\mathcal{G})$ and S_2^κ, T_2^κ in $\mathcal{S}_2(\mathcal{G})$ such that*

$$
\begin{aligned}
&max^\kappa(\mathcal{G}, F) = Pr^{(S_1^\kappa, S_2^\kappa)}[Reach(\mathcal{G}, F)] \text{ and}\\
&min^\kappa(\mathcal{G}, F) = Pr^{(T_1^\kappa, T_2^\kappa)}[Reach(\mathcal{G}, F)].
\end{aligned}
$$

$\square$

Using this theorem, extremal game values can be computed e.g. by variants of value iteration, similarly to MDPs (see e.g. Condon [20], Puterman [89]). The main theoretical result of this chapter is the counterpart of the results of Hahn et al. [52], Kattenbelt et al. [63]: for arbitrary abstract domains, the values of the four games described above indeed yield upper and lower bounds of the maximal and minimal probability of reaching the final states. We prove this in Theorem 5.

[1] An analogous result holds for (finite-state) MDPs.

4.4.4 Infinite Domains, Widenings, and Predicate Domains

If we use a predicate domain (and best abstract transitions) in the algorithm sketched in the previous paragraph, we obtain abstract game arenas that are essentially equivalent to the ones in Wachter [98] and Hahn et al. [52] (see Section 4.6.2). In order to give a first impression of the advantages of *general abstract domains* beyond predicate abstraction in the probabilistic case, consider the program given in form of pseudo code on the left of Fig. 4.9, a variant of the program from before. Here `coin`(p) models a call to a random number generator that returns 1 with probability p and 0 with probability $1 - p$.[1] It is easy to see that $c \leq 1$ is a global invariant, and so the probability of failure is exactly 0.5. Hence a simple invariant like $c \leq k$ for a $k \leq 100$, together with the postcondition $i > 100$ of the loop would be sufficient to negate the guard of the statement at line 5. PASS, a leading tool for probabilistic reachability problems using game-based abstraction with *predicate abstraction* and abstraction refinement uses a CEGAR-like approach (see Clarke et al. [19], Hahn et al. [52]): if a constructed arena gives too coarse lower and upper bounds for the extremal reachability values, PASS inspects the arena and extracts new, hopefully useful predicates, and uses them to refine (i.e. extend) its predicate domain and rebuilds the arena. PASS offers different options for generating such predicates. However, when the example program is analyzed with PASS, the while loop is unrolled 100 times because the tool fails to "catch" the invariant, independently of the options chosen to refine the abstraction.[2]

On the other hand, an analysis of the program with the standard interval domain for abstracting the integer variables, the standard widening operator, and the standard technique of delaying widenings (Blanchet et al. [15]), easily "catches" the invariant (see Section 4.8). The same happens for the (infinite-state) program on the right of the figure, which exhibits a more interesting probabilistic behaviour, especially a probabilistic choice within a loop: we obtain good upper and lower bounds for the probability of failure using the standard interval

[1]We give this and the following program in pseudo code for brevity; for our experiments we transformed the program into a PGP, similarly to the example above.

[2]Actually, the input language of PASS does not explicitly include while loops, they have to be simulated. But this does not affect the analysis.

```
  int c = 0, i = 0;
1: if (coin(0.5)) {
2:   while (i <= 100) {
3:     i++;
4:     c = c-i+2
   } }
5: if (c >= i) fail
end.
```

```
  int c = 0, i = 0;
1: while(i <= 100) {
2:   if (coin(0.5)) i++
3:   c = c-i+2; }
4: if (c >= i) fail
end.
```

Figure 4.9: Example programs 2 and 3.

domain. Notice that examples exhibiting the opposite behaviour (predicate abstraction with abstraction refinement succeeds where interval analysis fails) are not difficult to find. Our claim is only that the game-based abstraction approaches of Hahn et al. [52], Kattenbelt et al. [63] can be extended to arbitrary abstract domains, making it more flexible and efficient.

4.5 Formal Definition of Abstract Game Arenas

In this section we give a definition of abstract game arenas of a PGP P. We also prove that every abstract game arena can be used to obtain reliable lower and upper bounds for extremal reachability values of $\mathcal{M}_P$.

Definition 31. Let $P = (\mathcal{V}, \sigma_0, \mathcal{C})$ be a PGP with a final guard f forming an instance of Problem 3. Let $F \subseteq \Sigma_{\mathcal{V}}$ be the set of states that enable f. Let further $D = (D^\sharp, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$ be a domain abstracting $\Sigma_{\mathcal{V}}$. A 2-player game arena $\mathcal{G} = (Q_1, Q_2, Q_P, s_0, \to, \mathcal{C} \cup \{\times, \circ, \tau\} \cup (D^\sharp)^*, \mathbb{N})$ with finitely many states is an *abstract game arena* of P relative to f (for D) if

- Q_1 consists of a subset of $D^\sharp$ plus two distinguished states $\circledcirc, \otimes$;
- $s_0 = \alpha(\{\sigma_0\}) \in Q_1$;
- Q_2 is a set of pairs $\langle s, x\rangle$, where $s \in Q_1 \setminus \{\circledcirc, \otimes\}$ and either $x = \circledcirc$ or x is a guarded command of $\mathcal{C}$;
- Q_P is a set of tuples $\langle s, a, d\rangle$, where $s \in Q_1 \setminus \{\circledcirc, \otimes\}$, $a \in \mathcal{C}$, and $d \in (Q_1 \setminus \{\circledcirc, \otimes\})^{|up_a|}$, i.e. d is a sequence $\langle d_1, \ldots, d_{|up_a|}\rangle$ of $|up_a|$ abstract states.

For every $i \in \{1, \dots, |up_a|\}$, $\langle s, a, d\rangle \xrightarrow{p_a(i),i} d_i$.

- $\otimes \xrightarrow{\times} \otimes$ respectively $\circledcirc \xrightarrow{\circ} \circledcirc$ are the only outgoing transitions of $\otimes$ respectively $\circledcirc$,

and the following conditions hold:

(a) For every $s \in Q_1 \setminus \{\circledcirc, \otimes\}$: if $\gamma(s) \cap F \neq \emptyset$ then $s \xrightarrow{\circ} \langle s, \circledcirc\rangle \xrightarrow{\circ} \circledcirc$. If moreover $\gamma(s) \not\subseteq F$, then $\langle s, \circledcirc\rangle \xrightarrow{\times} \langle s, \circledcirc\rangle$.
(If $\gamma(s)$ contains some final state, then Player 1 has the possibility to propose $\circledcirc$. If not all states are final, then Player 2 can reject the proposal by staying in $\langle s, \circledcirc\rangle$.)

(b) For every $s \in Q_1 \setminus \{\circledcirc, \otimes\}$ with $a \in \mathcal{C}$: if $g_a(\gamma(s)) \neq \emptyset$, then $\langle s, a\rangle \in Q_2$ and $s \xrightarrow{a} \langle s, a\rangle$. *(If a concrete state of s enables g_a, Player 1 can choose a.)*

(c) For every $\langle s, a\rangle \in Q_2$ with $a \in \mathcal{C}$ and $up_a = \langle\langle p_1, c_1\rangle, \dots, \langle p_k, c_k\rangle\rangle$: if $\langle s, a\rangle \in Q_2$ and $g_a(\gamma(s)) \neq \gamma(s)$, then $\langle s, a\rangle \xrightarrow{\times} \otimes$. Further, for every $\sigma \in g_a(\gamma(s))$ there exists a sequence d_σ with k elements such that $\langle s, a\rangle \xrightarrow{d_\sigma} \langle s, a, d_\sigma\rangle \in Q_P$, and for s_i with $\langle s, a, d_\sigma\rangle \xrightarrow{p_i,i} s_i$, it holds that $[\![c_i]\!](\sigma) \in \gamma(s_i)$ for all $i \in \{1, \dots, k\}$.
(Player 2 can end the game if at least one concrete state in $\gamma(s)$ does not enable g_a. If Player 2 accepts the choice of Player 1, she can pick a $\sigma \in g_a(\gamma(s))$ by choosing the corresponding probabilistic state; this state then performs an "abstract" update operation.)

(d) If $\gamma(s) \cap F = \emptyset$ and for all $a \in \mathcal{C}$ it holds that $\gamma(g_a(s)) = \emptyset$, then $s \xrightarrow{\tau} s$.
(If s has not outgoing transitions we add a self-loop.)

Further every outgoing transition of $s \in Q_1 \setminus \{\circledcirc, \otimes\}$ has either the form $s \xrightarrow{a} \langle s, a\rangle$ for $a \in \mathcal{C}$ or $s \xrightarrow{\circ} \langle s, \circledcirc\rangle$; every outgoing transition of $\langle s, a\rangle \in Q_2$ with $a \in \mathcal{C}$ has the form $\langle s, a\rangle \xrightarrow{d} \langle s, a, d\rangle \in Q_P$ for a sequence d with $|up_a|$ elements, or $\langle s, a\rangle \xrightarrow{\times} \otimes$. □

We do not require an abstract game arena to be the *smallest* game arena satisfying the conditions above. The arena might contain additional transitions and states,

caused e.g. by using conservative tests for the premises of the conditions (a)-(d) during its construction (see also Section 4.6.1).
Abstract game arenas allow Player 2 to choose the action $\times$ every time if there is uncertainty due to abstraction whether a guard can be taken by *all* concrete states of a state or not. Note the action label $\circ$ corresponds to choosing $\circledcirc$ for both players; $\times$ corresponds to Player 2 rejecting a move of Player 1.

Structure of runs in $\mathcal{G}$. We describe in more detail runs possible in an abstract game arena $\mathcal{G}$, using the notations from Def. 31. A *game round* has the form

$$s \xrightarrow{a} \langle s, a\rangle \xrightarrow{d} \langle s, a, d\rangle \xrightarrow{i} s'$$

for s and s' states of Player 1, a a guarded command, and d a sequence of abstract states, $i \in \{1, \ldots, |up_a|\}$.
Each run $r \in \mathrm{Runs}^{\mathcal{G}}$ belongs to one of the following classes (r' denotes a sequence of game rounds, s a state in Q_1, $a \in \mathcal{C}$):

(R_1) it is an infinite sequence of game rounds;

(R_2) it ends in $\circledcirc$: $r = r' \xrightarrow{\circ} \langle s, \circledcirc\rangle \xrightarrow{\circ} \circledcirc \xrightarrow{\circ} \ldots$;

(R_3) it ends in a state $Q_1 \times \{\circledcirc\}$: $r = r' \xrightarrow{\circ} \langle s, \circledcirc\rangle \xrightarrow{\times} \langle s, \circledcirc\rangle \xrightarrow{\times} \langle s, \circledcirc\rangle \xrightarrow{\times} \ldots$;

(R_4) it ends in $\otimes$: $r = r' \to \langle s, a\rangle \to \otimes \xrightarrow{\times} \otimes \xrightarrow{\times} \ldots$;

(R_5) it gets stuck in s: $r = r' \to s \xrightarrow{\tau} s \xrightarrow{\tau} s \xrightarrow{\tau} \ldots$.

4.5.1 Obtaining Reachability Bounds

We can now state our main theorem: for a PGP P, the extremal game values from abstract game arenas of P provide upper and lower bounds of the extremal reachability values of $\mathcal{M}_P$.

Theorem 5. *(Extremal reachability values and abstract game arenas)*
Let P be a PGP and f be a guard forming an instance of Problem 3. Let $\mathcal{G}$ be

an abstract game arena of P relative to f for an abstract domain D. Then the following three propositions hold.

$$\begin{aligned}&\mathit{MinReach}(\mathcal{M}_P, F) \in [\mathit{min}^-(\mathcal{G}, \{\circledcirc, \otimes\}), \mathit{min}^+(\mathcal{G}, \{\circledcirc, \otimes\})] \text{ and}\\ &\mathit{MaxReach}(\mathcal{M}_P, F) \in [\mathit{max}^-(\mathcal{G}, \{\circledcirc\}), \mathit{max}^+(\mathcal{G}, \{\circledcirc\})].\end{aligned}$$

$\square$

The proof of the theorem is an immediate consequence of the following lemma:

Lemma 3. *(Strategies for extremal game values)*
Let P be a PGP and f be a guard forming an instance of Problem 3. Let $\mathcal{G}$ be an abstract game arena of P relative to f for an abstract domain D. Then

(1) Given a strategy S of the (single) player in $\mathcal{M}_P$, there exists a strategy $S_1 \in \mathcal{S}_1(\mathcal{G})$ such that

$$\begin{aligned}&\inf_{T \in \mathcal{S}_2(\mathcal{G})} Pr^{(S_1,T)}[\mathit{Reach}(\mathcal{G}, \{\circledcirc, \otimes\})]\\ &\leq Pr^S[\mathit{Reach}(\mathcal{M}_P, F)]\\ &\leq \sup_{T \in S_2(\mathcal{G})} Pr^{(S_1,T)}[\mathit{Reach}(\mathcal{G}, \{\circledcirc\})].\end{aligned}$$

(2) Given a strategy $S_1 \in \mathcal{S}_1(\mathcal{G})$ there exists a strategy $S \in \mathcal{S}(\mathcal{M}_P)$ such that

$$Pr^S[\mathit{Reach}(\mathcal{M}_P, F)] \leq \sup_{T \in \mathcal{S}_2(\mathcal{G})} Pr^{(S_1,T)}[\mathit{Reach}(\mathcal{G}, \{\otimes, \circledcirc\})].$$

(3) Given a strategy $S_1 \in \mathcal{S}_1(\mathcal{G})$ there exists a strategy $S \in \mathcal{S}(\mathcal{M}_P)$ such that

$$Pr^S[\mathit{Reach}(\mathcal{M}_P, F)] \geq \inf_{T \in S_2(\mathcal{G})} Pr^{(S_1,T)}[\mathit{Reach}(\mathcal{G}, \{\circledcirc\})].$$

We first give the proof of Theorem 5, assuming the correctness of Lemma 3, and prove Lemma 3 afterwards:

Proof (of Theorem 5).
Let $S \in \mathcal{S}(\mathcal{M}_P)$ be an arbitrary strategy for $\mathcal{M}_P$. By Lemma 3 (1) there exists a

strategy $S_1 \in \mathcal{S}_1(\mathcal{G})$ satisfying

$$\begin{aligned}&\inf_{T\in\mathcal{S}_2(\mathcal{G})} \Pr^{(S_1,T)}[\mathrm{Reach}(\mathcal{G},\{\circledcirc,\otimes\})]\\ &\leq \Pr^S[\mathrm{Reach}(\mathcal{M}_P,F)]\\ &\leq \sup_{T\in S_2(\mathcal{G})} \Pr^{(S_1,T)}[\mathrm{Reach}(\mathcal{G},\{\circledcirc\})].\end{aligned}$$

From this we conclude that for all $S \in \mathcal{S}(\mathcal{M}_P)$

$$\min{}^-(\mathcal{G},\{\circledcirc,\otimes\}) \leq \inf_{T\in\mathcal{S}_2(\mathcal{G})} \Pr^{(S_1,T)}[\mathrm{Reach}(\mathcal{G},\{\circledcirc,\otimes\})] \leq \Pr^S[\mathrm{Reach}(\mathcal{M}_P,F)],$$

i.e. $\min^-(\mathcal{G},\{\circledcirc,\otimes\})$ is a lower bound for $\Pr^S[\mathrm{Reach}(\mathcal{M}_P,F)]$, and so

$$\min{}^-(\mathcal{G},\{\circledcirc,\otimes\}) \leq \inf_{S\in\mathcal{S}(\mathcal{M}_P)} \Pr^S[\mathrm{Reach}(\mathcal{M}_P,F)] = \mathit{MinReach}(\mathcal{M}_P,F).$$

The inequality $\mathit{MaxReach}(\mathcal{M}_P,F) \leq \max^+(\mathcal{G},\{\circledcirc\})$ can be proved in the same way by using the left inequality of Lemma 3 (1). The remaining inequations are proved similarly using Lemma 3 (2) and Lemma 3 (3), respectively. □

We now give the proof of Lemma 3:

Proof (of Lemma 3).
Let

$$\mathcal{G} = (Q_1, Q_2, Q_P, s_0, \rightarrow, \mathcal{C} \cup \{\circ,\times,\tau\} \cup (D^\sharp)^*, \mathbb{N})$$

and

$$\mathcal{M}_P = (V_A, V_P, \{\sigma_0\}, \Rightarrow, \mathcal{C} \cup \{\tau\}, \mathbb{N}).$$

We set $Q = Q_1 \cup Q_2 \cup Q_P$ and $V = V_A \cup V_P$, and let Lab be the set of action labels and probabilistic labels of $\mathcal{G}$.
We will use some additional notation. For a guard g and a state σ we write $\sigma \models g$ iff σ enables g. We denote paths in $\mathcal{G}$ by $\Pi, \Pi', \hat{\Pi}, \ldots$, paths in $\mathcal{M}_P$ (or in $\mathcal{M}$, see below) by $\pi, \pi', \hat{\pi}, \ldots$. Sequences in $(D^\sharp)^*$ are denoted by $d, d', \hat{d}, \ldots$. We write $d(i)$ for the i-th element of a sequence $d \in (D^\sharp)^*$.
For the following proofs we define an auxiliary function β. Due to the requirements of an abstract game arena, we know that for every $\langle s,a\rangle \in Q_2$ and

$\sigma \in g_a(\gamma(s))$ there exists (at least) one sequence $d \in (D^\sharp)^*$ such that

- $\langle s, a, d\rangle \in Q_P$, $\langle s, a\rangle \xrightarrow{d} \langle s, a, d\rangle$, and
- for every $1 \leq i \leq k$ there exists a s_i with $\langle s, a, d\rangle \xrightarrow{p_i, i} s_i$ and $[\![c]\!](\sigma) \in \gamma(s_i)$,

see Def. 31 (c). But since there might be overlapping abstract states it is possible that there is more than one sequence d which satisfies these conditions. We therefore fix for every $\langle s, a\rangle$ and every $\sigma \in \gamma(g_a(s))$ an arbitrary sequence $d \in (D^\sharp)^*$ that satisfies the conditions above with respect to s, a and σ, and set $\beta(s, a, \sigma) = d$.

Proof of (1).

We restate the claim: Given a strategy S of the (single) player in $\mathcal{M}_P$, there exists a strategy $S_1 \in \mathcal{S}_1(\mathcal{G})$ such that

$$\begin{aligned}
&\inf_{T \in \mathcal{S}_2(\mathcal{G})} \mathrm{Pr}^{(S_1,T)}[\mathrm{Reach}(\mathcal{G}, \{\circledcirc, \otimes\})] \\
&\leq \mathrm{Pr}^{S}[\mathrm{Reach}(\mathcal{M}_P, F)] \\
&\leq \sup_{T \in S_2(\mathcal{G})} \mathrm{Pr}^{(S_1,T)}[\mathrm{Reach}(\mathcal{G}, \{\circledcirc\})].
\end{aligned}$$

We modify $\mathcal{M}_P$ as follows: we add the fresh action label $\circ$ to $\mathcal{M}_P$, a fresh state $\langle \sigma_f, \circledcirc\rangle$ to V_P, and a fresh state σ_f to V_A. The state $\langle \sigma_f, \circledcirc\rangle$ has only one outgoing transition $(\langle \sigma_f, \circledcirc\rangle, 1, \circ, \sigma_f)$ as well as σ_f has only one outgoing transition $(\sigma_f, \circ, \sigma_f)$; we add both transitions to $\Rightarrow$. We also add edges $(q, \circ, \langle \sigma_f, \circledcirc\rangle)$ to $\Rightarrow$ for *every* $q \in F$ and remove all other outgoing transitions of states in F. We denote this modified MDP by $\mathcal{M}$. So

$$q \overset{\circ}{\Rightarrow} \langle \sigma_f, \circledcirc\rangle \overset{1,\circ}{\Longrightarrow} \sigma_f \overset{\circ}{\Rightarrow} \sigma_f \overset{\circ}{\Rightarrow} \ldots$$

is the only possible path starting from a $q \in F$ in $\mathcal{M}$. This simple modification facilitates our mapping between paths in $\mathcal{G}$ and paths in $\mathcal{M}$.

Let now $S_{\mathcal{M}_P} \in \mathcal{S}(\mathcal{M}_P)$. $S_{\mathcal{M}_P}$ corresponds uniquely to a strategy $S \in \mathcal{S}(\mathcal{M})$, where S chooses σ_f in every state $\sigma \in F$ (which is the only option for S) and otherwise behaves exactly as $S_{\mathcal{M}_P}$. Note that we have $\mathrm{Pr}^{\mathcal{M},S}[\mathrm{Reach}(\mathcal{M}, \{\sigma_f\})] = \mathrm{Pr}^{\mathcal{M}_P,S'}[\mathrm{Reach}(\mathcal{M}_P, F)]$.

We show that there exist strategies $S_1 \in \mathcal{S}_1(\mathcal{G})$ and $S_2 \in \mathcal{S}_2(\mathcal{G})$ such that

$$\Pr^S[\text{Reach}(\mathcal{M}, \{\sigma_f\})] = \Pr^{(S_1,S_2)}[\text{Reach}(\mathcal{G}, \{\circledcirc\})],$$

which implies $\Pr^{S_{\mathcal{M}_P}}[\text{Reach}(\mathcal{M}, \{\sigma_f\})] = \Pr^{(S_1,S_2)}[\text{Reach}(\mathcal{G}, \{\circledcirc\})]$. Designing such strategies bears some similarity to the proof of Prop. 3 in Kattenbelt et al. [62]. However, we can not assume to have a partition of the state space in general, and also the structure of our game arenas is different e.g. due to the use of $\otimes$ and the fact that abstract states may contain final states as well as non-final states. The same reasons and the fact that we use two-player arenas also prevent us from using results from Segala and Lynch [93] for probabilistic simulations of MDPs in an obvious way (this will become more apparent in the proofs of part (2) and (3)). Therefore we build an appropriate strategy from scratch.

Definition of S_1 and S_2. We define S_1 and S_2 inductively: given a (S_1, S_2)-possible path Π, we define S_1 or S_2 for Π, and obtain possible "extensions" of Π, i.e. (S_1, S_2)-possible paths with Π as a prefix. During the definition of S_1, S_2 we implicitly construct $\mathcal{G}[S_1, S_2]$'s transition system.
We also define inductively a function

$$T : \text{Paths}^{\mathcal{G},(S_1,S_2)} \cap (Q\text{Lab})^*(Q_1 \cup Q_2) \to \text{Paths}^{\mathcal{M},S}$$

during this construction process. T's domain is the set of (S_1, S_2)-possible paths in $\mathcal{G}$ *ending in a player state*; T's range is the set of S-possible paths in $\mathcal{M}$. T is used to establish a one-to-one correspondence between paths in $\mathcal{G}$ and paths in $\mathcal{M}$. During the construction of the strategies and the definition of T, the following property (*) can be immediately shown for all newly defined values of T in each step of the construction:

Property (*) for a (S_1, S_2)-possible path Π with $T(\Pi) = \pi$:
π is an S-possible path. Further the last states states of Π and π are related as follows:

1. Π ends in a state $s \in Q_1 \setminus \{\circledcirc\}$ iff π ends in an action state $\sigma \neq \sigma_f$. Then $\sigma \in \gamma(s)$ holds.

2. Π ends in a state $\langle s, a\rangle$ with $a \in \mathcal{C}$ iff π ends in a state $\langle \sigma, a\rangle$; then $\sigma \in \gamma(s)$.

3. Π ends in a state $\langle s, ⊚\rangle$ iff π ends in $\langle \sigma_f, \circ\rangle$. Π ends in ⊚ iff π ends in σ_f.

4. The number of occurrences of Player 1 states in Π and the number of occurences of action states are equal. Also the number of occurrences of ⊚ in Π and the number of occurences of σ_f in π are equal.

□

In the definition of S_1 respectively S_2 for a path Π, we have $T(\Pi)$ at our disposal, and for all Π' that are prefixes of Π we can assume property (*) by induction hypothesis. We start our definition by setting $T(s_0) := \sigma_0$; property (*) is obviously satisfied for $\Pi = s_0$. Property (*) can immediately be verified for all newly defined values of T.

Case 1: $\Pi = s_0$ **or** $\Pi = \hat{\Pi} \to s, s \in Q_1 \setminus \{⊚\}$**.**
Let $L = \{a \in \mathcal{C} \mid S_1(\Pi)(a) > 0\}$.
$T(\Pi)$ ends in a state σ and $\sigma \in \gamma(s)$ (using property (*)).

- $\sigma \notin F$:

  (S_1) : For every command a enabled by σ: set $S_1(\Pi)(a) := S(T(\Pi))(a)$.

  (T) : For every command a enabled by σ with $S(T(\Pi))(a) > 0$: set

$$T(\Pi \xrightarrow{a} \langle s, a\rangle) := T(\Pi) \overset{a}{\Rightarrow} \langle \sigma, a\rangle.$$

- $\sigma \in F$: then $\circ$ can be chosen by S_1 due to Def. 31.

  (S_1) : Set $S_1(\Pi)(\circ) := 1$.

  (T) : Set $T(\Pi \xrightarrow{\circ} \langle s, ⊚\rangle) := T(\Pi) \overset{\circ}{\Rightarrow} \langle \sigma_f, \circ\rangle$.

$S_1(\Pi)$ forms a distribution since $1 = \sum_{a\in\mathcal{C}:\sigma\models a} S(T(\Pi))(a) = \sum_{a\in\mathcal{C}:\sigma\models a} S_1(\Pi)(a)$.

Case 2: $\Pi = \hat{\Pi} \xrightarrow{a} \langle s, a\rangle, a \in \mathcal{C}$**.**
Using Property (*), $T(\Pi)$ ends in a state $\langle \sigma, a\rangle$ and $\sigma \in \gamma(s)$. Let $d = \beta(s, a, \sigma)$ and let $up_a = \langle\langle p_1, c_1\rangle, \ldots, \langle p_k, c_k\rangle\rangle$. Possible extensions of Π start with $\Pi \xrightarrow{d} \langle s, a, d\rangle \xrightarrow{i} d(i)$, where $1 \leq i \leq k$.

(S_2) : Set $S_2(\Pi)(d) := 1$.

(T) : For every $1 \leq i \leq k$: set $T(\Pi \xrightarrow{d} \langle s, a, d\rangle \xrightarrow{i} d(i)) := T(\Pi) \overset{i}{\Rightarrow} [\![c_i]\!](\sigma)$.

Case 3: $\Pi = \hat{\Pi} \xrightarrow{\circ} \langle s, \circledcirc\rangle$ respectively $\Pi = \hat{\Pi} \xrightarrow{\circ} \circledcirc$.

$T(\Pi)$ ends with $\langle \sigma_f, \circ\rangle$ respectively σ_f due to property (*).

(S_2) : Set $S_2(\Pi)(\circ)$ respectively $S_1(\Pi)(\circ)$ to 1.

(T) : Set $T(\Pi \xrightarrow{\circ} \circledcirc) := T(\Pi) \overset{\circ}{\Rightarrow} \sigma_f$.

This concludes the construction.
For all other paths we define S_1 and S_2 arbitrarily (the paths are not (S_1, S_2)-possible anyway). Note also that a path Π is an (S_1, S_2)-possible path if $T(\Pi)$ is defined (the construction above is a complete definition of T, since we covered all possible paths). By our choice of S_1 and S_2 we never reach a “stuck” state s where the only possible outgoing transition is $s \xrightarrow{\tau} s$.
By a simple induction we get that cylinders of paths related by T have the same probability, i.e. for all $\Pi \in \text{Paths}^{\mathcal{G},(S_1,S_2)} \cap (Q\text{Lab})^*(Q_1 \cup Q_2)$ with $\pi = T(\Pi)$ it holds that

$$\Pr^{\mathcal{G},(S_1,S_2)}[\text{Cyl}(\Pi, \mathcal{G})] = \Pr^{\mathcal{M},S}[\text{Cyl}(\pi, \mathcal{M})]. \tag{4.3}$$

We give a proof in Lemma 19 in Appendix A.1. By another induction proof we get that T actually is a *bijection* (for a proof see Lemma 20 in Appendix A.1).
We denote by P_f the paths in $\text{Paths}^{\mathcal{M},S}$ with exactly one occurrence of σ_f:

$$P_f := \{\pi \in \text{Paths}^{\mathcal{M},S} \mid \pi = \hat{\pi} \to \sigma_f \wedge \text{last}(\hat{\pi}) \neq \sigma_f\}.$$

Similarly we denote by P_F the paths in $\text{Paths}^{\mathcal{G},(S_1,S_2)}$ with exactly one occurrence of $\circledcirc$:

$$P_F := \{\Pi \in \text{Paths}^{\mathcal{G},(S_1,S_2)} \mid \Pi = \hat{\Pi} \to \circledcirc \wedge \text{last}(\hat{\Pi}) \neq \circledcirc\}.$$

With property (*) and the fact that T is bijective we get that the restriction T on P_F is also bijective, and $T(P_F) = P_f$. Note also that for every run r that ends in σ_f, i.e. for every $r \in \text{Reach}(\mathcal{M}, \{\sigma_f\})$, there is exactly one $\pi \in P_f$ such that

$r \in \mathrm{Cyl}(\pi, \mathcal{M})$ (analogously for runs ending in $\circledcirc$ and cylinder sets of paths in P_F). We can conclude:

$$\begin{aligned}
\Pr^S[\mathrm{Reach}(\mathcal{M}, \{\sigma_f\})] &= \sum_{\pi \in P_f} \Pr^S[\mathrm{Cyl}(\pi, \mathcal{M})] \\
&= \sum_{\pi \in T(P_F)} \Pr^S[\mathrm{Cyl}(\pi, \mathcal{M})] && \text{(note above)} \\
&= \sum_{\Pi \in P_F} \Pr^{(S_1,S_2)}[\mathrm{Cyl}(\Pi, \mathcal{G})] && \text{(T bijective, Eq. 4.3)} \\
&= \Pr^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, \{\circledcirc\})].
\end{aligned}$$

S_2 never chooses $\otimes$, and so we get also

$$\Pr^S[\mathrm{Reach}(\mathcal{M}, \{\sigma_f\})] = \Pr^{(S_1,S_2)}\mathrm{Reach}(\mathcal{G}, \{\circledcirc, \otimes\}).$$

We can transfer this result back to $\mathcal{M}_P$ and get

$$\Pr^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, \{\circledcirc\})] = \Pr^{S_{\mathcal{M}_P}}[\mathrm{Reach}(\mathcal{M}_P, \{\sigma_f\})] = \Pr^{(S_1,S_2)}[\mathrm{Reach}(\mathcal{G}, \{\circledcirc, \otimes\})].$$

Taking the infimum over strategies of Player 2 on the left and the supremum over strategies of Player 2 on the right side proves the claim.

Proof of (2).
We restate the claim: given a strategy $S_1 \in \mathcal{S}_1(\mathcal{G})$, there exists a strategy $S \in \mathcal{S}(\mathcal{M}_P)$ such that

$$\Pr^S[\mathrm{Reach}(\mathcal{M}_P, F)] \leq \sup_{T \in \mathcal{S}_2(\mathcal{G})} \Pr^{(S_1,T)}[\mathrm{Reach}(\mathcal{G}, \{\otimes, \circledcirc\})].$$

At the beginning of the proof of (1) we used a modified version of $\mathcal{M}_P$ called $\mathcal{M}$. We perform similar modifications here: we add the fresh action label $\circ$ to $\mathcal{M}_P$ and also add a new state $\langle \sigma_f, \circledcirc \rangle$ to V_P and a state σ_f to V_A. $\langle \sigma_f, \circledcirc \rangle$ has only one outgoing transition $(\langle \sigma_f, \circledcirc \rangle, 1, \circ, \sigma_f)$; σ_f has only one outgoing transition $(\sigma_f, \circ, \sigma_f)$. We add both transitions to $\Rightarrow$. We also add edges $(q, \circ, \langle \sigma_f, \circledcirc \rangle)$ to $\Rightarrow$ for *every* $q \in V_A \setminus \{\sigma_f\}$; for every $q \in F$, we also remove all other outgoing transitions of states in F. We denote this modified MDP again by $\mathcal{M}$. So

$$q \overset{\circ}{\Rightarrow} \langle \sigma_f, \circledcirc \rangle \overset{1,\circ}{\Longrightarrow} \sigma_f \overset{\circ}{\Rightarrow} \sigma_f \overset{\circ}{\Rightarrow} \dots$$

is possible for *every* $q \in V_A \setminus \{\sigma_f\}$ (and every run reaching a $q \in F$ ends in this way).
We show that there exist strategies $S \in \mathcal{S}(\mathcal{M})$ and $S_2 \in \mathcal{S}_2(\mathcal{G})$ such that

$$\Pr^S[\text{Reach}(\mathcal{M}, \{\sigma_f\})] = \Pr^{\mathcal{G},(S_1,S_2)}[\text{Reach}(\mathcal{G}, \{\circledcirc, \otimes\})],$$

This equality proves (2): observe that for every $S \in \mathcal{S}(\mathcal{M})$ there exists a strategy $S_{\mathcal{M}_P} \in \mathcal{S}(\mathcal{M}_P)$ such that $\Pr^{\mathcal{M}_P,S_{\mathcal{M}_P}}[\text{Reach}(\mathcal{M}_P, F)] \leq \Pr^{\mathcal{M},S}[\text{Reach}(\mathcal{M}, \{\sigma_f\})]$ (simply distribute the probability assigned to choosing $\circ$ by S to other labels arbitrarily if this is not possible in $\mathcal{M}_P$). So for every $S \in \mathcal{S}(\mathcal{M})$ that satisfies the equality we have

$$\begin{aligned}
&\Pr^{\mathcal{M}_P,S_{\mathcal{M}_P}}[\text{Reach}(\mathcal{M}_P, F)] \\
&\leq \Pr^{\mathcal{G},(S_1,S_2)}[\text{Reach}(\mathcal{G}, \{\circledcirc, \otimes\})] \\
&\leq \sup_{S_2 \in \mathcal{S}_2(\mathcal{G})} \Pr^{\mathcal{G},(S_1,S_2)}[\text{Reach}(\mathcal{G}, \{\circledcirc, \otimes\})].
\end{aligned}$$

We again relate paths in $\mathcal{M}$ to paths in $\mathcal{G}$. The idea of our choice of S and S_2 is: whenever S_1 chooses an action that is not possible in a corresponding situation (path) in $\mathcal{M}$, S_2 and S choose to end the play by going to $\otimes$ respectively σ_f (the roles of $\otimes$ and $\circledcirc$ coincide in this part of the proof).

Definition of S and S_2. In this proof we define S and S_2 inductively and define *both* the S-possible paths in $\mathcal{M}$ and the (S_1, S_2)-possible paths in $\mathcal{G}$ in the same construction. Again we define a function T "on the way"; however, this time we have to relate multiple paths in $\mathcal{M}$ to paths in $\mathcal{G}$ and define

$$T : \text{Paths}^{\mathcal{G},(S_1,S_2)} \cap (Q\text{Lab})^*(Q_1 \cup Q_2) \times \text{Paths}^{\mathcal{M},S} \to [0, 1].$$

T distributes probabilities of a (S_1, S_2)-possible path in $\mathcal{G}$ to probabilities of S-possible paths in $\mathcal{M}$ (and vice versa). We define similar as in the proof of part

(1) a property that can easily be shown inductively by a syntactic check of the new definitions in the construction:

Property (*) for an (S_1, S_2)-possible path Π:
For all paths π with $T(\Pi, \pi) > 0$: π is a S-possible path. Further the last states of Π and π are related as follows:

1. Π ends in a state $s \in Q_1 \setminus \{\circledcirc, \otimes\}$ iff π ends in an action state $\sigma \neq \sigma_f$. Then $\sigma \in \gamma(s)$ holds.
2. Π ends in a state $\langle s, a\rangle$ with $a \in \mathcal{C}$ iff either
   - π ends in $\langle \sigma_f, \circ\rangle$, and then $\langle s, a\rangle \xrightarrow{\times} \otimes$ in $\mathcal{G}$, or
   - π ends in a state $\langle \sigma, a\rangle$, and then $\sigma \in \gamma(s)$ holds.
3. If Π ends in a state $\langle s, \circledcirc\rangle$, then π ends in $\langle \sigma_f, \circ\rangle$. Π ends in $\circledcirc$ or in $\otimes$ iff π ends in σ_f.
4. The number of occurrences of Player 1 states in Π and the number of occurences of action states are equal. Also the number of occurrences of either $\circledcirc$ or $\otimes$ in Π and the number of occurences of σ_f in π are equal.

We perform a case distinction on the shape of a (S_1, S_2)-possible path Π, and can assume by induction hypothesis that $T(\Pi', \pi)$ for all Π' prefixes of Π and all π has already been defined. In the case distinction we define, given an (S_1, S_2)-possible path Π having one of the three possible forms, strategy choices for S_2 and for S for all π with $T(\Pi, \pi) > 0$. We now implicitly construct the transition systems of $\mathcal{G}[S_1, S_2]$ as well as of $\mathcal{M}[S]$. If T is not defined by the construction below for Π and π, we set $T(\Pi, \pi) = 0$ by default. We start by setting $T(s_0, \sigma_0) := 1$ (property (*) holds for s_0).

Case 1: $\Pi = s_0$ or $\Pi = \hat{\Pi} \to s, s \in Q_1 \setminus \{\circledcirc, \otimes\}$.
Let $L = \{a \in \mathcal{C} \mid S_1(\Pi)(a) > 0\}$.
We define for every $\pi \in \mathrm{Paths}^{\mathcal{M}}$ with $T(\Pi, \pi) > 0$ the following:
let $\sigma = \mathrm{last}(\pi)$ with $\sigma \in \gamma(s)$ (property (*), Def. 31).

Let $G_\sigma = \{a \in \mathcal{C} \mid \sigma \models a\}$. For every $a \in \mathcal{C}$ holds: $\sigma \models a$ implies $s \xrightarrow{a} \langle s, a\rangle$ (Def. 31).

(S) : For each $a \in G_\sigma$: Set $S(\pi)(a) := S_1(\Pi)(a)$.
Set $S(\pi)(\circ) := 1 - \sum_{a \in G_\sigma} S_1(\Pi)(a)$.

(T) : For $a \in G_\sigma$: set $T(\Pi \xrightarrow{a} \langle s, a\rangle, \pi \overset{a}{\Rightarrow} \langle \sigma, a\rangle) := T(\Pi, \pi) \cdot S_1(\Pi)(a)$.
For $a \in \mathcal{C} \setminus G_\sigma$: set $T(\Pi \xrightarrow{a} \langle s, a\rangle, \pi \overset{\circ}{\Rightarrow} \langle \sigma_f, \circledcirc\rangle) := T(\Pi, \pi) \cdot S_1(\Pi)(a)$.
If $S_1(\Pi)(\circ) > 0$: set $T(\Pi \xrightarrow{\circ} \langle s, \circledcirc\rangle, \pi \overset{\circ}{\Rightarrow} \langle \sigma_f, \circledcirc\rangle) := T(\Pi, \pi) \cdot S_1(\Pi)(\circ)$.

Case 2: $\Pi = \hat{\Pi} \xrightarrow{a} \langle s, a\rangle, a \in \mathcal{C}$.

We give the following definitions for every $\pi \in \text{Paths}^{\mathcal{G}}$ with $T(\Pi, \pi) > 0$. By property (*), two cases arise:

(a) : $\langle \sigma_f, \circ\rangle = \text{last}(\pi)$: then $\langle s, a\rangle \xrightarrow{\times} \otimes$ in $\mathcal{G}$ (property (2)).

(T) : Set $T(\Pi \xrightarrow{\times} \otimes, \pi \overset{\circ}{\Rightarrow} \sigma_f) := T(\Pi, \pi)$.

(b) : $\langle \sigma, a\rangle = \text{last}(\pi)$: then $\sigma \in \gamma(s)$; let $d = \beta(s, a, \sigma)$.
Let $up_a = \langle \langle p_1, c_1\rangle, \ldots, \langle p_k, c_k\rangle\rangle$.

(T) : Set for every $1 \leq i \leq k$: $T(\Pi \xrightarrow{d} \langle s, a, d\rangle \xrightarrow{i} d(i), \pi \overset{i}{\Rightarrow} [\![c_i]\!](\sigma)) := T(\Pi, \pi) \cdot p_i$.

Now we define $S_2(\Pi)$:

(S_2) : Let M_a be the set of all paths π such that there exists $\sigma \in Q_A$ with $\langle \sigma, a\rangle = \text{last}(\pi)$. Let $t = \sum_{\pi \in \text{Paths}^{\mathcal{M}, s}} T(\Pi, \pi)$. Set $S_2(\Pi)(\times) := 1 - \frac{1}{t} \cdot \sum_{\pi \in M_a} T(\Pi, \pi)$.
For every possible sequence d:
let $M_{a,d}$ be the paths $\pi \in M_a$ having a last state $\langle \sigma, a\rangle$ s.t. $\beta(s, a, \sigma) = d$.
Set $S_2(\Pi)(d) := \frac{1}{t} \cdot \sum_{\pi \in M_{a,d}} T(\Pi, \pi)$. [1]

Case 3: $\Pi = \hat{\Pi} \xrightarrow{\circ} \langle s, \circledcirc\rangle$ respectively $\Pi = \hat{\Pi} \rightarrow q$, $q \in \{\circledcirc, \otimes\}$.

We carry out the following definitions for every $\pi \in \text{Paths}^{\mathcal{M}}$ with $T(\Pi, \pi) > 0$. The path π ends with $\langle \sigma_f, \circ\rangle$ respectively σ_f due to property (*).

[1] $S_2(\Pi)$ is then a distribution: the sets $M_{a,\cdot}$ form a partition of M_a.

$(S):$ Set $S(\pi)(\circ) := 1$ (if π ends with σ_f).

$(S_2):$ If $\Pi = \hat{\Pi} \to \langle s, \circledcirc\rangle$: set $S_2(\Pi)(\circ) := 1$.

$(T):$ Set $T(\Pi \overset{\circ}{\to} \circledcirc, \pi \overset{\circ}{\Rightarrow} \sigma_f) := T(\Pi, \pi)$.

This concludes the construction. For all other paths we again define S_1 and S arbitrarily. We state the easy fact that T is well-defined, i.e. no two definitions collide (follows from the definition of T and the distinctness of the three cases). Note that only S_1 is used for defining T, and that for every (S_1, S_2)-possible path there is a $\pi \in \text{Paths}^{\mathcal{M},S}$ with $T(\Pi, \pi) > 0$ (in particular, $t > 0$ holds in case 2). Again with an inductive argument we can prove that S is well-defined, i.e., no two definitions of S collide; for the proof see Lemma 21 in Appendix A.2.
The probabilities of (S_1, S_2)-possible paths can be obtained by summing up their T-values, i.e. for every $\Pi \in \text{Paths}^{\mathcal{G},(S_1,S_2)}$ it holds that

$$\Pr^{\mathcal{G},(S_1,S_2)}[\text{Cyl}(\Pi, \mathcal{G})] = \sum_{\pi \in \text{Paths}^{\mathcal{M},S}} T(\Pi, \pi) \tag{4.4}$$

(this can be shown by a simple induction, see Lemma 22 in Appendix A.2).
A dual result holds for every S-possible path π in $\mathcal{M}$, i.e. it holds that

$$\Pr^{\mathcal{M},S}[\text{Cyl}(\pi, \mathcal{M})] = \sum_{\Pi \in \text{Paths}^{\mathcal{G},(S_1,S_2)}} T(\Pi, \pi) \tag{4.5}$$

(see Lemma 23 in Appendix A.2). Now let $F_{\mathcal{G}}$ be the set of (S_1, S_2)-possible paths Π in $\mathcal{G}$ such that Π ends in $\circledcirc$ or $\otimes$ and has no other occurrence of $\circledcirc$ or $\otimes$. We can write

$$\Pr^{\mathcal{G},(S_1,S_2)}[\text{Reach}(\mathcal{G}, \{\circledcirc, \otimes\})] = \sum_{\Pi \in F_{\mathcal{G}}} \Pr^{\mathcal{G},(S_1,S_2)}[\text{Cyl}(\Pi, \mathcal{G})]$$

(note that $\text{Cyl}(\Pi, \mathcal{G}) \cap \text{Cyl}(\Pi', \mathcal{G}) = \emptyset$ for $\Pi \neq \Pi'$ paths in $F_{\mathcal{G}}$). Analogously let $F_{\mathcal{M}}$ be the set of S-possible paths π in $\mathcal{M}$ such that π ends in σ_f and has no other occurence of σ_f. Again it holds that

$$\Pr^{\mathcal{M},S}[\text{Reach}(\mathcal{M}, \{\sigma_f\})] = \sum_{\pi \in F_{\mathcal{M}}} \Pr^{\mathcal{M},S}[\text{Cyl}(\pi, \mathcal{M})].$$

We conclude:

$$\begin{aligned}
&\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Reach}(\mathcal{G},\{\circledcirc,\otimes\})] \\
&= \sum_{\Pi\in F_{\mathcal{G}}} \Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\Pi,\mathcal{G})] \\
&= \sum_{\Pi\in F_{\mathcal{G}}} \left(\sum_{\pi\in\mathrm{Paths}^{\mathcal{M},S}} T(\Pi,\pi) \right) && \text{(Eq. 4.4)} \\
&= \sum_{\Pi\in F_{\mathcal{G}}} \left(\sum_{\pi\in F_{\mathcal{M}}} T(\Pi,\pi) \right) && \text{(property (*))} \\
&= \sum_{\pi\in F_{\mathcal{M}}} \left(\sum_{\Pi\in F_{\mathcal{G}}} T(\Pi,\pi) \right) \\
&= \sum_{\pi\in F_{\mathcal{M}}} \Pr^{\mathcal{M},S}[\mathrm{Cyl}(\pi,\mathcal{M})] && \text{(Eq. 4.5)} \\
&= \Pr^{\mathcal{M},S}[\mathrm{Reach}(\mathcal{M},\{\sigma_f\})].
\end{aligned}$$

Proof of (3).
Let us restate the claim: Given a strategy $S_1 \in \mathcal{S}_1(\mathcal{G})$, there exists a strategy $S \in \mathcal{S}(\mathcal{M}_P)$ such that

$$\Pr^S \mathrm{Reach}(\mathcal{M}_P, F) \geq \inf_{T\in S_2(\mathcal{G})} \Pr^{(S_1,T)}[\mathrm{Reach}(\mathcal{G},\{\circledcirc\})].$$

The proof can be done very similarly to the previous one. However, in contrast to part (2), we have to "distinguish" between a run ending up in $\circledcirc$ and ending up in $\otimes$ in $\mathcal{G}$. Again we transform $\mathcal{M}_P$ into a slightlify modified MDP $\mathcal{M}$: we add the fresh labels $\circ$ and $\times$ to $\mathcal{M}_P$, two new states σ_f and σ_r to V_A and two new states $\langle\sigma_f,\circledcirc\rangle, \langle\sigma_r,\otimes\rangle$ to V_P. Both $\langle\sigma_f,\circledcirc\rangle$ respectively $\langle\sigma_r,\otimes\rangle$ have only one outgoing transition $(\langle\sigma_f,\circledcirc\rangle, 1, \circ, \sigma_f)$ respectively $(\langle\sigma_r,\otimes\rangle, 1, \times, \sigma_r)$; both σ_f respectively σ_r have only one outgoing transition $(\sigma_f, \circ, \sigma_f)$ respectively $(\sigma_r, \times, \sigma_r)$; we add all these transitions to $\Rightarrow$. We also add again edges $(q, \circ, \langle\sigma_f,\circledcirc\rangle)$ to $\Rightarrow$ for *every* $q \in F$ and remove all other outgoing transitions of states in F. Finally we add edges $(q, \times, \langle\sigma_r,\otimes\rangle)$ to $\Rightarrow$ for *every* $q \in V_A \setminus (F \cup \{\sigma_f, \sigma_r\})$. We denote this modified MDP again by $\mathcal{M}$. So

$$q \overset{\times}{\Rightarrow} \langle \sigma_r, \otimes \rangle \overset{1,\times}{\Longrightarrow} \sigma_r \overset{\times}{\Rightarrow} \sigma_r \overset{\times}{\Rightarrow} \dots$$

is possible for every $q \in V_A \setminus \{\sigma_f\}$. σ_r is the "counterpart" to $\otimes$ in this setting. Every run reaching a state in $q \in F$ ends in

$$q \overset{\circ}{\Rightarrow} \langle \sigma_f, \circledcirc \rangle \overset{1,\circ}{\Longrightarrow} \sigma_f \overset{\circ}{\Rightarrow} \sigma_f \overset{\circ}{\Rightarrow} \dots$$

We can show similarly to (2) that there exist strategies $S \in \mathcal{S}(\mathcal{M})$ and $S_2 \in \mathcal{S}_2(\mathcal{G})$ such that

$$\Pr^S[\text{Reach}(\mathcal{M}, \{\sigma_f\})] = \Pr^{(S_1,S_2),\mathcal{G}}[\text{Reach}(\mathcal{G}, \{\circledcirc\})].$$

This proves (3), since for every $S \in \mathcal{S}(\mathcal{M})$ there exists a strategy $S_{\mathcal{M}_P} \in S_1(\mathcal{M}_P)$ such that $\Pr^{\mathcal{M}_P,S_{\mathcal{M}_P}}[\text{Reach}(\mathcal{M}_P, F)] \geq \Pr^S[\text{Reach}(\mathcal{M}, \{\sigma_f\})]$ (distribute the probability with which S chooses $\times$ to other arbitrary labels). So for every $S \in \mathcal{S}(\mathcal{M})$ that satisfies the equality we have

$$\begin{aligned} &\Pr^{\mathcal{M}_P,S_{\mathcal{M}_P}}[\text{Reach}(\mathcal{M}_P, F)] \\ &\geq \Pr^{\mathcal{G},(S_1,S_2)}[\text{Reach}(\mathcal{G}, \{\circledcirc\})] \\ &\geq \inf_{T \in \mathcal{S}_2(\mathcal{G})} \Pr^{\mathcal{G},(S_1,T)}[\text{Reach}(\mathcal{G}, \{\circledcirc\})]. \end{aligned}$$

For the construction of the strategies and the completion of the proof we refer the reader to Appendix A.3. □

4.6 An Algorithm for Building Abstract Game Arenas

After proving that abstract game arenas can be used to obtain bounds for extremal reachability values, we now present Algorithm 1, our general algorithmic framework for building abstract game arenas $\mathcal{G}$ of a PGP P relative to a guard f. The method is instantiated by

- an abstract domain $D = (D^\sharp, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$,

Algorithm 1: Computing $\mathcal{G}$.

```
Input: PGP $P = (\mathcal{V}, \sigma_0, \mathcal{C})$, final guard $f$ with final states
       $F = \{\sigma \in \Sigma_{\mathcal{V}} \mid \sigma$ enables $f\}$, abstract domain
       $D = (D^\sharp, \sqsubseteq, \bigsqcup, \bigsqcap, \top, \bot)$, widening $\nabla$ for $D$.
Output: Abstract game arena
       $\mathcal{G} = (Q_1, Q_2, Q_P, s_0, E, \mathcal{C} \cup (D^\sharp)^* \cup \{\circ, \times, \tau\}, \mathbb{N})$.
$s_0 = \alpha(\{\sigma_0\})$; pred($s_0$) $\leftarrow$ nil;
$Q_1 \leftarrow \{s_0, \otimes, \circledcirc\}$; $Q_2 \leftarrow \emptyset$; $Q_P \leftarrow \emptyset$; $E \leftarrow \{(\circledcirc, \circ, \circledcirc), (\otimes, \times, \otimes)\}$;
WL $\leftarrow \{s_0\}$;
while $WL \neq \emptyset$ do
    Remove $s$ from the head of WL; processState($s$)

Procedure processState($s \in Q_1$):
$hasSucc \leftarrow$ false;
1  if $\gamma(s) \cap F \neq \emptyset$ then
2      $E \leftarrow E \cup \{(s, \circ, \langle s, \circledcirc \rangle), (\langle s, \circledcirc \rangle, \circ, \circledcirc)\}$;
3      if $\gamma(s) \not\subseteq F$ then
4          $E \leftarrow E \cup \{(\langle s, \circledcirc \rangle, \times, \langle s, \circledcirc \rangle))\}$
       $hasSucc \leftarrow$ true
forall $a \in \mathcal{C}$ do
5      if $g_a(\gamma(s)) \neq \emptyset$ then
6          $Q_2 \leftarrow Q_2 \cup \{\langle s, a \rangle\}$; $E \leftarrow E \cup \{(s, a, \langle s, a \rangle)\}$;
7          if $g_a(\gamma(s)) \neq \gamma(s)$ then  $E \leftarrow E \cup \{(\langle s, a \rangle, \times, \otimes)\}$;
8          $T \leftarrow$ abstractUpdate($s', a$);
           forall $t = \langle s_1, \ldots, s_k \rangle \in T$ do
               Create fresh array $d : \{1, \ldots, k\} \rightarrow D^\sharp$;
               forall $1 \leq i \leq k$ do
                   $s' \leftarrow$ extrapolate($s_i, s, a$) ;
                   if $s' \notin Q_1$ then
                       add $s'$ to WL; $Q_1 \leftarrow Q_1 \cup \{s'\}$; pred($s'$) $\leftarrow s$; ;
                   Set $i$-th entry $d(i)$ of $d$ to $s'$
               $Q_P \leftarrow Q_P \cup \{\langle s, a, d \rangle\}$;
               forall $1 \leq i \leq k$ do  $E \leftarrow E \cup \{(\langle s, a, d \rangle, p_a(i), i, d(i))\}$ ;
9               $E \leftarrow E \cup \{(\langle s, a \rangle, d, \langle s, a, d \rangle)\}$;
           $hasSucc \leftarrow$ true;
10 if $\neg hasSucc$ then $E \leftarrow E \cup \{(s, \tau, s)\}$

Procedure extrapolate($v \in D^\sharp, s \in Q_1 \setminus \{\circledcirc, \otimes\}, a \in \mathcal{C}$):
$\langle s', a' \rangle \leftarrow$ pred($s$);
while $pred(s') \neq nil$ do
    if $a' = a$ then  return $s \nabla (s \sqcup v)$;
    else  { buffer $\leftarrow s'$; $\langle s', a' \rangle \leftarrow$ pred($s'$); $s \leftarrow$ buffer }
return $v$
```

- a widening $\nabla : D^\sharp \times D^\sharp \to D^\sharp$,
- and a procedure `abstractUpdate`.

4.6.1 General Structure

The algorithm is inspired by the approaches of Beyer et al. [13] and Gulavani et al. [49] for constructing abstract reachability trees. It constructs the initial state $s_0 = \alpha(\{\sigma_0\})$ and generates transitions and successor states in a breadth-first fashion using a work list `WL`. The `processState` procedure constructs successors of a player 1 state guided by the rules from Def. 31. In the initialization phase of the algorithm, $\circledcirc$ and $\otimes$ and their transitions are added to $\mathcal{G}$. `processState` closely follows the definition of an abstract game arena. A widening is required to guarantee termination of the algorithm. It is used in the procedure `extrapolate`; in particular, for $x =$ `extrapolate`(s, s', a) it holds that $\gamma(x) \subseteq \gamma(s)$. We discuss `extrapolate` in Section 4.6.3. `abstractUpdate` is used to compute successors of abstract update operations. It can be tailored towards the used domain. However the following assumption has to hold *for every implementation* of `abstractUpdate`:

Requirement 1. (Requirement for `abstractUpdate`)
Let $s \in D^\sharp$ and $a \in \mathcal{C}$ with $up_a = \langle\langle p_1, c_1\rangle, \ldots, \langle p_k, c_k\rangle\rangle$. Then the function call `abstractUpdate`(s, a) has to return a set T of tuples of the form $\langle s_1, s_2, \ldots, s_k\rangle \in (D^\sharp)^k$. For every $\sigma \in g_a(\gamma(s))$ such that g_a is enabled by σ, there has to be a tuple $t = \langle s_1, \ldots, s_k\rangle \in T$ such that $[\![c_i]\!](\sigma) \in \gamma(s_i)$ for all $1 \leq i \leq k$. □

We give example instantiations satisfying this requirement in Section 4.6.2.

Correctness. Player 1 states that are generated by the algorithm satisfy the conditions (a)-(d) from Def. 31: lines 1-4 guarantee that condition (a) holds. Condition (b) follows from lines 5-6, the first part of condition (c) from the code between line 8 and line 9 together with the requirement for `abstractUpdate` that we stated above: let us assume that $s \in Q_1$ is processed by the algorithm, and let $\sigma \in \Sigma_\mathcal{V}$ be with $\sigma \in g_a(\gamma(s))$. One of the sequences $t = \langle s_1, \ldots, s_k\rangle \in T$ returned by the call to `abstractUpdate` satisfies that $c_i(\sigma) \in \gamma(s_i)$ for all $1 \leq i \leq k$. For the array d constructed using t it holds that $\gamma(d(i)) \subseteq \gamma(s_i)$ for all $1 \leq i \leq k$, due to the property of `extrapolate` we stated before, and $\langle s, a, d \rangle$ is the probabilistic state required by condition (c). The second part of condition (c) is guaranteed by line 7. Finally, *hasSucc* records whether s has successors; if not, we add a self-loop in line 29, and so satisfy condition (d).

Conservative tests. `processState` assumes that it can be decided whether $\gamma(s) \cap F \neq \emptyset$, $\gamma(s) \not\subseteq F$, $g_a(\gamma(s)) \neq \emptyset$ or $g_a(\gamma(s)) \neq \gamma(s)$ hold (lines 1,3,4, and 7). Each of these tests however can safely be replaced by a *conservative* decision procedure having the only requirement that if the exact test returns true, then also the conservative decision procedure (however there may be *false positives*). It is easy to see that then still all conditions (a)-(d) are satisfied, since we formulated them as implications. These conservative tests can e.g. be realized by using abstract transitions (recall Section 4.2), and this is what we did in our experiments. For example, if $(\neg f)^\sharp$ respectively $g_a^\sharp$ are abstract transitions for the negation of the guard f respectively for g_a, $\gamma(s) \not\subseteq F$ can be replaced by $(\neg f)^\sharp(s) \neq \bot$, and $g_a(\gamma(s)) \neq \emptyset$ can be replaced by $g_a^\sharp(s) \neq \bot$.

4.6.2 Procedure `abstractUpdate`

We first give an example of a generic instantiation of `abstractUpdate` that can be used for arbitrary domains. We use this instantiation for the experiments in Section 4.8. Afterwards we show how `abstractUpdate` can be implemented for predicate domains, which is in essence the method given in Hahn et al. [52], Wachter [98].

`abstractUpdate` for arbitrary domains. We slightly extend our notion of abstract transitions and assume that for every guarded command $a \in \mathcal{C}$ there exists $g_a^\sharp : D^\sharp \to 2^{D^\sharp}$ that satisfies, for all $s \in D^\sharp$,

- $\bigcup_{s' \in g_a^\sharp(s)} \gamma(s') \supseteq g_a(\gamma(s))$ ($g_a^\sharp$ is a safe over-approximation of g_a) and
- $|g_a^\sharp(s)| < \infty$ and $\bot \notin g_a^\sharp(s)$.

I.e. we allow $g_a^\sharp$ to return *a set of elements in* $D^\sharp$ instead of just one abstract state to get more precise results, e.g. if the guard is a disjunction (we make implicit use of finite powerset domains of $D^\sharp$, see e.g. Nielson et al. [81]). We also assume abstract transitions $c^\sharp : D^\sharp \to D^\sharp$ for the assignments c of every $\langle p, c\rangle \in up_a$. Now `abstractUpdate`$(s, a)$ can be implemented as follows: Let $up_a = \langle\langle p_1, c_1\rangle, \ldots, \langle p_k, c_k\rangle\rangle$. We use two variables R, B, each storing a set. First set $R := \emptyset$ and compute $B := g_a^\sharp(s)$. Then for every s' in B, add to R the tuple $\langle c_1^\sharp(s'), c_2^\sharp(s'), \ldots, c_k^\sharp(s')\rangle$. Return R.
It is easy to prove that this implementation satisfies Requirement 1: Let $\sigma \in g_a(\gamma(s))$. We have to find a tuple $t = \langle s_1, \ldots, s_k\rangle \in T$ such that $[\![c_i]\!](\sigma) \in \gamma(s_i)$ for all $1 \le i \le k$. There is $s' \in g_a^\sharp(s)$ such that $\sigma \in \gamma(s')$, due to the assumptions on $g_a^\sharp$. For every i with $1 \le i \le k$, $[\![c_i]\!](\sigma) \in \gamma(c_i^\sharp(s'))$, because $c_i^\sharp$ is an abstract transition, and therefore we can choose the tuple $t = \langle c_1^\sharp(s'), c_2^\sharp(s'), \ldots, c_k^\sharp(s')\rangle$.

`abstractUpdate` for predicate domains. For domains $\mathbb{PRED}(\phi_1, \ldots, \phi_n)$ (with $\phi_1, \ldots, \phi_n$ predicates over $\Sigma_\mathcal{V}$) we can implement `abstractUpdate` by reusing techniques of generating successor states in the predicate-abstraction approaches of Kattenbelt et al. [63], Wachter [98], Wachter and Zhang [99]: We assume the predicates $\phi_1, \ldots, \phi_n$ are representable as formulas of a decidable theory of first order logic, and that we have an SMT solver at our disposal which is capable of returning satisfying models for formulas of this theory. Let $s \in D^\sharp$ and $a \in \mathcal{C}$, with $up_a = \langle\langle p_1, c_1\rangle, \ldots, \langle p_k, c_k\rangle\rangle$. For an element $x \in D^\sharp \setminus \{\top, \bot\}$ let $\overline{x}$ be the representation of x as a conjunction, i.e. for $\sigma \in \Sigma$

$$\overline{x}(\sigma) := y_1(\sigma) \wedge y_2(\sigma) \wedge \ldots \wedge y_n(\sigma),$$

where $y_i \in \{\neg\phi_i, \phi_i\}$ for $1 \le i \le n$, and $y_i = \phi_i$ iff all states in x satisfy ϕ_i. We set $\overline{\bot} = \text{false}$ and $\overline{\top} = \text{true}$. We use the SMT solver to obtain *all* sequences $\langle d_1, \ldots, d_k\rangle \in (D^\sharp \setminus \{\top, \bot\})^k$ for which there exists a $\sigma \in \Sigma_\mathcal{V}$ such that

$$g_a(\sigma) \wedge \overline{d_1}([\![c_1]\!](\sigma)) \wedge \ldots \wedge \overline{d_k}([\![c_k]\!](\sigma))$$

is true. The approach is described in great detail e.g. in Wachter [98] (Chapter 6), where also optimizations are discussed. This instantiation of `abstractUpdate` shows that the construction step of menu-based abstraction can be seen as a special case of our algorithm. Kattenbelt describes in Kattenbelt [60] a similar method: he considers only finite-state programs, and so SAT-solvers can be used for generating the successor states.

4.6.3 Procedure `extrapolate`

We already saw a possible implementation of `extrapolate` in our introductory example: there we used as a domain the product of a (finite) predicate domain and the interval domain. Information represented in the first component of domain elements was used for picking existing states in the abstract game arena to performing widening operations. In our example, the first component represented control flow information (values of the variable `loc`), and widening was applied when, roughly speaking, the program visited the same control flow location again during the exploration.

We describe now a method that is applicable for *arbitrary* domains. During the construction we use the function pred$(\cdot)$ to store for every state $s \in Q_1 \backslash \{s_0, \circledcirc, \otimes\}$ its (Player 2-) predecessor in the spanning tree induced by the construction (we call it *the spanning tree* from now on; $pred(s) = \langle s', a \rangle$ implies $s' \xrightarrow{a} s$ in the spanning tree). For a state $s' \in Q_1$ that was created as the result of chosing a guarded command a, the procedure finds the nearest predecessor s in the spanning tree with the same property, and uses s to perform a widening operation. We can now prove:

Theorem 6. *Algorithm 1 terminates, and its result $\mathcal{G}$ is an abstract game arena.* $\square$

Proof. Assume for the sake of contradiction that Algorithm 1 does not terminate. Since every state in $\mathcal{G}$ has only finitely many successors, the spanning tree of $\mathcal{G}$ contains an infinite branch $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots$ by König's lemma, and at least one action $a \in \mathcal{C}$ appears infinitely often in the branch, since $\mathcal{C}$ is finite. Let $q_1, t_1, q_2, t_2 \ldots$ be the sequence of all states in the branch such that $q_l \xrightarrow{a} t_l$ for all $l \in \mathbb{N}$. Then, by the definition of `extrapolate`, there exists a sequence $v_1, v_2, \ldots$ of elements in $D^\sharp$ such that $t_{l+1} = t_l \mathbin{\nabla} (v_l \sqcup t_l) \sqsupseteq t_l$ for all $l \geq 1$, and so

$t_1 \sqsubseteq t_2 \sqsubseteq \ldots$ holds. Define $a^{(1)} := t_1$ and $a^{(l+1)} := v_{l+1} \sqcup t_{l+1}$ for $l \geq 1$. A simple induction shows that this sequence is monotonically increasing. Since $t_1 = a^{(1)}$ and for $l \geq 1$ it holds that $t_{l+1} = t_l \nabla (v_l \sqcup t_l) = t_l \nabla a^{(l)}$, we conclude from Def.22 from Section 4.2 that there is a number k such that $t_k = t_{k+1}$, a contradiction to the assumption that the branch is infinite (since then we would have a cycle).

We already pointed out that every newly constructed state in $\mathcal{G}$ satisfies the conditions from Def. 31. We conclude that $\mathcal{G}$ is indeed an abstract game arena. $\square$

4.7 Refining Abstract Game Arenas: Quantitative Widening Delay

Algorithm 1 applies the widening operator whenever the current state has a predecessor in the spanning tree that was created by applying the same guarded command. This strategy usually leads to too many widenings and poor abstract game arenas. A popular solution in nonprobabilistic abstract interpretation is to delay widenings in an initial stage of the analysis (Blanchet et al. [15]), in our case until the spanning tree reaches a given depth. We call this approach *depth-based unrolling.* Note that if $\mathcal{M}_P$ is finite and the application of widenings is the only source of imprecision, this simple refinement method is complete.

A shortcoming of this approach is that it is insensitive to the probabilistic information. We propose to combine it with another heuristic. Given an abstract game arena $\mathcal{G}$, our procedure yields two pairs (S_1^-, S_2^-) resp. (S_1^+, S_2^+) of memoryless and nonprobabilistic strategies that satisfy $\Pr^{\mathcal{G},(S_1^-,S_2^-)}[\text{Reach}(\mathcal{G}, \{\circledcirc\})] = \max^-(\mathcal{G}, \{\circledcirc\})$ respectively $\Pr^{\mathcal{G},(S_1^+,S_2^+)}[\text{Reach}(\mathcal{G}, \{\circledcirc\})] = \max^+(\mathcal{G}, \{\circledcirc\})$ (analogously for $\min^-$ and $\min^+$). Given a state s for Player 1, let P_s^+ and P_s^- denote the probability of reaching $\circledcirc$ (resp. $\circledcirc$ or $\otimes$ if we are interested in minimal probabilities) starting at s and obeying the strategies (S_1^+, S_2^+) resp. (S_1^-, S_2^-) in $\mathcal{G}$. In order to refine $\mathcal{G}$ we can choose any state $s \in Q_1 \cap D^\sharp$ such that $P_s^+ - P_s^- > 0$ (i.e., a state whose probability has not been computed exactly yet), such that at least one of the direct successors of s in the spanning tree has been constructed using a widening. We call these states the *candidates* (for delaying widening). The question is which candidates to select. We propose to use the following simple heuristic:

> Sort the candidates s according to the product $\eta_s = w_s \cdot (P_s^+ - P_s^-)$, where w_s denotes the product of the probabilities on the path of the spanning tree of $\mathcal{G}$ leading from s_0 to s. Choose the n candidates with largest product, for a given n.

We call this heuristic the *mass heuristic.* The *mixed heuristic* delays widenings for states with depth less than a threshold i, and for n states of depth larger than or equal to i with maximal η_s. In the next section we illustrate depth-based unrolling, the mass heuristic, and the mixed heuristic on some examples.
We mention that abstraction refinements based on counter-example guided abstraction refinement (like the sophisticated techniques developed e.g. in Wachter and Zhang [99] and Kattenbelt et al. [63]) can also be integrated in our framework, if we either choose a predicate domain or a product domain containing a predicate domain as a component. Such refinements then correspond to an extension of the used predicate domain.

4.8 Experiments

We have implemented a prototype of our approach on top of the Parma Polyhedra Library (Bagnara et al. [6]), which provides several numerical domains (see e.g. Bagnara et al. [5]). We present some experiments showing how simple domains like intervals can outperform predicate abstraction. We stress again that examples exhibiting the opposite behaviour are also easy to find: our experiments are not an argument against predicate abstraction, but an argument for abstraction approaches not limited to it.
If the computed lower and upper bounds differ by more than 0.01, we select refinement candidates using the different heuristics presented before and rebuild the abstract game arena. We used a Intel© core 2 duo machine with 4GB RAM running Linux. For computing extremal game values we used value iteration.
Two small programs. Consider the PGPs of Fig. 4.10. We compute bounds with different domains: intervals, octagons, integer grids, and the direct product of integer grids and intervals. For the refinement we use the mass (M) depth (D)

```
int a=0, ctr=0;
a1: (ctr=0)
    -> 0.5:(a'=1)&(ctr'=1)
      +0.5:(a'=0)&(ctr'=1);
a2: (ctr=1)&(a>=-400)&(a<= 400)
    -> 0.5:(a'=a+5)
      +0.5:(a'=a-5);
a3: (ctr=1) -> 1:(ctr'=2);
reach: (a=1)&(ctr=2)
```

```
int x=0, y=0, c=0;
a1: (c=0)&(x<=1000)
    -> 0.25:(x'=3*x+2)&(y'=y-x)
      +0.75:(x'=3*x)&(y'=30);
a2: (c=0)&(x>1000) -> 1:(c'=1);
a3: (c=1)&(x>=3) -> 1:(x'=x-3);
reach: (c=1)&(x=2)&(y>=30)
```

Figure 4.10: Two guarded-command programs.

Program	Value	Interval			Octagon			Grid			Product		
		M	D	Mix	M	D	Mix	M	D	Mix	M	D	Mix
Left	Iters:	23	81	24	28	81	28	1	1	1	1	1	1
	Time:	25	66.1	27.6	26.6	63.2	26.9	0.39	0.39	0.39	0.6	0.6	0.6
	Size:	793	667	769	681	691	681	17	17	17	61	61	61
Right	Iters:	-	-	-	-	-	-	-	-	-	3	7	3
	Time:	-	-	-	-	-	-	-	-	-	8.3	20.3	8.2
	Size:	-	-	-	-	-	-	-	-	-	495	756	495

Table 4.1: Experimental results for the programs in Fig. 4.10. Iters is the number of iterations needed. Time is given in seconds. '-' means the analysis did not return a precise enough bound after 10 minutes (i.e., lower and upper bounds still differed by more than 0.01). Size denotes the maximal number of states belonging to Player 1 that occured in one of the constructed games.

and mixed (Mix) heuristics. For M and Mix we choose 15 refinement candidates at each iteration. The results are shown in Table 4.1. For the left program the integer grid domain (and the product) computes precise bounds after one iteration. After 10 minutes, the PASS tool (Hahn et al. [52]) only provides the bounds $[0.5, 0.7]$ for the maximal reachability value. For the right program only the product of grids and intervals is able to "see" that $x \equiv 0 \pmod 3$ or $y < 30$ holds, and yields precise bounds after 3 refinement steps. After 10 minutes PASS only provides the bounds $[0, 0.75]$. The example illustrates how pure depth-based unrolling, ignoring probabilistic information, leads to poor results: the mass and mixed heuristics perform better. PASS may perform better after integrating appropriate theories, but the example shows that combining domains is powerful and easily realizable by using abstract interpretation tools.

Programs of Fig. 4.9. For these PASS does not terminate after 10 minutes, while using the interval domain our approach computes the exact value after at most 5 iterations and less than 10 seconds. Most of the predicates added by PASS during the refinement for program 2 have the form $\mathtt{c} \leq \alpha \cdot \mathtt{i} + \beta$ with $\alpha > 0, \beta < 0$: PASS's interpolation engines seem to take the wrong guesses during the generation of new predicates. This effect remains also if we change the refinement strategies of PASS. The tool offers the option of manually adding predicates. Interestingly it suffices to add a predicate as simple as e.g. $i > 3$ to help the tool to derive the solution after 3 refinements for program 2.

Zeroconf protocol (Interval domain)	$K = 4$ P1	$K = 6$ P1	$K = 8$ P1	$K = 4$ P2	$K = 6$ P2	$K = 8$ P2
Time (Mass heuristic):	6.2	16.8	32.2	5.8	18.5	50.6
Time (Depth heuristic):	2.6	6.0	6.6	2.6	6.7	8.1
Time (Mix):	2.6	6.3	6.8	2.6	6.9	8.4
Time PASS:	0.6	0.8	1.1	0.7	0.9	1.2

Table 4.2: Experimental results for the Zeroconf protocol. Time in seconds.

Zeroconf. This is a simple probabilistic model of the Zeroconf protocol, adapted from Kattenbelt et al. [63], where it was analyzed using PRISM and predicate abstraction. It is parameterized by K, the maximal number of probes sent by the protocol. We check it for $K = 4, 6, 8$ and two different properties. *Zeroconf* is a very good example for predicate abstraction, and so it is not surprising that PASS beats the interval domain (see Table 4.2). The example shows how the mass heuristic by itself may not provide good results either, with depth-unrolling and the mixed heuristics performing substantially better.

4.9 Related work

Apart from Hahn et al. [52], Hermanns et al. [55], Kattenbelt et al. [62, 63], Wachter [98] and Wachter and Zhang [99], Monniaux has studied in Monniaux [76] how to abstract probability distributions over program states (instead of the states themselves), but only considers upper bounds for probabilities, as already pointed out in Wachter and Zhang [99]. In Monniaux [77], the author analyses different quantitative properties of Markov decision processes, again using abstractions of probability distributions. In contrast, our approach constructs an abstract game arena using “nonprobabilistic” domains and widenings and then performs the computation of strategies and strategy values, which might be used for a refinement of the abstract game arena. Finally, in Di Pierro et al. [31] Hankin, Di Pierro, and Wiklicky develop a framework for probabilistic abstract interpretation which, loosely speaking, replaces abstract domains by linear spaces and Galois connections by special linear maps, and aims at computing expected values of random variables. In contrast, we stick to the standard framework, since in particular we wish to apply existing tools, and aim for upper and lower bounds of extremal reachability values.

4.10 Conclusion

We have shown in this chapter that game-based approaches for abstraction of probabilistic systems (Kattenbelt et al. [62, 63], Wachter and Zhang [99]) can be extended to arbitrary abstract domains, allowing probabilistic checkers to profit from well developed libraries for abstract domains like intervals, octagons, and polyhedra (Bagnara et al. [6], Jeannet and Miné [59]).
For this we presented an algorithm for constructing *abstract game arenas* from probabilistic programs. We showed how they can be used to obtain bounds for extremal reachability values of the MDP corresponding to the program. The algorithm can be instantiated by arbitrary abstract domains and widenings. Since a generalization of the proofs in Wachter and Zhang [99] and Kattenbelt et al. [62] did not seem to be straightforward, we provided a new correctness proof.
The new approach allows to refine abstractions using standard techniques like delaying widenings. We presented a technique that selectively delays widenings using a heuristics based on quantitative properties of the abstract game arenas.

Chapter 5

Termination of Probabilistic Programs

In the previous chapter we have used abstraction techniques to compute bounds for reachability probabilities. These bounds are valuable to assess the reliability of a program or the system which it models. However, besides obtaining *quantitative* bounds for probabilities, one is often interested in proving the *qualitative* property that a program reaches a set of states with a certain given probability. In this part of the thesis we study an important special case of this question, namely how to prove *almost-sure termination (a.s.-termination for short).* A program is almost-surely terminating if the probability of a program run reaching a terminating state is one. We develop a method for proving a.s.-termination for both finite-state programs and an important class of infinite-state programs. Almost-sure termination has to be distinguished from the "classical" notion of termination, which we call *sure* termination: a program is surely terminating if *every* program run eventually reaches a final state.

For many probabilistic programs, a.s.-termination is of critical importance: for example, protocols for distributed systems as well as randomized algorithms often do not enjoy sure termination, but a.s.-termination, which still gives an adequate termination warranty in virtually every practical setting.

For clarifying the different notions of termination consider for example the following program:

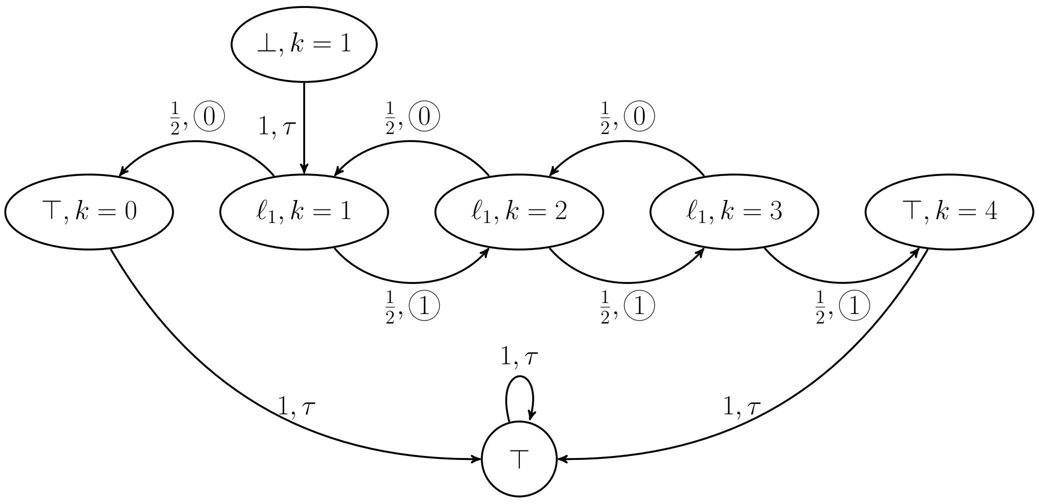


Figure 5.1: MDP of the introductory example. The transitions are labeled by the coin toss outcomes $\textcircled{0}$ and $\textcircled{1}$.

```
int k = 1;
l1: while (0 < k < 4) { if (coin(p)) k++ else k-- }
end.
```

Recall that `coin(p)` yields 1 (or "true") with probability $0 < p < 1$, and 0 (or "false") with probability $(1-p)$. We assume that p is a constant in the following. We can give it a semantics as an MDP $\mathcal{M}$ (see Fig. 5.1)[1]: states of $\mathcal{M}$ are program states, transitions are chosen according to the program flow and labeled with the outcome value of `coin(p)`-calls. We also add an initial state $\langle \bot, k = 1 \rangle$ and a final state $\top$. Terminating configurations, i.e. those that correspond to reaching the end location, are labeled by $\top$ and lead to the final state. The resulting MDP has no action states, and we therefore treat it as a Markov chain in the following. It is essentially equivalent to the random walk described in Fig. 3.1.

For checking sure termination we can replace the `coin(p)`-call by a nondeterministic choice operator that returns 0 or 1 nondeterministically. The modified program terminates surely iff the original one does; Fig. 5.1 then represents its transition system semantics (if we ignore the probabilistic weights on the tran-

[1]In this example, we "condensed" the MDP slightly for clarity, since we subsumed the entire semantic effect of one execution of the loop body into a single transition. See section 5.1 for a thorough semantic translation of probabilistic programs into MDPs.

sitions). Sure termination of programs with finitely many reachable states is a purely topological property of the transition system associated to the program, namely absence of cycles. Since Fig. 3.1 contains loops, the program cannot be surely terminating. A nonterminating program run is e.g.

$$\langle \bot, k = 1\rangle \to \langle \ell_1, k = 1\rangle \to \langle \ell_1, k = 2\rangle \to \langle \ell_1, k = 1\rangle \to \langle \ell_1, k = 2\rangle \to \dots$$

However, the program is almost surely terminating, since we can verify using Lemma 2 that $\Pr_{\langle \bot, k=1\rangle}[\text{Reach}(\mathcal{M}, \{\top\})] = 1$.
Sure termination implies almost-sure termination. However, as we can deduce from the example above, the converse implication is not true. For nonprobabilistic programs, i.e. programs where no probabilistic choices occur, both notions of termination trivially coincide.
Proving sure termination is a fundamental challenge of computer science from its early beginnings. One of the most important results from computability theory states that the problem of deciding whether a program is surely terminating is undecidable for general programs. Nonetheless sure termination is expressible in temporal logic, and so in theory LTL or CTL model checkers can be used to tackle the problem. However, recent research has shown that special purpose tools, like Terminator (Cook et al. [22]) and ARMC (Podelski and Rybalchenko [85]), and techniques like *transition invariants*, can be dramatically more efficient (Podelski and Rybalchenko [86, 87], Rybalchenko [92]). Since almost-sure termination is an equally important property for randomized algorithms and probabilistic protocols as sure termination is for nonprobabilistic programs, the question arises whether the very strong advances in automatic termination proving can be exploited in the probabilistic case.
However, without further restricting the question, the answer is negative. Consider for instance the program

```
int k = 1;
l1: while (0 < k) { if (coin(p)) k++ else k-- }
end.
```

whose corresponding MDP is in fact equivalent to the unbounded random walk (see Fig. 3.2). The program has the same executions for all values of p (only their probabilities change), but it only terminates a.s. for $p \leq 1/2$. This shows that proving a.s.-termination requires arithmetic reasoning not offered by termination provers (we will see a possible way of solving the problem for simple programs like the one above in Chapter 6).

The situation changes if we restrict our attention to *weakly finite* probabilistic programs. Loosely speaking, a program is weakly finite if the set of states reachable from any initial state is finite. Notice that the state space may be infinite, because the set of initial states may be infinite. Weakly finite programs are a large class, which in particular contains *parameterized probabilistic programs*, i.e., programs with parameters that can be initialized to arbitrary large values, but are finite-state for every valuation of the parameters. One can show that a.s.-termination is a topological property for weakly finite programs. If the program does not contain nondeterministic choices, then it terminates a.s. iff for every reachable state s there is a path leading from s to a terminating state, which corresponds to the CTL property $AG\,EF\,end$ (in the nondeterministic case there is also a corresponding topological property).

As in the nonprobabilistic case that we discussed above, generic infinite-state model checkers perform poorly for these properties because of the quantifier alternation $AG\,EF$. In particular, CEGAR approaches usually fail, because, crudely speaking, they tend to unroll loops, which is essentially useless for proving termination.

The same problem appears in particular if we try to apply game-based abstraction techniques. Here we have to obtain *exact* extremal reachability values for proving a.s.-termination, i.e. upper and lower bounds of the minimal reachability values have both to be 1. In our experiments we learned that this is not feasible in practice for weakly finite programs, see also Section 5.6. Interestingly though, we show that termination information can help increasing the efficiency of abstraction techniques in Section 5.7.

In Arons et al. [3], Arons, Pnueli and Zuck present a different and very elegant approach that reduces *a.s.-termination* of a *probabilistic* program to *sure termination* of a *nondeterministic* program obtained with the help of a *Planner*. A Planner occasionally and infinitely often determines the outcome of the next k random choices for some fixed k, while the other random choices are performed nondeterministically. The planner approach is based on the following proof rule, with P a probabilistic program and R a measurable set of runs of P:

$$\frac{\Pr[R] = 1 \qquad \text{Every } r \in R \text{ is terminating}}{P \text{ terminates a.s.}}$$

In the following we generalize this approach, with the goal of profiting from recent advances on termination tools and techniques not available when Arons et al. [3] was published. While we also partially fix the outcome of random choices, we do so more flexibly with the help of *patterns*. A first advantage of patterns is that we are able to obtain a *completeness* result for weakly finite programs, which is not the case for Planners. Further, in contrast to Arons et al. [3], we show how to automatically derive patterns for finite-state and weakly finite programs using an adapted version of the CEGAR approach (for weakly finite programs, a simple extrapolation step is used). For this we give an algorithm which exploits the power of state-of-the-art model checkers and termination provers for non-probabilistic programs: it calls such tools within a refinement loop and thereby iteratively constructs a "terminating pattern", which is a set of terminating runs with probability one.

This chapter is mostly based on joint work with Javier Esparza and Stefan Kiefer; parts of the work have been published in Esparza et al. [39] and Esparza et al. [40].

Structure of the chapter. We first introduce our program model for proving a.s.-termination, called probabilistic imperative programs (PIPs). We then present patterns and show that, as a proof method, they are complete for finite and weakly finite programs. It follows a description of our algorithms for constructing patterns automatically respectively semi-automatically for finite respectively general weakly finite programs. We report on various case studies illustrating the effectiveness of our algorithm. Finally, we apply our technique to improve algorithms for *quantitative* probabilistic verification based on abstraction refinement as discussed in Chapter 4 and Kattenbelt et al. [62, 63], Wachter and Zhang [99]. We close with related work and a short conclusion.

5.1 Probabilistic Imperative Programs

We introduce probabilistic imperative programs (PIPs for short), a standard program model that is very similar to the one in Kattenbelt et al. [63].
The reason for introducing a new model different from the one in Chapter 4 is that the tools we use take imperative C-like programs as input; also most of our test cases are naturally given in an imperative-style language. Differences are mainly syntactical, and translating programs from one model into the other is straightforward.
PIPs are given in form of flow graphs whose transitions are labeled with *commands*. Let $\mathcal{V}$ be a set of variable names over the integers (the variable domain could be easily extended) and let $\Sigma_{\mathcal{V}}$ be the set of possible *valuations* of $\mathcal{V}$, also called *states*. The set of commands contains

- conditional statements, i.e., boolean combinations of expressions $e \leq e'$, where e, e' are arithmetic expressions (e.g, $x + y \leq 5 \wedge y \geq 3$);
- deterministic assignments $x := e$ and nondeterministic assignments $x :=$ nondet() that nondeterministically assign to x the value 0 or 1;
- probabilistic assignments $x := \text{coin}(p)$ that assign to x the value 0 or 1 with probability p or $(1 - p)$, respectively, where $0 < p < 1$ is a constant.

We can now define our program model:

Definition 32. (Probabilistic Imperative Program)
A *probabilistic imperative program (PIP for short) P* is a tuple $(\mathcal{L}, I, \hookrightarrow, \text{label}, \perp, \top)$, where

- $\mathcal{L}$ is a finite set of control flow *locations*,
- $I \subseteq \Sigma_{\mathcal{V}}$ is a set of *initial configurations*,
- $\hookrightarrow \subseteq \mathcal{L} \times \mathcal{L}$ is the *flow relation*,
- label is a function that assigns a command to each edge,
- $\perp \in \mathcal{L}$ is the *start location*, and $\top \in \mathcal{L}$ is the *end location*.

$$\begin{tikzcd}
\bot \arrow[r, "k \geq 100?"] \arrow[d, "k < 100?"] & \top \\
\ell_1 \arrow[d, "oldx := x"] & \\
\ell_2 \arrow[d, "x := \mathrm{coin}(p)"] & \\
\ell_3 \arrow[r, "x \neq oldx?"'] \arrow[uuu, bend left, "x = oldx?"] & \ell_4 \arrow[uuul, bend right, "k = k + 1"']
\end{tikzcd}$$

```
int k = 0, oldx = 0, x = 0;
while (k < 100) {
  oldx = x;
  x = coin(p);
  if (x != oldx) k++
}
end.
```

Figure 5.2: Example program `FW`, given as a flow graph on the left and as a code listing on the right.

We call the elements of $\hookrightarrow$ *edges*, and as usual we write $l \hookrightarrow l'$ for $(l, l') \in \hookrightarrow$. The following standard conditions must hold:

(i) the only outgoing edge of $\top$ is $\top \hookrightarrow \top$,

(ii) either all or none of the outgoing edges of a location are labeled by conditional statements; if all, then every configuration satisfies the condition of exactly one outgoing edge; if none, then the location has exactly one outgoing edge,

(iii) if an outgoing edge of a location is labeled by an assignment, then it is the only outgoing edge of this location.

A location is *nondeterministic* if it has an outgoing edge labeled by a nondeterministic assignment, otherwise it is *deterministic*. Deterministic locations can be *probabilistic* or *nonprobabilistic*. A PIP is *deterministic* if all its locations are deterministic. $\square$

Observe the program `FW` shown on the left side of Fig. 5.2. It is an adaption of a part of the FireWire protocol (McIver et al. [73]). The initial configuration assigns 0 to `k`, `x`, and `oldx`. `FW` can be represented by the C-like code shown on the right side of Fig. 5.2.

In the following we often use a representation of PIPs as program listings for the sake of brevity.

5.1.1 Semantics of PIPs

The semantics of a PIP is given by an MDP:

Definition 33. (Semantics of Probabilistic Imperative Programs)
Let P be a *PIP* $(\mathcal{L}, I, \hookrightarrow, \text{label}, \bot, \top)$, and let $\mathcal{L}_D, \mathcal{L}_A$ denote the sets of deterministic and nondeterministic locations of P. The semantics of P is the MDP $\mathcal{M}_P = (Q_A, Q_D, \text{Init}, \to, \text{Lab}_A, \text{Lab}_P)$, where $Q_A = \mathcal{L}_A \times \Sigma_{\mathcal{V}}$ is the set of nondeterministic states, $Q_D = ((\mathcal{L} \setminus \mathcal{L}_A) \times \Sigma_{\mathcal{V}}) \cup \{\top\}$ is the set of deterministic states, $\text{Init} = \{\bot\} \times I$ is the set of initial states, $\text{Lab}_A = \{a_0, a_1\}$ is the set of action labels, $\text{Lab}_P = \{\tau, 0, 1\}$ is the set of probabilistic labels, and the relation $\to$ is defined as follows: for every state $v = \langle \ell, \sigma \rangle$ of $\mathcal{M}_P$ and every edge $\ell \hookrightarrow \ell'$ of P

- if $\text{label}(\ell, \ell') = (x := \text{coin}(p))$, then $v \xrightarrow{p,0} \langle \ell', \sigma[x \mapsto 0] \rangle$ and $v \xrightarrow{1-p,1} \langle \ell', \sigma[x \mapsto 1] \rangle$;
- if $\text{label}(\ell, \ell') = (x := \text{nondet}())$, then $v \xrightarrow{a_0} \langle \ell', \sigma[x \mapsto 0] \rangle$ and $v \xrightarrow{a_1} \langle \ell', \sigma[x \mapsto 1] \rangle$;
- if $\text{label}(\ell, \ell') = (x := e)$, then $v \xrightarrow{1,\tau} \langle \ell', \sigma[x \to e(\sigma)] \rangle$, where $\sigma[x \to e(\sigma)]$ denotes the state obtained from σ by updating the value of x to the expression e evaluated under σ;
- if $\text{label}(\ell, \ell') = c$ for a conditional c satisfying σ, then $v \xrightarrow{1,\tau} \langle \ell', \sigma \rangle$.

Further for each state $v = \langle \top, \sigma \rangle$, $v \xrightarrow{1,\tau} \top$ holds; $\top \xrightarrow{1,\tau} \top$ is the only outgoing transition of $\top$. □

We distinguish between terminating and nonterminating runs of a PIP:

Definition 34. (Terminating runs)
Let P be a PIP. A run $r \in \text{Runs}^{\mathcal{M}_P}$ is *terminating* if it reaches the final state $\top$ of P, i.e. if $r \in \text{Reach}(\mathcal{M}_P, \{\top\})$; otherwise r is *nonterminating*. □

We can now define almost-sure termination of a PIP:

Definition 35. (Almost-sure termination)
A PIP $P = (\mathcal{L}, I, \hookrightarrow, \text{label}, \bot, \top)$ is *a.s.-terminating* if

$$\Pr^{\mathcal{M}_P, S}\Big[\{r \in \text{Runs}^{\mathcal{M}_P} \mid r \text{ is nonterminating } \}\Big] = 1$$

for every strategy S, i.e. $\mathit{MinReach}(\mathcal{M}_P, \{\top\}) = 1$. $\square$

5.1.2 Program Classes

We mainly investigate *finite* and *weakly finite* PIPs in the following. A PIP is

- *finite* if finitely many states are reachable from the initial states of $\mathcal{M}_P$;
- *weakly finite* if P_b is finite for all $b \in I$, where P_b is obtained from P by fixing b as the only initial state.

Assumption. In the following sections the focus is on *deterministic* programs, and we will assume (in particular in examples and explanations) that the programs to be analyzed are deterministic. However we will clearly state this assumption in theorems. We will discuss nondeterministic programs in Section 5.5.

5.2 Patterns

In this section we introduce our notion of *patterns*. We recall our example program `FW` from the previous section, which is a finite PIP:

```
int k = 0, oldx = 0, x = 0;
while (k < 100) {
  oldx = x;
  x = coin(p);
  if (x != oldx) k++
}
end.
```

Loosely speaking, `FW` terminates a.s. because if we keep tossing a coin then with probability 1 we observe 100 times two consecutive tosses with the opposite outcome (we even see 100 times the outcome 01). More formally, let us write

$C := \{0,1\}$ for the set of coin toss outcomes in the following, and let us identify a *run* of FW with the sequence of 0's and 1's corresponding to the results of the coin tosses carried out during it. For instance, $(01)^{51}$ and $(001100)^{50}$ are terminating runs of FW, and 0^ω is a nonterminating run. Every run that is a prefix of $(C^*01)^\omega$ eventually reaches the end-statement, and the set of all such runs has probability 1. From this it follows that FW terminates with probability 1, i.e. is a.s.-terminating. But it is easy to see that the runs that are prefixes of $(C^*01)^\omega$ are also the runs of the following nondeterministic program FW':

```
int k = 0, oldx = 0, x = 0;
int c1 = ?, c2 = 2;
while (k < 100) {
  oldx = x;
  if (c1 > 0)      { x = nondet(); c1-- }
  elseif (c2 = 2 ) { x = 0; c2-- }
  elseif (c2 = 1 ) { x = 1; c2-- }
  else /* c1 = 0 and c2 = 0 */ { c1 = ?; c2 = 2 }
  if (x != oldx) k++
}
end.
```

The assignment c = ? nondeterministically sets c to an arbitrary nonnegative integer. FW' sets $c1$ to some value and repeatedly tosses the coin (but nondeterministically), decreasing the counter $c1$ each time, until it reaches 0; at this point, FW' sets x first to 0 and then to 1, sets the counters again, and iterates. Termination of FW' can easily be proved with the help of termination checkers like ARMC, thus we have proved a.s.-termination of FW.

The set $(C^*01)^\omega$ is an example of a *pattern*. We use patterns to restrict a probabilistic program by imposing particular sequences of coin toss outcomes on the program runs.

In the rest of the section we introduce patterns formally, and prove properties that allow us to use them as a proof method for a.s.-termination. We recall our definition $C := \{0,1\}$. We fix a PIP $P = (\mathcal{L}, I, \hookrightarrow, \text{label}, \bot, \top)$ and its associated MDP $\mathcal{M}_P = (Q_A, Q_P, \text{Init}, \to, \text{Lab}_A, \text{Lab}_P)$, and set $Q = Q_A \cup Q_P$. We first introduce the concept of *traces*:

Definition 36. (Traces)
Let $\mathcal{M} = (Q_A, Q_P, \mathrm{Init}, \rightarrow, \mathrm{Lab}_A, \mathrm{Lab}_P)$ be an MDP and $\mathrm{Lab} = \mathrm{Lab}_A \cup \mathrm{Lab}_P$. The *trace* of a run $r = q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} \ldots \in \mathrm{Runs}^{\mathcal{M}}$, denoted by $\bar{r}$, is the infinite sequence $\alpha_1\alpha_2 \ldots \in (\mathrm{Lab})^*$ of its labels. Given $\Sigma \subseteq \mathrm{Lab}$, we define $\bar{r}|_\Sigma$ as the projection of $\bar{r}$ onto Σ. □

Observe that $\bar{r}|_\Sigma$ can be finite.
We introduce patterns as a special class of sets of traces:

Definition 37. (Patterns)
A *pattern* is a subset of C^ω of the form $C^*w_1C^*w_2C^*w_3\ldots$, where $w_1, w_2, \ldots \in C^*$. We say the sequence $w_1, w_2, \ldots$ *induces* the pattern. A pattern is *simple* if it is of the form $(C^*w)^\omega$. If $w_1, w_2, \ldots$ is an enumeration of C^*, we call $C^*w_1C^*w_2C^*w_3\ldots$ a *universal* pattern. □

For a pattern $\Phi \subseteq C^\omega$, a run $r \in \mathrm{Runs}^{\mathcal{M}_P}$ is Φ-*conforming* if there is $v \in \Phi$ such that $\bar{r}|_C$ is a prefix of v. We call Φ *terminating (for P)* if all Φ-conforming runs terminate, i.e., reach $\top$. Note also that for each word $w \in C^*$, the set C^*wC^ω is a pattern, induced by the sequence $w, \epsilon, \epsilon, \ldots$.
The following theorem illustrates the usefulness of patterns for proving a.s.-termination:

Theorem 7. *(Pattern as a proof method)*

(1) Let Φ be a pattern. The set of Φ-conforming runs has probability 1. In particular, if Φ is terminating, then P is a.s.-terminating.

(2) If P is a.s.-terminating, deterministic and finite with $n < \infty$ reachable states in $\mathcal{M}_P$, then there exists a word $w \in C^*$ with $|w| \in \mathcal{O}(n^2)$ such that C^*wC^ω is terminating for P, and $(C^*w)^\omega$ is a terminating simple pattern for P.

(3) If P is a.s.-terminating, deterministic and weakly finite, then every universal pattern is terminating for P.

□

Part (1) of Theorem 7 is the basis for the pattern approach. It allows to ignore the set of runs that are not Φ-conforming, because it has probability 0. Part (2) indicates that for finite programs there is a short word such w that $(C^*w)^\omega$ is terminating. Part (3) states that the pattern approach is "complete" for a.s.-termination and weakly finite programs: for any a.s.-terminating and weakly finite program there is a terminating pattern (in fact every universal pattern is terminating).

Proof of Theorem 7.
Part (1) (Sketch, for the full proof see App. B):
We can show that the set of runs r that visit infinitely many probabilistic locations and do not have the form $C^*w_1C^\omega$ has probability zero. This result can then easily be generalized to the set of runs visiting infinitely many probabilistic locations and not having the form $C^*w_1C^*w_2\dots C^*w_iC^\omega$, for every $i \geq 2$. The set of all runs not conforming to Φ is the countable union of all these sets, which has again probability zero.
Part (2) :
We define the term "ending up" as follows: A state $q \in Q$ *ends up in* a state $q' \in Q$ *following* $w = c_1c_2\dots c_m \in C^*$ iff

$$q \xrightarrow{\tau^*c_1\tau^*c_2\tau^*\dots\tau^*c_m\tau^*} q', \text{ and either } q' = \langle \ell, \sigma \rangle \text{ with } \ell \textit{ probabilistic} \text{ or } q' = \top.$$

Note that q' is unique if it exists. For every reachable state q and every sequence $w \in C^*$ holds that either: (i) q ends up in a state $\langle \ell, \sigma \rangle$ with ℓ probabilistic following w, or (ii) q ends up in $\top$ following a (possibly empty or not proper) prefix of w. Otherwise there would exist a state $\tilde{q}$ from which no state with probabilistic location or $\top$ is reachable any more, which contradicts that P is a.s.-terminating.
For every state $q \in Q$, there exists a sequence $c_1c_2\dots c_m \in C^*$ such that q ends up in $\top$ following the sequence, again due to the a.s.-termination property of P. We fix for every $q \in Q$ such a sequence $w(q)$; note that we can choose $w(q)$ such that $|w(q)| < n$ by removing cycles. We set $w(\top) = \epsilon$.
We construct a sequence $w^{(0)}, w^{(1)}, \dots, w^{(m)}$ using the following algorithm. Set $w^{(0)} := \epsilon$ and $i := 1$.

1. Pick a $q'_i \in Q$ that does end up in a state $q_i \neq \top$ following $w^{(i-1)}$. If no such q'_i exists set $w := w^{(i-1)}$ and terminate.

2. Set $w^{(i)} := w^{(i-1)}w(q_i)$. Set $i := i + 1$ and go to (1).

See Fig. 5.3 for an illustration of the algorithm.

The state set $Q^{(i)}$ in which a state of Q might end up after following $w^{(i)}$ contains at most $n - i$ states. This is certainly true for $i = 0$. In the i-th iteration the chosen q'_i ends up in $q_i \in Q^{(i-1)}$ after following $w^{(i-1)}$. After following $w(q_i)$, q_i ends up in $\top$. Thus q'_i ends up in $\top$ after $w^{(i)}$. This implies that $|Q^{(i)}| < |Q^{(i-1)}|$, since every state can end up in at most one state after following a nonempty coin sequence, and every $Q^{(i)}$ contains $\top$.

After at most $m \leq n - 1$ iterations, $|Q^{(m)}| \leq 1$ (i.e., $Q^{(m)} = \{\top\}$), and the algorithm terminates. Hence $|w^{(m)}| \leq (n - 1) \cdot \max_{q \in Q} |w(q)| \leq (n - 1)^2$. Every state of Q ends up in $\top$ after following a prefix of $w^{(m)}$. If it ended up in another state $\hat{q}$, the algorithm would have performed another iteration (making the computed sequence element longer); if there were no state q' such that q ends up in q', a prefix of w must have led it to $\top$ before.

We can conclude that every run r for which $\bar{r}|_C$ is a prefix of a word $C^*w^{(m)}C^\omega$ is terminating. Using $(C^*w^{(m)})^\omega \subseteq C^*w^{(m)}C^\omega$ it also follows that $(C^*w^{(m)})^\omega$ is terminating. This proves the claim.

Part (3) :

Let $\sigma_1, \sigma_2, \ldots$ be a (countable or infinite) enumeration of the states in I. Using Part (2) we obtain for each $i \geq 1$ a word w_i such that $C^*w_iC^\omega$ is terminating for P, if the only starting state considered is σ_i. Every universal pattern is a subset of $C^*w_iC^\omega$ for every $i \geq 1$ (by its definition), so it is also terminating. □

5.3 Constructing Patterns

Theorem 7 (3) guarantees that, for every weakly finite a.s.-terminating program P, every universal pattern is terminating. This suggests the following method for proving a.s.-termination of arbitrary, weakly finite P:

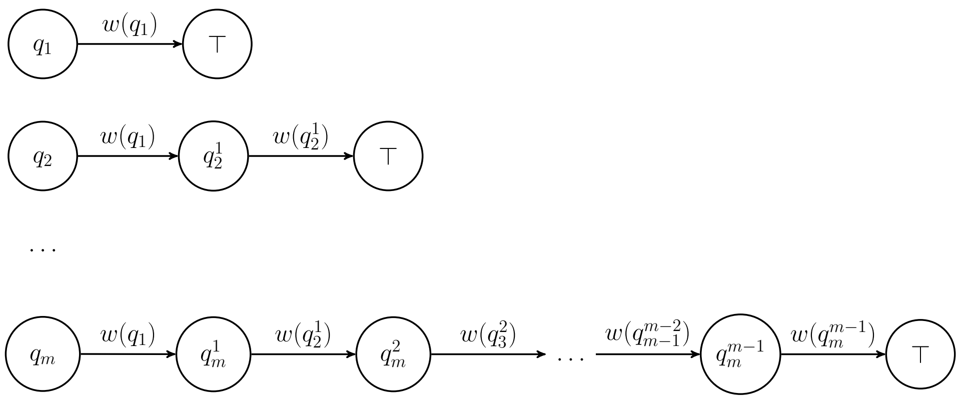


Figure 5.3: Illustration for the proof of Theorem 7, part 2. The algorithm chooses successively $q_1, q_2, \dots, q_m$. The resulting sequence $w^{(m)} = w(q_1)w(q_2^1)\dots w(q_{m-1}^{m-2})w(q_m^{m-1})$ leads all states to $\top$.

1. replace in P all probabilistic assignments by nondeterministic ones and instrument the program so that all its runs are conforming to a universal pattern;

2. check the resulting program for termination with a termination checker such as ARMC (Podelski and Rybalchenko [85]).

Although this approach is sound and complete (modulo the strength of the termination checker), our experience is that it turns out to be rather useless in practice. This is because the crucial loop invariants are extremely hard to catch for termination checkers. Already the instrumentation that produces the enumeration of C^* requires a nontrivial procedure (such as a binary counter) whose loops are difficult to analyze.

Therefore we devise in the following another algorithm that tries to compute a terminating pattern $C^* w_1 C^* w_2 \dots$. The algorithm operates on P and is "refinement"-based. It uses a "pattern checker" subroutine which takes a sequence $w_1, w_2, \dots$, and checks (or attempts to check) whether the induced pattern is terminating. If it is not, the pattern checker may return a *lasso* as counterexample. Formally, a lasso is a sequence

$$\langle l_1, \sigma_1 \rangle \to \langle l_2, \sigma_2 \rangle \to \ldots \to \langle l_m, \sigma_m \rangle \to \ldots \to \langle l_n, \sigma_n \rangle \quad \text{with } \langle l_n, \sigma_n \rangle \to \langle l_m, \sigma_m \rangle$$

with $\langle l_1, \sigma_1 \rangle \in \text{Init}$ and for $1 \leq i < j \leq n$ it holds that $\langle l_i, \sigma_i \rangle \neq \langle l_j, \sigma_j \rangle$. We call the sequence $\langle l_m, \sigma_m \rangle \to \ldots \to \langle l_n, \sigma_n \rangle$ the *lasso loop* of the lasso. Note that a lasso naturally induces a unique run in $\text{Runs}^{\mathcal{M}_P}$. If P is finite, pattern checkers can be made complete, i.e., they either prove the pattern terminating or return a lasso.

We first show how to automatically construct simple terminating patterns for finite programs. For this we use a finite-state model checker (in our experiments SPIN (Holzmann [56])). We then describe how to construct patterns for weakly finite programs using the procedure for finite-state programs as a building block. Details about the implementation of pattern checkers and in particular about the instrumentation of programs is given in Section 5.4.

For the rest of the section we fix a PIP $P = (\mathcal{L}, I, \hookrightarrow, \text{label}, \perp, \top)$ and its associated MDP $\mathcal{M}_P = (Q_A, Q_P, \text{Init}, \to, \text{Lab}_A, \text{Lab}_P)$.

5.3.1 Finite Programs

Using Theorem 7, we can prove a.s.-termination for every finite PIP P by finding a simple terminating pattern Φ and constructing a nondeterministic program P' whose runs are the Φ-conforming runs of P, and proving that P' terminates. It remains the question of how to find a simple terminating pattern for finite programs.

The proof of Theorem 7 (3) in fact gives us a method for computing such a pattern. However, the construction operates on the Markov chain $\mathcal{M}_P$, which is expensive to compute. Moreover, we will see later that it is advantageous to construct simple patterns $(C^*w)^\omega$ with w as short as possible (in particular if we invoke nonprobabilistic termination provers), which this construction does not provide. We therefore devise a procedure which operates on P, utilizes (nonprobabilistic) verification tools, such as model checkers and termination provers, and computes a terminating simple pattern $(C^*w)^\omega$ for P with w having minimal length.

An example. We first give an explanation of our algorithm using an example. Assume we want to find a terminating pattern for the finite program given in the introduction (see also Fig. 5.1):

```
int k = 1;
l1: while (0 < k < 4) { if (coin(p)) k++ else k-- }
end.
```

First we check if some nonterminating run of the program conforms to $\Phi_0 = C^\omega$, and get (for example) $v_1 = (10)^\omega$ as answer. This corresponds to the lasso

$$\langle \bot, k = 1 \rangle \rightarrow \langle \ell_1, k = 1 \rangle \rightarrow \langle \ell_1, k = 2 \rangle \rightarrow \langle \ell_1, k = 1 \rangle \rightarrow \langle \ell_1, k = 2 \rangle \rightarrow \dots$$

in the MDP of the example (see Fig. 5.1). We compute a *spoiler* w_1 of v_1, i.e. a finite word that is not an infix of v_1. The algorithm yields $w_1 = 00$. We now check if some nonterminating run of the program conforms to $\Phi_1 = (C^* w_1)^\omega$, get $v_2 = (1100)^\omega$ as counterexample, and construct a spoiler w_2 of *both* v_1 and v_2: a finite word that is an infix of *neither* v_1^ω *nor* v_2^ω. We get $w_2 = 000$, and check if some nonterminating run of the program conforms to $\Phi_2 = (C^* w_2)^\omega$. The checker finds no counterexamples, and so Φ_2 is terminating.

The algorithm. We now give a more detailed description of the approach. Let P be a finite PIP. Our algorithm may take a *base word* $s_0 \in C^*$ as input, which is set to $s_0 = \epsilon$ by default. Further it uses a counter variable i which is set to 0 in the beginning. The algorithm then proceeds as follows:

1. It runs the pattern checker on $C^* s_i C^* s_i \dots$.

2. If the pattern checker shows the pattern terminating for P, then, by Theorem 7 (1), P is a.s.-terminating, and the algorithm terminates. Otherwise the pattern checker returns a lasso $\langle l_1, \sigma_1 \rangle \rightarrow \dots \rightarrow \langle l_m, \sigma_m \rangle \rightarrow \dots \rightarrow \langle l_n, \sigma_n \rangle$.

3. The algorithm extracts from the lasso loop a word $u_{i+1} \in C^*$, which indicates a sequence of outcomes of the coin tosses in the lasso loop. If $u_{i+1} = \epsilon$, then the pattern checker has found a nonterminating run with

only finitely many coin tosses, hence P is not a.s.-terminating. Otherwise (i.e., if $u_{i+1} \neq \epsilon$), let $s_{i+1} \in C^*$ be a shortest word (the spoiler) such that

- s_0 (the base word) is a prefix of s_{i+1} and
- s_{i+1} is not an infix of any u_j^ω with $1 \leq j \leq i$.

We give details on how to construct such an s_{i+1} in the proof of Prop. 4.

4. i is set to $i+1$ and the algorithm goes to step (1).

The algorithm is complete for finite and a.s.-terminating programs:

Theorem 8. *(Simple patterns for finite-state PIPs)*
Let P be a finite, deterministic, and a.s.-terminating PIP. Then the algorithm finds a shortest word w such that the pattern $C^*wC^*w\dots$ is terminating and s_0 is a prefix of w, thus proving termination of P. □

Proof.
Recall from the proof of Theorem 7 (3) that there is a fixed word $z \in C^*$ which leads from an arbitrary state in $\text{Runs}^{\mathcal{M}_P}$ to termination. In particular, z is never an infix of u_i^ω for any i. It follows that s_0z is never an infix of u_i^ω for any i. Assume for a contradiction that our algorithm does not succeed in proving termination. Since the s_i are all pairwise different, our algorithm eventually chooses $s_j := s_0z$ for some $j \in \mathbb{N}$ (note that the algorithm always chooses a *shortest* spoiler). By the definition of z the pattern $C^*zC^*z\dots$ is terminating, and then also $C^*s_jC^*s_j\dots$ is terminating. It follows that the pattern checker reports $C^*s_jC^*s_j\dots$ terminating, which is a contradiction. For proving that w is the *shortest* word such that $C^*wC^*\dots$ is terminating and s_0 is a prefix of w assume the algorithm terminates after k iterations. For every word v such that $C^*vC^*v\dots$ is terminating and s_0 is a prefix of v, it holds that v is never an infix of u_i^ω for any $1 \leq i \leq k$. The result w computed by the algorithm is a *shortest word* also enjoying this property, since we always choose a shortest spoiler. It follows that w has minimal length. □

In each iteration the algorithm picks a word s_j that destroys all previously discovered lasso loops. If the loops are small, then the word is short:

Lemma 4. *We have $|s_j| \leq |s_0| + 1 + \log_2(|u_1| + \dots + |u_j|)$.*

```
int k = 1;
while (0 < k < N) {
  if (coin(p)) k++ else k--
}
end.
```

```
int K = 2; int c1 = ?; int c2 = K;
int k = 1;
while (0 < k < N) {
    if (c1 > 0) {
      if nondet() k++ else k--; c1--
    }
    elseif (c2 > 0) { k--; c2-- }
    else { K++; c1 = ?; c2 = K }
}
end.
```

Figure 5.4: The PIP `RW` and the program `RW'`.

Proof. If a word w is not an infix of any of the words $u_1^\omega, \ldots, u_j^\omega$, then neither is $s_0 w$. Hence it suffices to construct such a word w with $|w| \leq 1 + \log_2 K$, where $K := |u_1| + \cdots + |u_j|$. Let $p_1, \ldots, p_K$ be an enumeration of all suffixes of the words $u_1^\omega, \ldots, u_j^\omega$. For any word w, we denote by $S(w) \subseteq \{p_1, \ldots, p_K\}$ the set of words $p \in \{p_1, \ldots, p_K\}$ such that w is a prefix of p. It suffices to construct w such that $|w| \leq 1 + \log_2 K$ and $S(w) = \emptyset$. We construct w iteratively. Let $w_0 := \epsilon$. In each iteration i, choose $w_{i+1} := w_i c$ with $c \in \{0, 1\}$ so that $|S(w_i c)|$ is minimized. Observe that $|S(w_{i+1})| \leq |S(w_i)|/2$, as all words in $S(w_i)$ start with either $w_i 0$ or $w_i 1$. It follows that $S\left(w_{1+\lfloor \log_2 K \rfloor}\right) = \emptyset$. □

5.3.2 Weakly Finite Programs

We now address our main goal, namely proving a.s.-termination for weakly finite programs. We note that the problem is undecidable in general, since already proving sure termination for weakly finite programs without probabilistic choices is undecidable (see e.g. Apt and Kozen [2]). Therefore every approach that attempts to prove termination has to be incomplete.

For weakly finite a.s.-terminating programs there might not be a *simple* terminating pattern. Consider the random walk program `RW` on the left of Fig. 5.4, where N is an input variable. `RW` terminates a.s., but we can easily show (by setting N to a large enough value) that no simple pattern is terminating. However, there *is* a terminating pattern, namely $\Phi = C^*00C^*000C^*0000\ldots$: every Φ-conforming run terminates, whatever value N is set to. Since, by Theorem 7 (1),

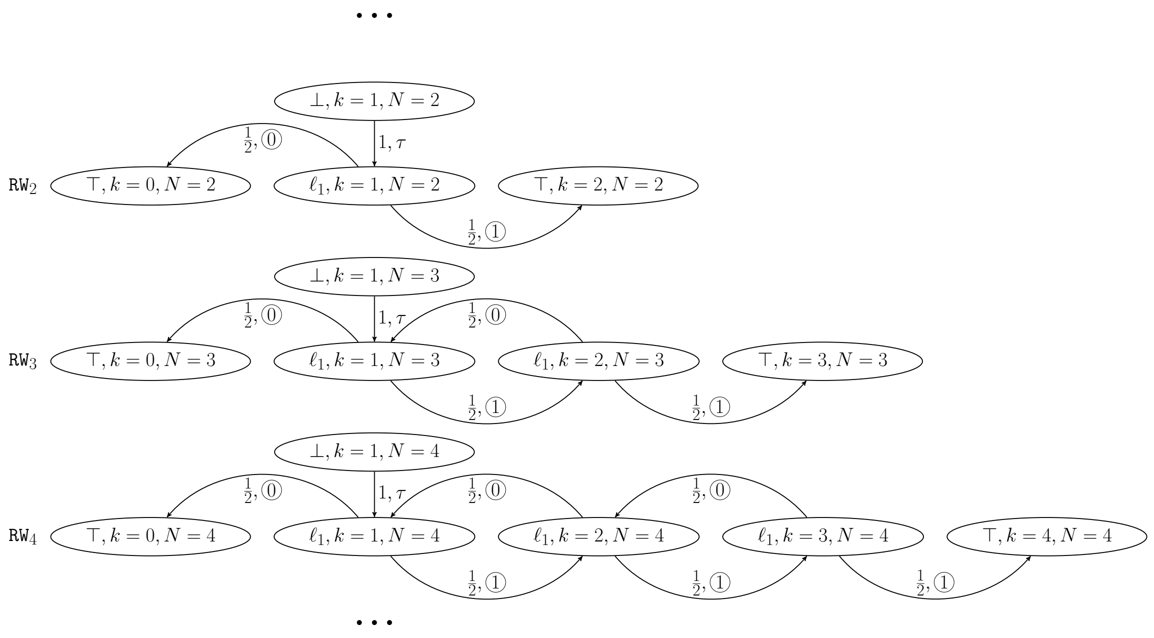


Figure 5.5: MDP associated to RW: the states belonging to RW$_2$, RW$_3$ and RW$_4$ are shown. The state $\top$ and transitions to $\top$ are omitted for clarity.

the Φ-conforming runs have probability 1 (intuitively, when tossing a coin we will eventually see longer and longer chains of 0's), RW terminates a.s.
We propose a technique that uses the procedure for finite-state programs as a building block, and extends it with an extrapolation step in order to produce a candidate for a terminating pattern.

An example. We again first sketch the procedure, this time using RW as an example. The MDP corresponding to RW is shown in Fig. 5.5. Let RW$_i$ be the program RW with $N = i$. Since every RW$_i$ is finite-state, we can find terminating patterns $\Phi_i = (C^* u_i)^\omega$ for a finite set of values of i, say for $i = 1, 2, 3, 4, 5$. We obtain $u_1 = u_2 = \epsilon$, $u_3 = 00$, $u_4 = 000$, $u_5 = 0000$. As we will see later, we choose u_i such that Φ_i is not only terminating for RW$_i$, but also for every RW$_j$ with $j \le i$. This suggests to extrapolate and take the pattern $\Phi = C^*00C^*000C^*0000C^*00000\ldots$ as a candidate for a terminating pattern of RW. We automatically construct the nondeterministic program RW' on the right of Fig. 5.4 (see Section 5.4 for details). The theorem prover ARMC proves that RW' terminates, and so that RW terminates almost surely.

The algorithm. We give a more detailed description of our algorithm. Let P be an a.s.-terminating and weakly finite PIP. Let $b_1, b_2, \ldots$ be an enumeration of the set I of initial states. Our algorithm first fixes b_1 as the only initial state. This leads to a finite program, so we can run the previously described algorithm for finite programs, yielding a word w_1 such that $C^* w_1 C^* w_1 \ldots$ is terminating for the initial state b_1. Next our algorithm fixes b_2 as the only initial state, and runs the previously described algorithm taking w_1 as base word. As before, this establishes a terminating pattern $C^* w_2 C^* w_2 \ldots$. By construction of w_2, the word w_1 is a prefix of w_2, so the pattern $C^* w_2 C^* w_2 C^* w_2 \ldots$ is terminating for the initial states $\{b_1, b_2\}$. Continuing in this way we obtain a sequence $w_1, w_2, \ldots$ such that $C^* w_1 C^* w_2 \ldots$ is terminating. Our algorithm may not terminate, because it may keep computing $w_1, w_2, \ldots$. However, as we saw in the `RW`-example that it is promising to compute the first few w_i and then *guess* an expression for general w_i. For instance if $w_1 = 0$ and $w_2 = 00$, then one may guess $w_i = 0^i$. We represent the guessed sequence $w_1, w_2, \ldots$ in a finite way. For example, in our experiments the sequences can often be described by $w_i = w^{i+c}$ for a $c \in \mathbb{N}$ and $w \in \{0,1\}^*$. We pass the obtained pattern $C^* w_1 C^* w_2 \ldots$ to a pattern checker, which then may show the pattern terminating, establishing a.s.-termination of the weakly finite program P.

5.4 Implementing Pattern Checkers

In this section we give technical details about constructing pattern checkers both for checking finite and general weakly finite programs. We use existing tools and techniques like finite-state model checkers for temporal properties and termination provers in our tool chain.

Finite Programs. We describe how to build a pattern checker for a finite program P and a pattern of the form $C^* w C^* w \ldots$ (i.e. a simple pattern). We employ a model checker for finite-state nonprobabilistic programs that can verify temporal properties: given as input a finite program and a Büchi automaton $\mathcal{A}$, the model checker returns a lasso if there is a program run accepted by $\mathcal{A}$

(such runs are called "counterexamples" in classical terminology). Otherwise it states that there is no counterexample. For our case studies, we use the SPIN tool (Holzmann [56]).

Given a finite probabilistic program P and a pattern $\Phi = C^*wC^*w\ldots$, we first transform P into a nonprobabilistic program P' as follows. We introduce two fresh variables c and term, with ranges $\{0,1,2\}$ and $\{0,1\}$, respectively, and add assignments `term = 0` and `term = 1` at the beginning and end of the program, respectively. Then every location ℓ of P with $label(\ell,\ell') = (x := \text{coin}(p))$ for a label ℓ' is replaced by a nondeterministic choice and an if-statement as follows:

```
x = nondet();
if (x = 0) { c = 0; c = 2 }
else { c = 1; c = 2 }
```

In this way we can distinguish coin toss outcomes in a program trace by inspecting the assignments to c. Now we perform two checks on the nonprobabilistic program P': first we use SPIN to translate the LTL formula $G\,\neg\text{term} \wedge FG(c \notin \{0,1\})$ into a Büchi automaton and check whether P' has a run that satisfies this formula. If there is indeed such a lasso, our pattern checker reports it. Observe that by the construction of the LTL formula the lasso encodes a nonterminating run in P that eventually stops visiting probabilistic locations. So the lasso loop does not contain any coin tosses (and our algorithm will later correctly report that P is not a.s.-terminating). Otherwise, i.e. if no run satisfies the formula, we know that all nonterminating runs involve infinitely many coin tosses. Then we perform a second query: we construct a Büchi automaton $\mathcal{A}(w)$ that represents the set of infinite Φ-conforming runs, see Fig. 5.6. We use SPIN to check whether P' has a run that is accepted by $\mathcal{A}(w)$. If yes, then there is an infinite Φ-conforming run, and our pattern checker reports the lasso. Otherwise, it reports that Φ is a terminating pattern.

Weakly Finite Programs. Recall that for weakly finite programs, the pattern checker needs to handle patterns of a more general form, namely $\Phi = C^*w_1C^*w_2\ldots$. Even easy patterns like $C^*0C^*00C^*000\ldots$ cannot be represented by a finite Büchi automaton. Therefore we need a more involved instrumentation of the program to restrict its runs to Φ-conforming ones. Now our pattern checker

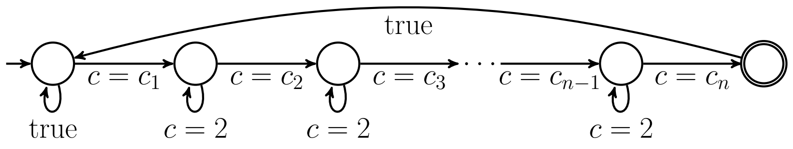


Figure 5.6: Büchi automaton $\mathcal{A}(w)$, for $w = c_1 c_2 \ldots c_n \in C^*$. Note that the number of states in $\mathcal{A}(w)$ grows linearly in $|w|$.

employs a termination checker for infinite-state programs. For our experiments we use ARMC (Podelski and Rybalchenko [85]).

Given a weakly finite program P and a pattern $\Phi = C^* w_1 C^* w_2 \ldots$, we transform P into a nonprobabilistic program P^Φ as follows. We use a command `x = ?`, which nondeterministically assigns a nonnegative integer to x. Further we assume that we can access the k-th letter of the i-th element of $(w_i)_{i \in \mathbb{N}}$ by `w[i][k]` and $|w_i|$ by `length(w[i])`. For example, if $w_i = w^{i+c}$ for a $c \in \mathbb{N}$ and $w \in \{0,1\}^*$, then `w[i][k]` is the $((k\ mod\ |w|) + 1)$-th letter of w and `length(w[i])` equals $|w| \cdot (i + c)$. We add fresh variables `ctr`, `next` and `pos`, where `ctr` is initialized nondeterministically with an arbitrary nonnegative integer and `next` and `pos` are both initialized with 1. If a run r is Φ-conforming, $\bar{r}|_C$ is a prefix of $v_1 w_1 v_2 w_2 v_3 w_3 \ldots$, with $v_i \in C^*$. The variable `ctr` is used to "guess" the length of the words v_i; the individual elements in v_i are irrelevant. We replace every command $x := \text{coin}(p)$ by the code sequence given in Fig. 5.7.

The runs in the resulting program P^Φ correspond exactly to the Φ-conforming runs in P. Then P^Φ is given to the termination checker. If it proves termination, we report "Φ is a terminating pattern for P". Otherwise, the tool might either return a lasso, which our pattern checker reports, or give up on P^Φ, in which case our pattern checker also has to give up.

In our experiments, a weakly finite program typically has an uninitialized integer variable N whose value is nondeterministically fixed in the beginning. The pattern $C^* w_1 C^* \ldots C^* w_N C^\omega$ is then often terminating, which makes next $\leq N$ an invariant in P^Φ. The termination checker ARMC may benefit from this invariant, but may not be able to find it automatically. We therefore enhanced ARMC to "help itself" by adding the invariant next $\leq N$ to the program if ARMC's reachability mode can verify the invariant.

```
x = nondet();
if (ctr <= 0) {
  if (pos > length(w[next])) { ctr = ?; pos = 1; next = next+1 }
  else { x = w[next][pos]; pos = pos+1 }
}
else ctr = ctr-1
```

Figure 5.7: Code transformation for coin tosses in weakly finite programs.

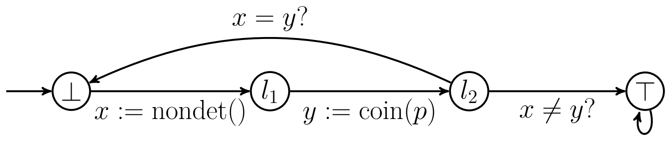


Figure 5.8: Nondeterministic a.s.-terminating program without terminating pattern.

5.5 Nondeterministic Programs

In the previous sections we have described how to use patterns for proving a.s.-termination for *deterministic* PIPs. We now show how to extend our notion of patterns to obtain a complete method for finite and weakly finite *nondeterministic* PIPs.

For nondeterministic a.s.-terminating programs, there might not exist a terminating pattern, even if the program is finite. Figure 5.8 shows an example.

Let Φ be a pattern and $c_1c_2c_3\ldots \in \Phi$. The run

$$\langle \bot, \sigma_0\rangle \xrightarrow{a_{c_1}} \langle l_1, \sigma_1\rangle \xrightarrow{c_1} \langle l_2, \sigma_1'\rangle \xrightarrow{\tau} \langle \bot, \sigma_1'\rangle \xrightarrow{a_{c_2}} \langle l_1, \sigma_2\rangle \xrightarrow{c_2} \langle l_2, \sigma_2'\rangle \xrightarrow{\tau} \langle \bot, \sigma_2'\rangle \xrightarrow{a_{c_3}} \ldots$$

in $\mathcal{M}_P$ is Φ-conforming but nonterminating.

We therefore propose another pattern class that also takes nondeterministic decisions into account. We fix an arbitrary PIP $P = (\mathcal{L}, I, \hookrightarrow, \text{label}, \bot, \top)$, and its associated MDP $\mathcal{M}_P = (Q_A, Q_P, \text{Init}, \to, \text{Lab}_A, \text{Lab}_P)$. We write $A := \{a_0, a_1\}$ and $G := \{a_0, a_1\} \cup C$.

We assume that P is in a special *normal form*: every nondeterministic location has a probabilistic location as its successor, and every probabilistic location has a nondeterministic location as its successor. It is easy to transform a PIP into normal form by adding redundant probabilistic and nondeterministic locations such that the transformed program terminates iff the original one does. For example, the PIP in Fig. 5.8 is in normal form. If P is in normal form, then every run $\bar{r} \in \mathcal{M}_P$ satisfies $r|_G \in (AC)^\infty$.

A set $W \subseteq (AC)^*$ is called a *response* of length $n \geq 0$ if (i) every $w \in W$ has length $2n$, (ii) for $w_1, w_2 \in W$ with $w_1 \neq w_2$, $w_1|_A \neq w_2|_A$ holds, and (iii) W contains exactly 2^n elements. We denote by $\mathit{Resp}(n)$ the set of responses of length n, and set $\mathit{Resp} := \bigcup_{n \in \mathbb{N}} \mathit{Resp}(n)$. Intuitively, a response R of length n contains for every sequence of nondeterministic actions of length n a sequence of coin toss outcomes of length n (interleaved in one word of R). For example, $\{a_0c_1, a_1c_0\}$ is a response of length one, $\{a_0c_0a_0c_1, a_0c_0a_1c_1, a_1c_0a_0c_1, a_1c_0a_1c_1\}$ is a response of length two. We can now define *response patterns*:

Definition 38. (Response pattern)
A *response pattern* is a subset of $(AC)^\omega$ of the form $(AC)^*R_1(AC)^*R_2(AC)^* \ldots$, where $R_1, R_2, \ldots$ are responses. We say $R_1, R_2, \ldots$ induces the response pattern. As in the deterministic case, if $R_1, R_2, \ldots$ is an enumeration of all responses, we call the pattern $(AC)^*R_1(AC)^*R_2(AC)^*$ a *universal response pattern*, and a pattern of the form $((AC)^*R_1)^\omega$ a *simple response pattern.* $\square$

For a response pattern Φ, a run $r \in \mathrm{Runs}(\mathcal{M}_P)$ is Φ-conforming if there is $v \in \Phi$ such that $\bar{r}|_G$ is a prefix of v. We call a response pattern Φ *terminating* if all Φ-conforming runs terminate.

Analogously to the deterministic case, we can show the following theorem. Its proof is similar to the one of Theorem 7 and is given in Appendix B:

Theorem 9. *(Response pattern as a proof method)*
Let P be a PIP in normal form.

(1) Let Φ be a response pattern. The set of Φ-conforming runs has probability 1 for every strategy S for $\mathcal{M}_P$. In particular, if P has a terminating response pattern, then P is a.s.-terminating.

(3) If P is a.s.-terminating and finite with $n < \infty$ reachable states in $\mathcal{M}_P$, then there exists a response R of length in $\mathcal{O}(n^2)$ such that $(AC)^*R(AC)^\omega$ is terminating for P and $((AC)^*R)^\omega$ is a simple terminating response pattern for P.

(2) If P is a.s.-terminating and weakly finite, then the universal response pattern is terminating for P.

□

For the program in Fig. 5.8, a terminating response pattern is

$$\Phi = (AC)^*\{a_0c_1, a_1c_0\}(AC)^\omega.$$

Every Φ-conforming run has the form

$$\langle \bot, \sigma_0 \rangle \to \ldots \to q \xrightarrow{a_i} q' \xrightarrow{c_{1-i}} q'' \to \top \to \ldots$$

for an $i \in \{0, 1\}$.

5.6 Experimental Evaluation

We apply our methods to several parameterized programs taken from the literature.[1]

- *firewire:* A fragment of FireWire's symmetry-breaking protocol, adapted from McIver et al. [73] (we used a simplified version as example program `FW`). Roughly speaking, the number 100 of Fig. 5.2 is replaced by a parameter N.
- *randomwalk:* A slightly different version of the finite-range, one-dimensional random walk example `RW`.
- *herman:* An abstraction of Herman's randomized algorithm for leader election used in Nakata [80]. It can be seen as a more complicated finite random walk, with N as the walk's length.
- *zeroconf:* A model of the Zeroconf protocol taken from Kattenbelt et al. [63] that we also analyzed in Section 4.8. The protocol assigns IP addresses in a network. The parameter N is the number of probes sent after choosing an IP address to check whether it is already in use.

[1]The sources can be found at `http://www.model.in.tum.de/~gaiser/cav2012.html`.

Name	#loc	Pattern words for $N = 1, 2, 3, 4$				Time (SPIN)	i-th word of guessed pattern	Time (ARMC)
firewire	19	010	010	010	010	17 sec	010	1 min 36 sec
randomwalk	16	ϵ	0^2	0^3	0^4	23 sec	0^i	1 min 22 sec
herman	36	010	$0(10)^2$	$0(10)^3$	$0(10)^4$	47 sec	$0(10)^i$	7 min 43 sec
zeroconf	39	0^3	0^4	0^5	0^6	20 sec	0^{i+2}	26 min 16 sec
brp	57	00	00	00	00	19 sec	00	45 min 14 sec

Figure 5.9: Constructed patterns of the case studies and runtimes.

- *brp:* A model adapted from Kattenbelt et al. [63] that models the well-known bounded retransmission protocol. The original version can be proven a.s.-terminating with the trivial pattern C^ω; hence we study an "unbounded" version, where arbitrarily many retransmissions are allowed. The parameter N is the length of the message that the sender must transmit to the receiver.

Proving a.s.-termination. We prove a.s.-termination of the examples using SPIN (Holzmann [56]) to find patterns of finite-state instances, and ARMC (Podelski and Rybalchenko [85]) to prove termination of the nondeterministic programs derived from the guessed pattern. All experiments were performed on an Intel© i7 machine with 8 GB RAM. The results are shown in Fig. 5.9. The first two columns give the name of the example and its size. The next two columns show the words $w_1, \ldots, w_4$ of the terminating patterns $C^* w_1 C^\omega, \ldots, C^* w_4 C^\omega$ computed for $N = 1, 2, 3, 4$ (see Theorem 7 (3) and Section 5.4), and SPIN's runtime. The last two columns give word w_i in the guessed pattern $C^* w_1 C^* w_2 C^* w_3 \ldots$ (see Section 5.4), and ARMC's runtime. For instance, the entry $0(10)^i$ for *herman* indicates that the guessed pattern is $C^*010C^*01010C^*0101010\ldots$.
We derive two conclusions. First, a.s.-termination is proved by very simple patterns: the general shape is easily guessed from patterns for $N = 1, 2, 3, 4$, and the need for human ingenuity is virtually reduced to zero. This speaks in favor of the Planner technique of Arons et al. [3] and our extension to patterns, compared to other approaches using fairness and Hoare calculus (McIver and Morgan [72], Pnueli and Zuck [84]). Second, the runtime is dominated by the termination tool, not by the finite-state checker. So the most direct way to improve the efficiency of our technique is to produce faster termination checkers.

5.7 Termination Information and Reachability Probabilities

In the beginning of this chapter we claimed that general purpose probabilistic model checkers perform poorly for a.s.-termination, since they are not geared towards this problem. To supply some evidence for this, we tried to prove a.s.-termination of the first four examples from our experiments (*firewire, randomwalk, herman, zeroconf*) using the known PASS tool (Hahn et al. [52]) that we already used for comparisons in Chapter 4. In all four cases the refinement loop did not terminate.[1] Essentially, the tool unfolds the concrete system for specific single initial states during the refinement, but is never able to prove a.s.-termination for all initial states.

We sketch the underlying problems of model checkers that use abstraction techniques with the help of an program example P:

```
int x = N;
while (x > 0) {
  if (coin(0.5)) x--
}
end.
```

We want to prove that P is a.s.-terminating using a game-based abstraction approach. We use our notation from the previous chapter, where we constructed game arenas for obtaining extremal reachability values. We transform P into a PGP:

```
int x = N;
a: (x > 0) -> 0.5: x' = x-1 + 0.5: x' = x
reach: !(x>0)
```

P is almost surely terminating. Let $\mathcal{G}$ be an abstract game arena of P. First recall that proving a.s.-termination is equivalent to showing that $\mathit{MinReach}(\mathcal{M}_P, F) = 1$, for F the set of states enabling $\neg(x > 0)$. For $\mathcal{G}$ therefore

$$\min^-(\mathcal{G}, \{\circledcirc, \otimes\}) = \min^+(\mathcal{G}, \{\circledcirc, \otimes\}) = 1$$

[1]Other checkers, like PRISM, cannot be applied because they only work for finite-state systems.

has to hold. We sketch in the following that every $\mathcal{G}$ for which Eq. 5.7 holds is large: it contains at least $N+1$ states belonging to Player 1 (i.e. it has at least as many Player 1 states as $\mathcal{M}_P$ has action states).
Let us assume for the sake of contradiction that $\mathcal{G}$ has less than $N+1$ Player 1 states in $\mathcal{G}$ and satisfies Eq. 5.7. Let γ be the concretization function of the abstract domain used for building $\mathcal{G}$. Let us further assume for simplicity that the abstract domain is a predicate domain, i.e. for every concrete program state σ reachable in $\mathcal{M}_P$ there exists exactly one Player 1 state in s such that $\sigma \in \gamma(s)$.[1]
Since for every reachable state σ of the program there has to be one abstract state s in the abstraction with $\sigma \in \gamma(s)$, there is a (reachable) Player 1 state s in $\mathcal{G}$ with $\{\sigma_1, \sigma_2\} \subseteq \gamma(s)$ such that $0 \leq \sigma_1(x) < \sigma_2(x)$, with σ_1, σ_2 two reachable action states in $\mathcal{M}_P$. Since $\sigma_2(x) > 0$, Player 1 can choose a from s, and Player 2 can approve and choose a state $\langle s, a, d_1\rangle$ such that

$$\langle s, a, d_1\rangle \xrightarrow{0.5,1} s_1$$

with $[\![c_a(1)]\!](\sigma_2) \in \gamma(s_1)$ and

$$\langle s, a, d_1\rangle \xrightarrow{0.5,2} s$$

(recall that we use a predicate domain, and $[\![c_a(2)]\!](\sigma_2) = \sigma_2 \in \gamma(s)$). After choosing $\langle s, a, d_1\rangle$ the game is either again in state s, or in an abstract state s_1 that contains the concrete state $[\![c_a(1)]\!](\sigma_2)$. If $[\![c_a(1)]\!](\sigma_2)(x) = 0$, then $\sigma_1(x) = 0$ since $0 \leq \sigma_1(x) < \sigma_2(x)$ holds, and $s = s_1$ holds; in this case the play can stay in $s, \langle s, a\rangle$, and $\langle s, a, d\rangle$ forever, and so never reaches $\circledcirc$ or $\otimes$. If $[\![c_a(2)]\!](\sigma_2)(x) > 0$, Player 1 can choose a in s_1 again, and Player 2 can approve by choosing a state d_2 such that $\langle s_1, a, d_2\rangle \xrightarrow{0.5,1} s_2$ and $\langle s_1, a, d_2\rangle \xrightarrow{0.5,2} s_1$ where $[\![c_a(1)]\!]([\![c_a(1)]\!](\sigma_2)) \in \gamma(s_2)$, i.e. we either stay in s_1 or reach an abstract state containing a concrete state whose x-value is again decreased. If we continue in this manner, we eventually reach a state s_k with $\sigma_1 \in \gamma(s_k)$, and then $s_k = s$ holds. The players can again play according to the rules from above, resulting in a cycle: the game stays in $s, \langle s, a\rangle, \langle s, a, d_1\rangle, s_1, \langle s_1, a\rangle, \langle s_1, a, d_2\rangle, \ldots, \langle s_{k-1}, a, d_{k-1}\rangle$

[1] the following argument applies also for arbitrary domains and overlapping abstract states, however it is less tedious to use a domain that partitions the state space for showing the claim.

forever. Hence $\min^-(\mathcal{G}, \{\circledcirc, \otimes\}) < 1$ has to hold.

Essentially the same problem can arise in all variants of game-based approaches developed so far (Kattenbelt et al. [62, 63], Wachter and Zhang [99]). In particular, if we want to show termination for the weakly finite version of P, i.e. if we replace `int x = N` by `int x = ?`, no (finite) game-based abstraction is sufficient for proving a.s.-termination.[1]

5.7.1 Improving Lower Bounds for Reachability

Consider now a program of the form

```
if coin(0.8) {P1()}
else {P2()}
ERROR
end.
```

We are interested in the probability of reaching `ERROR`. We saw in the previous section that game-based abstraction approaches have trouble computing lower bounds for extremal reachability values: in the example they work with abstractions of P1 and P2, and so they may not be able to ascertain that paths of the abstraction are concrete paths of the program, leading to poor lower bounds. Information on a.s.-termination helps: if e.g. P1 terminates a.s., then we already have a lower bound of 0.8. We demonstrate this technique on two examples. The first one is the following modification of *firewire*:

```
int N = 1000; int k = 0; int miss = 0;
while (k < N) {
  oldx = x; x = coin(0.5);
  if (x = oldx) k++
  elseif (k < 5) miss = 1
}
end.
```

[1]In the previous chapter we did not use more than one concrete initial state for P; this restriction however is not essential and can easily be removed by using more than one initial abstract state in the abstract game arenas.

For $i \in \{0, 1\}$, let p_i be the probability that the program terminates with miss $= i$. After 20 refinement steps PASS returns upper bounds of 0.032 for p_0 and 0.969 for p_1, but a lower bound of 0 for p_1, which stays 0 after 300 iterations. Our algorithm establishes that the loop a.s.-terminates, which implies $p_0 + p_1 = 1$, and so after 20 iterations we already get $0.968 \leq p_1 \leq 0.969$. We apply the same technique to estimate the probabilities p_1, p_0 that *zeroconf* detects/does-not-detect an unused IP address. For $N = 100$, after 20 refinement steps PASS reports an upper bound of 0.999 for p_0, but a lower bound of 0 for p_1, which stays 0 for 80 more iterations. With our technique after 20 iterations we get $0.958 \leq p_1 \leq 0.999$.

5.8 Related Work

Almost-sure termination is highly desirable for protocols if termination within a fixed number of steps is not feasible. For instance, Bracha and Toueg [16] considers the problem of reaching consensus within a set of interconnected processes, some of which may be faulty or even malicious. They succeed in designing a probabilistic protocol to reach consensus a.s., although it is known that no deterministic algorithm terminates within a bounded number of steps. Therefore the problem of proving probabilistic programs a.s.-terminating has been tackled quite early in the research community. A well-known approach for proving a.s.-termination are Pnueli et al.'s notions of extreme fairness and α-fairness (Pnueli [83], Pnueli and Zuck [84]). These proof methods, although complete for finite-state systems, are hard to automatize and require a lot of knowledge about the considered program. The same applies for the approach of McIver et al. in McIver and Morgan [72] that offers proof rules for probabilistic loops in pGCL, an extension of Dijkstra's guarded language. The paper Monniaux [78] discusses probabilistic termination in an abstraction-interpretation framework. It focuses on programs with a (single) loop and proposes a method of proving that the probability of taking the loop k times decreases exponentially with k. This implies a.s.-termination. In contrast to our work there is no tool support in Monniaux [78].

5.9 Conclusion

In this chapter we have presented an approach for automatically proving a.s.-termination of probabilistic programs. Inspired by the Planner approach of Arons et al. [3], we instrument a probabilistic program P into a nondeterministic program P' such that the runs of P' correspond to a set of runs of P with probability 1. The instrumentation is fully automatic for finite-state programs, and requires an extrapolation step for weakly finite programs. We automatically check termination of P' profiting from new tools that were not available to Arons et al. [3]. While our approach maintains the intuitive appeal of the Planner approach, it allows to prove completeness results. Furthermore, while in Arons et al. [3] the design of the Planner was left to the verifier, we have provided a CEGAR-like approach. In the case of parameterized programs, the approach requires an extrapolation step, which however in our case studies proved to be straightforward. Finally, we have also shown that our approach improves the game-based abstraction techniques of Kattenbelt et al. [62], Wachter and Zhang [99] and our method from Chapter 4 for computing upper and lower bounds for the probability of reaching a program location. While this technique often provides good upper bounds, the lower bounds are not so satisfactory (often 0), due to spurious non-terminating runs introduced by the abstraction. Our approach allows to remove the effect of these runs.

Chapter 6

Extinction in Branching Processes

In the previous chapters we have studied reachability and termination properties of probabilistic programs. In Chapter 4 we have developed a method for computing bounds of extremal reachability values; in Chapter 5 we have given a proof method for showing that a program is almost-surely terminating. For the latter we restricted ourselves to weakly finite programs, since for them almost-sure termination is a topological property of their corresponding MDP. This allowed us to neglect the actual values of probabilities occuring in the programs. For other kind of probabilistic programs, however, we argued that more elaborate techniques are necessary, and used the example program

```
int k = 1;
l1: while (0 < k) { if (coin(p)) k++ else k-- }
end.
```

as an illustration: in this example almost-sure termination depends on the concrete value of `p`.

In this part of the thesis we study *multi-type finite branching processes* (MFBPs for short), a stochastic model which is widely used in numerous scientific branches. More precisely, we develop algorithms for investigating *extinction probabilities* for MFBPs. Our methods enables us to prove almost-sure termination for a special class of simple recursive programs (besides solving problems in other areas). In particular we can represent the example program above within this program class.

An MFBP describes the development of a population of individuals. Each individual of a population has a *type* i from $\{1, \dots, n\}$ for a fixed n. It produces a random number of children of different types determined by a fixed probability distribution; the union of all offsprings of every individual then forms the next generation of the process. The distribution only depends on the type of the individual (there are no interactions between members of the current population). Examples of individuals include elementary particles, genes, animals, or program threads (Athreya and Ney [4], Harris [54]). It turns out that the probability of a given initial population becoming ultimately extinct is important for many applications of MFBPs.

We can represent an MFBP as an infinite-state Markov chain; its state space consists of populations. *Extinction* in an MFBP corresponds to reaching the empty population (without any individuals) in the corresponding Markov chain. We already studied classes of infinite-state Markov chains and MDPs (e.g. those corresponding to PGPs or weakly finite PIPs) for which computing reachability probabilities for infinite-state Markov chains is undecidable in general, therefore we relied on incomplete methods. It is even undecidable to give nontrivial bounds for probabilities in these chains In the case of Markov chains corresponding to MFBPs, however, given an initial population, we can give efficient algorithms for (i) deciding whether the probability of reaching the empty population is one and (ii) computing bounds for the probability of reaching the empty population. We show that the latter is even possible in strongly-polynomial time.

It turns out that the *extinction probabilities* of an MFBP play an important role for both problems (i) and (ii). For every type i of an MFBP, the extinction probability ψ_i denotes the probability that a population consisting of a *single* individual of type i eventually goes extinct. We group them together into a vector $\psi = (\psi_1, \dots, \psi_n)^\top$. Given ψ we can easily compute the probability of an *arbitrary* given population going ultimately extinct, so we will focus on investigating properties of ψ in the following.

MFBPs and in particular the study of their extinction probabilities appear in many different scientific areas, often with different names: in computer science MFBPs are used for the study of reachability probabilities and almost-sure termination in computation models like *stateless probabilistic pushdown automata*

(Esparza et al. [38]) and *1-exit recursive Markov chains* (Etessami and Yannakakis [43]). The *consistency problem* of stochastic context-free grammars, an important question in statistical natural language processing (Manning and Schuetze [70]), boils down to solving (i).

MFBPs are also important in biology, physics and chemistry, e.g. for modeling the growth of animal populations (and their possible extinction). In Section 6.5 we give an example of how extinction probabilities can be used as a model to assess the probability of nuclear chain reactions.

Extinction probabilities can be expressed as solutions of an equation system of the form

$$X_1 = f_1(X_1, \ldots, X_n) \quad \ldots \quad X_n = f_n(X_1, \ldots, X_n)\ ,$$

where, for every $i \in \{1, \ldots, n\}$, f_i is a polynomial over $X_1, \ldots, X_n$ with positive rational coefficients that *add up to 1*. The solutions are the fixed points of the function $f : \mathbb{R}^n \to \mathbb{R}^n$ with $f = (f_1, \ldots, f_n)$. By Kleene's theorem (2), every such f has a least nonnegative fixed point, μ_f, given by the limit of the sequence $\overline{0}, f(\overline{0}), f(f(\overline{0})), \ldots$. A fundamental result of the theory of MFBPs (see e.g. Athreya and Ney [4], Harris [54]) states that extinction probabilities of species are equal to μ_f, i.e. $\psi = \mu_f$. However, extinction probabilities cannot be represented by radicals in general (see Etessami and Yannakakis [43]), so we can only approximate them, e.g. by giving lower and upper bounds.

We develop efficient but nevertheless precise algorithms for the following two problems: given a PSP f with least nonnegative fixed point μ_f,

(A) decide whether $\mu_f = \overline{1}$, and

(B) given a rational number $\epsilon > 0$, compute $\mathbf{lb}, \mathbf{ub} \in \mathbb{Q}^n$ such that $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ and $\mathbf{ub} - \mathbf{lb} \leq \overline{\epsilon}$ (where $\mathbf{u} \leq \mathbf{v}$ for vectors $\mathbf{u}, \mathbf{v}$ means $\leq$ in all components).

Problem (B) immediately allows us to obtain bounds for extinction probabilities of an MFBP with a given accuracy. Besides the fact that Problem (A) can be used to decide whether a one-exit recursive Markov chain or a stateless probabilistic pushdown automaton is almost-surely terminating (see also Esparza et al. [38], Etessami and Yannakakis [43]), we give another motivation in the case study of Section 6.5: there we consider a family of PSPs, taken from Harris [54], modelling the neutron branching process in a ball of radioactive material of radius D (the family is parameterized by D). The least fixed point is the probability that a neutron produced through spontaneous fission *does not* generate an infinite "progeny" through successive collisions with atoms of the ball; loosely speaking, this is the probability that the neutron *does not* generate a chain reaction and the ball *does not* explode. Since the number of atoms in the ball is very large, spontaneous fission produces many neutrons per second, and so even if the probability that a given neutron produces a chain reaction is very small, the ball will explode with large probability in a very short time. It is therefore important to determine the largest radius D at which the probability of no chain reaction is still 1 (usually called the *critical radius*). An algorithm for Problem (B) allows to compute the critical radius using binary search.

Etessami and Yannakakis [43] show that Problem (A) can be solved in polynomial time by a reduction to (exact) Linear Programming (LP), which is not known to be strongly polynomial (see Grötschel et al. [47]). Our solution reduces Problem (A) essentially to the problem of solving *two* systems of linear equations, resulting in a *strongly polynomial* algorithm for Problem (A). The Maple computer algebra system (*Maple* [71]) offers exact arithmetic LP solvers and exact methods for solving systems of linear equations, which we use to test the performance of our new algorithm. We also do comparisons with a standalone exact LP solver. In the neutron branching process discussed above we obtain speed-ups of about one order of magnitude with respect to LP.

Our second result is, to the best of our knowledge, the first practical algorithm for Problem (B). Lower bounds for μ_f can be computed using Newton's method for approximating a root of the function $f(\overline{X}) - \overline{X}$. This has recently been investigated in detail (Esparza et al. [37], Etessami and Yannakakis [43], Kiefer et al. [64]). However, Newton's method faces considerable numerical problems. Experiments show that naive use of exact arithmetic is inefficient, while floating-point computation leads to false results even for very small systems. For instance, the PReMo tool described in Wojtczak and Etessami [102], which implements Newton's method with floating-point arithmetic for efficiency, reports $\mu_f \geq \overline{1}$ for a PSP with only 7 variables and small coefficients, although $\mu_f < \overline{1}$ is the case (see Section 6.3.3).

Our algorithm produces a sequence of guaranteed lower and upper bounds, both of which converge linearly to μ_f. Linear convergence means that, loosely speaking, the number of accurate bits of the bounds increases linearly with the number of sequence elements. The algorithm is based on the following idea. Newton's method is an iterative procedure that, given a current lower bound $\mathbf{lb}$ on μ_f, applies a certain operator $\mathcal{N}$ to it, yielding a new, more precise lower bound $\mathcal{N}(\mathbf{lb})$. Instead of computing $\mathcal{N}(\mathbf{lb})$ using exact arithmetic, our algorithm computes *two* consecutive Newton steps, i.e., $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$, using *inexact* arithmetic. Then it checks if the result satisfies a carefully chosen condition. If so, the result is taken as the next lower bound. If not, then the precision is increased, and the computation redone. The condition is eventually satisfied, assuming the results of computing with increased precision converge to the exact result. Usually, the repeated inexact computation is much faster than the exact one. At the same time, a careful analysis shows that the sequence of lower bounds converges linearly to μ_f.

Computing *upper* bounds is harder, and has not been considered in the literature before to the best of our knowledge. Similarly to the case of lower bounds, we apply f twice to $\mathbf{ub}$, i.e., we compute $f(f(\mathbf{ub}))$ with increasing precision until a condition holds. The sequence so obtained may not even converge to μ_f. So we need to introduce a further operation, after which we can then prove linear convergence.

We test our algorithm on the neutron branching process. The time needed to obtain lower and upper bounds on the probability of no explosion with $\epsilon = 0.0001$ lies below the time needed to check, using Maple's exact LP, whether this probability is 1 or smaller than one. That is, in this case study our algorithm is faster, and provides more information.

Structure of the chapter. We first give in Section 6.1 a more detailed introduction to MFBPs and show that extinction probabilities can be represented as solutions of PSPs. We then introduce briefly several concepts from computer science that are closely related to MFBPs. Important properties of PSPs are investigated in Section 6.2.2. We then present our algorithm for Problem (A) in Section 6.3 and show its advantages compared to existing approaches in the

case study of Section 6.3.3. Our algorithm for Problem (B) is presented in Section 6.4. The final case study in Section 6.5 is taken from nuclear physics and models a neutron branching process; we use it to show the efficiency of both our algorithms. We end with a short conclusion.
This chapter is mainly based on joint work with Javier Esparza and Stefan Kiefer; parts of the material have been published in Esparza et al. [35]. In particular, Stefan Kiefer first proved several propositions in Section 6.4.

6.1 Multi-Type Finite Branching Processes

For the following we fix $n \in \mathbb{N}$. A *population* is a vector $\mathbf{x} \in \mathbb{N}^n$, where $\mathbf{x}_i$ for $i \in \{1, \dots, n\}$ denotes the number of individuals of type i. We call $\mathbb{P} = \mathbb{N}^n$ the *population space*. A multi-type finite branching process (MFBP) models the development of populations for which certain assumptions apply:

- Populations are changing in discrete time steps.
- In each time step and for each individual of type i of a population, the following happens:
    1. The individual generates randomly a number of children of possibly different types (a population), according to a fixed probability distribution $\mathrm{Pr}_i : \mathbb{P} \to [0, 1]$. Pr_i does only depend on the type of the object, not on the actual time step or on other individuals in the population.
    2. The individual dies.

  The new population after one time step is the union of the populations generated by each individual of the old population.

We note that a line of descendants of an individual eventually goes extinct (i.e., no descendants are alive at a certain time step) if and only if all lines of its children go eventually extinct. The entire development of such a population system can be studied by giving the probabilities Pr_i for $i \in \{1, \dots, n\}$:

Definition 39. (Multi-type finite branching processes)
A sequence $\mathcal{B} = (\mathrm{Pr}_1, \mathrm{Pr}_2, \dots, \mathrm{Pr}_n)$ of distributions with $\mathrm{Pr}_i \in \mathrm{Dist}_F(\mathbb{P})$ for every

$i \in \{1, \ldots, n\}$ is called a *multitype finite branching process (MFBP) with n types* or an *n-type MFBP*. $\square$

We give a simple example from cell biology, which we have taken and slightly modified from Haccou et al. [51]. We study the development of a population of bacteriae. We distinguish between bacteriae with alleles A and B for a specific gene. A bacteria with allele A dies with probability 0.25 in a time step. Otherwise it divides and produces two offsprings. With probability 0.75 both offsprings exhibit allele A; with probability 0.25, one of the bacteriae is a mutation and possesses allele B (but never do both offsprings mutate). A bacteria with allele B has a higher risk of dying (0.99) during one time step; otherwise it divides into two offsprings, where with probability 0.5, one of the offsprings mutates and posseses allele A, otherwise both have allele B. The scenario is summarized in Fig. 6.1. It gives rise to a 2-type MFBP $\mathcal{B}$: let us assume type 1 corresponds to individuals with allele A, type 2 to those with allele B. Then we can define $\mathcal{B} = (\mathrm{Pr}_1, \mathrm{Pr}_2)$ by setting

$$\mathrm{Pr}_1(\overline{0}) = 0.25, \mathrm{Pr}_1((2,0)^\top) = 0.75 \cdot 0.75, \mathrm{Pr}_1((1,1)^\top) = 0.75 \cdot 0.25 \text{ and}$$
$$\mathrm{Pr}_2(\overline{0}) = 0.99, \mathrm{Pr}_2((0,2)^\top) = 0.01 \cdot 0.5, \mathrm{Pr}_2((1,1)^\top) = 0.01 \cdot 0.5.$$

We interpret MFBPs as Markov chains: the states are populations and transitions from a population $\mathbf{x}$ to a population $\mathbf{y}$ are labeled by the probability that, under the assumption that the current population is $\mathbf{x}$, in one step the system will reach the population $\mathbf{y}$. For stating this probability we use the assumption that reproduction of an individual is independent from other individuals or the concrete time step.

Definition 40. (Markov chain induced by an MFBP)
Let $\mathcal{B} = (\mathrm{Pr}_1, \mathrm{Pr}_2, \ldots, \mathrm{Pr}_n)$ be a MFBP with n types. We define the Markov chain

$$\mathcal{M}[\mathcal{B}] \;=\; (\mathbb{P},\; \rightarrow,\; \{\tau\}),$$

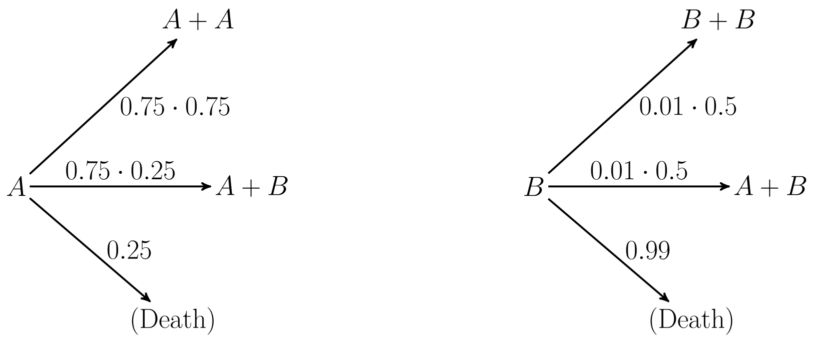


Figure 6.1: Example MFBP. Bacteria with different allele type multiply or die with different probabilities. Mutations can occur in the case of multiplication.

with the relation $\rightarrow$ defined as follows: we define for every $\mathbf{z} \in \mathbb{P}, k \in \mathbb{N}$ and $1 \leq i \leq n$

$$g(\mathbf{z}, i, k) := \sum_{\substack{(\mathbf{v^{(1)}},\ldots,\mathbf{v^{(k)}}) \in \mathbb{P}^k: \\ \mathbf{v^{(1)}}+\ldots+\mathbf{v^{(k)}}=\mathbf{z}}} Pr_i(\mathbf{v^{(1)}}) \cdot \ldots \cdot Pr_i(\mathbf{v^{(k)}}),$$

the probability that k individuals of type i generate the population $\mathbf{z}$ in one time step. For all $\mathbf{x} \in Q$ and $\mathbf{y} \in Q$ let

$$p(\mathbf{x}, \mathbf{y}) := \sum_{\substack{(\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}) \in \mathbb{P}^n: \\ \mathbf{w}^{(1)}+\ldots+\mathbf{w}^{(n)}=\mathbf{y}}} \left(\prod_{i=1}^{n} g(\mathbf{w}^{(i)}, i, \mathbf{x}_i) \right)$$

be the probability of reaching the population $\mathbf{y}$ in one time step from population $\mathbf{x}$. If $p(\mathbf{x}, \mathbf{y}) \neq 0$, then $\mathbf{x} \xrightarrow{p(\mathbf{x},\mathbf{y}),\tau} \mathbf{y}$ holds. □

Note that each transition is labeled by the empty label τ, since we do not need additional information about the transitions. The proof that $\mathcal{M}[\mathcal{B}]$ is well-defined can be found in Appendix C. Fig. 6.2 shows the MFBP belonging to the introductory example.

For the rest of the section we fix an $n - type$ MFBP $\mathcal{B} = (\mathrm{Pr}_1, \mathrm{Pr}_2, \ldots, \mathrm{Pr}_n) \in (\mathrm{Dist}_F(\mathbb{P}))^n$. We are interested in the probability of a population going eventually extinct, i.e. reaching the state $\overline{0}$.

Definition 41. (Extinction probabilities)
Let ψ_i be the probability of reaching $\overline{0}$ in $\mathcal{M}[\mathcal{B}]$, starting at $\mathbf{e}^i$, for $i \in \{1, \ldots, n\}$,

i.e. $\psi_i = \Pr_{\mathbf{e}^i}[\mathrm{Reach}(\mathcal{M}[\mathcal{B}], \{\overline{0}\})]$. We call the vector $\psi = (\psi_1, \ldots, \psi_n)^\top \in [0,1]^n$ the *extinction probabilities* of $\mathcal{B}$. $\square$

Given ψ, we can compute the probability of reaching $\overline{0}$ for arbitrary starting populations $\mathbf{s} \in \mathbb{P}$, which is $\prod_{i=1}^{n} (\psi_i)^{\mathbf{s}_i}$. However, $\mathcal{M}[\mathcal{B}]$ is an infinite-state Markov chain, hence we are not able to construct it explicitly. Luckily, this is not necessary for our purposes. It turns out that for computing extinction probabilities it is more helpful to consider *generating functions* of MFBPs:

Definition 42. (Generating functions of MFBPs)
Let $\mathbf{s} \in \mathbb{N}^n$. We define the generating function $f[\mathcal{B}]_{\mathbf{s}}$, which is a formal power series in the variables $X_1, \ldots, X_n$, by the following rules:

- If $\mathbf{s} = \mathbf{e}^i$ for an $i \in \{1, \ldots, n\}$, then

$$f[\mathcal{B}]_{\mathbf{s}}(X_1, \ldots, X_n) := \sum_{\mathbf{d} \in \mathbb{P}} \Pr_i(\mathbf{d}) \cdot \prod_{j=1}^{n} X_j^{\mathbf{d}_j}$$

- If otherwise $\mathbf{s} = \sum_{\ell=1}^{n} \alpha_\ell \cdot \mathbf{e}^\ell$ for $\{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{N}$, then

$$f[\mathcal{B}]_{\mathbf{s}}(X_1, \ldots, X_n) := \sum_{\ell=1}^{n} \alpha_\ell \cdot f[\mathcal{B}]_{\mathbf{e}^\ell}(X_1, \ldots, X_n).$$

$\square$

For instance, the generating functions of $\mathbf{e}^1$ and $\mathbf{e}^2$ for our introductory example are given by

$$f[\mathcal{B}]_{\mathbf{e}^1}(X_1, X_2) = 0.75^2 \cdot X_1^2 + 0.75 \cdot 0.25 \cdot X_1 X_2 + 0.25$$
$$f[\mathcal{B}]_{\mathbf{e}^2}(X_1, X_2) = 0.01 \cdot 0.5 \cdot X_2^2 + 0.01 \cdot 0.5 \cdot X_1 X_2 + 0.99.$$

Generating functions are in fact a compact representation of $(\Pr_1, \Pr_2, \ldots, \Pr_n)$: Every monomial of $f[\mathcal{B}]_{\mathbf{e}^i}$ (with nonzero coefficient) describes an element $\mathbf{d} \in \mathbb{P}$ with $\Pr_i(\mathbf{d}) > 0$ (encoded by the exponents of the variables $X_1, \ldots, X_n$) as well as $\Pr_i(\mathbf{d})$ itself (as coefficient). Extinction probabilities and generating functions of a branching process are closely related, as the following theorem shows:

Theorem 10. *(ψ as solution of a nonlinear equation system)*
The equation system in the variables $X_1, \ldots, X_n$

$$\begin{aligned} X_1 &= f[\mathcal{B}]_{\mathbf{e}^1}(X_1, \ldots, X_n) \\ &\ldots \\ X_n &= f[\mathcal{B}]_{\mathbf{e}^n}(X_1, \ldots, X_n). \end{aligned}$$

has a last nonnegative solution $X_1 = \mu_1, \ldots, X_n = \mu_n$, i.e. for every other nonnegative solution $X_1 = x_1, \ldots, X_n = x_n$, $(\mu_1, \ldots, \mu_n)^\top \leq (x_1, \ldots, x_n)^\top$; further $\psi = (\mu_1, \ldots, \mu_n)^\top$ holds. $\square$

Proof.
We define $f : [0,1]^n \to [0,1]^n$ by

$$f(\mathbf{x}) = (f[\mathcal{B}]_{\mathbf{e}^1}(\mathbf{x}_1, \ldots, \mathbf{x}_n), \ldots, f[\mathcal{B}]_{\mathbf{e}^n}(\mathbf{x}_1, \ldots, \mathbf{x}_n))^\top, \tag{6.1}$$

for every $\mathbf{x} \in [0,1]^n$ (note the we can choose $[0,1]^n$ as range of f since the coefficients of each $f[\mathcal{B}]_{\mathbf{e}^i}$ sum up to 1 and are nonnegative). Every solution of 10 is a fixed point of f and vice versa.
Since f is a continuous function (with respect to the complete lattice $([0,1]^n, \leq)$), by Kleene's theorem (Theorem 2) there exists a least nonnegative fixed point $\mu_f = \lim_{k\to\infty} f^k(\overline{0})$.
We define for all $j \in \{1, \ldots, n\}$ the set of runs $S(j,k)$ of $\mathcal{M}[\mathcal{B}]$ that start at $\mathbf{e}^j$ and end in the $\overline{0}$-population after maximal k time steps :

$$S(j,k) = \bigcup_{\substack{\pi = \mathbf{e}^j \xrightarrow{\ell_1} q_1 \xrightarrow{\ell_2} \ldots \xrightarrow{\ell_i} q_i : \\ q_i = \overline{0} \wedge i \leq k}} \mathrm{Cyl}(\pi, \mathcal{M}[\mathcal{B}]).$$

We denote by $s(j,k) = \mathrm{Pr}_{\mathbf{e}^j}[S(j,k)]$ the probability of reaching $\overline{0}$ in at most k steps starting at $\mathbf{e}^j$. Since $S(j,k) \subseteq S(j,k+1)$ for all k, and

$$\psi_j = \mathrm{Pr}_{\mathbf{e}^j}\left[\bigcup_{k\geq 0} S(j,k)\right],$$

we get with Def. 41 that $\lim_{k\to\infty} s(j,k) = \psi_j$. Note also that for $k > 0$ by Def. 40

$$s(j,k) = \sum_{\mathbf{d}\in\mathbb{P}} \mathrm{Pr}_j(\mathbf{d}) \cdot \prod_{i=1}^{n} s(i,k-1)_i^{\mathbf{d}_i}$$

(the line of every individual generated after one step has to become extinct after at most $k-1$ steps). We show $s(j,k) = (f^k(\overline{0}))_j$ for all $k \leq 0$ and $j \in \{1,\ldots,n\}$ by induction over k.

- $k = 0$: $s(j,k) = 0 = (f^k(\overline{0}))_j$.
- $k = 1$: $(f^k(\overline{0}))_j = \sum_{\mathbf{d}\in\mathbb{P}} \mathrm{Pr}_j(\mathbf{d}) \cdot \prod_{i=1}^{n} 0^{d_i} = \mathrm{Pr}_j(\overline{0}) = s(j,1)$.
- $k > 1$:

$$\begin{aligned}
(f^k(\overline{0}))_j &= (f(f^{k-1}(\overline{0})))_j \\
&= (f((s(1,k-1), s(2,k-1), \ldots, s(n,k-1))^\top))_j && \text{(Ind. hypothesis)} \\
&= \sum_{\mathbf{d}\in\mathbb{P}} \mathrm{Pr}_j(\mathbf{d}) \cdot \prod_{i=1}^{n} s(i,k-1)_i^{\mathbf{d}_i} && \text{(Def. 42, Def. of } s) \\
&= s(j,k). && \text{(note above)}
\end{aligned}$$

Putting all together we obtain

$$(\mu_f)_j = \lim_{k\to\infty} (f^k(\overline{0}))_j = \lim_{k\to\infty} s(j,k) = \psi_j.$$

$\square$

$\overline{1}$ is always a nonnegative solution for the equation system described in the previous theorem. However, in general it is not the least one: in our example $\mu_f = (\mu_1, \mu_2)^\top$, with $0.4436 < \mu_1 = \psi_1 < 0.4437$ and $0.997 < \mu_2 = \psi_2 < 0.998$. If the population starts e.g. with just one bacteria possessing allele B, the population eventually goes extinct with probability at least 0.997. Interestingly, if the probability of an individual with allele A dying in one time step were 0.45, ψ would become $\overline{1}$, i.e. *every* starting population would eventually die out with probability 1.

In summary we have seen that we can obtain the extinction probabilities of an MFBP by computing the least nonnegative solution of the equation system from Theorem 10. In the rest of this section we sketch how important problems of related formalisms, namely stochastic context-free grammars and special classes of probabilistic recursive programs, can be reduced to computing extinction probabilities of MFBPs.

6.1.1 Stochastic Context-Free Grammars

Stochastic context-free grammars (SCFGs) are a formal model used in natural language processing and for structure prediction in molecular biology (see e.g. Anderson et al. [1]). They represent context-free grammars whose production rules are weighted with a non-zero probability, and the probabilities of all rules with the same left hand side add up to 1. At each derivation step a production rule is selected according to its probability. More precisely, a *stochastic context-free grammar (SCFG for short)* is a tuple

$$G = (N, \Sigma, \hookrightarrow, N_1),$$

where $N = \{N_1, \dots, N_n\}$ is a finite set of nonterminals and Σ a finite alphabet of terminals. Nonterminals and terminals form disjunct sets. The relation $\hookrightarrow \subseteq N \times ((0,1] \cap \mathbb{Q}) \times (N \cup \Sigma)^*$ contains the *production rules* of the grammar. For $(X, p, w) \in \hookrightarrow$ we write $X \overset{p}{\hookrightarrow} w$. For every $X \in N$ it holds that $\sum_{X \overset{p}{\hookrightarrow} w} p = 1$. The nonterminal N_1 is the axiom or starting nonterminal of the grammar.
Consider for example the grammar $G = (\{N_1, N_2\}, \{*, \times\}, \hookrightarrow)$, with the following production rules:

$$N_1 \xhookrightarrow{0.75 \cdot 0.75} N_1 * N_1,\ N_1 \xhookrightarrow{0.75 \cdot 0.25} N_1 * N_2,\ N_1 \xhookrightarrow{0.25} \times \text{ and}$$
$$N_2 \xhookrightarrow{0.01 \cdot 0.5} N_2 * N_2,\ N_2 \xhookrightarrow{0.01 \cdot 0.5} N_1 * N_2,\ N_2 \xhookrightarrow{0.99} \times$$

G is an alternative representation of the bacteria population from the introductory example: every rule represents a reproduction step of a single bacteria, a terminal "$*$" represents a division taking place, "$\times$" represents the death of a cell.
Derivations in SCFGs can be compactly represented as *parse trees*, similar to

classical context-free grammars. A parse tree T is a finite ordered tree; its root r is labeled by N_1 and a probability p_r, every inner node i is labeled by an element from $N \times ((0,1] \cap \mathbb{Q})$, and every leaf is labeled by an element from $\Sigma \times \{1\}$. Every node labeled by $(X, p) \in N \times ((0,1] \cap \mathbb{Q})$ has an ordered sequence of children $c_1, \ldots, c_k$ labeled by $(s_1, p_1), \ldots (s_k, p_k)$, respectively, such that $X \overset{p}{\hookrightarrow} s_1 \ldots s_k$. The word $w \in \Sigma^*$ generated by T is obtained by the sequence of first components of leaf labels in prefix order. We can assign probabilities to a parse tree by multiplying all probabilities (i.e., second components of the labels) in the tree. For example consider the parse tree in the left part of Fig. 6.3. Its probability is $\frac{1}{4}^3 \cdot \frac{3}{4}^3 \cdot \frac{99}{100} \approx 0.00653$.

We can assign a probability $P(w)$ to every word $w \in \Sigma^*$ as the sum of the probabilities of all parse trees generating w. Thus we can define the *language* $L(G)$ of an SCFG G by setting $L(G) : \Sigma^* \to [0,1]$ with $L(G)(w) = P(w)$. Interestingly, it might be the case that $\sum_{w \in \Sigma^*} L(G)(w) < 1$, i.e., there might be a non-zero probability that a derivation starting at N_1 never terminates with a word in Σ^*. In this case we call G *inconsistent*, otherwise G is *consistent*.

Deciding whether a grammar is consistent is an important problem in Natural Language Processing (Manning and Schuetze [70]). We will reduce it to computing extinction probabilities for a branching process. Let G be an SCFG in the following. We define a branching process $\mathcal{B}_G$ which can be seen as an abstraction of G: we do not take generated words into account, but rather only the probabilities of generating new nonterminals during a derivation step. Since G is context-free, the derivations of nonterminals do not interfere with each other. A derivation d ends if and only if each of the derivations starting at a generated nonterminal of d end. Therefore, only the number of occurences of the different nonterminals is important for the termination of a derivation, not their position within the derived strings.

For a given alphabet Σ and an ordered alphabet $M = \{A_1, \ldots, A_n\}$ with $M \subseteq \Sigma$, we define for every $w \in \Sigma$ the *commutative image* relative to M by setting

$$ci_M(w) := (|w|_{A_1}, |w|_{A_2}, \ldots, |w|_{A_n})^\top.$$

Definition 43. (MFBPs from SCFGs)

Let $G = (\{N_1, \ldots, N_n\}, \Sigma, \hookrightarrow, \delta, N_1)$ be an SCFG. We assume that $\{N_1, \ldots, N_n\}$ is an ordered alphabet. We define the n-type MFBP $\mathcal{B}_G = (\mathrm{Pr}_1, \ldots, \mathrm{Pr}_n)$ by setting for every $\mathbf{d} \in \mathbb{N}^n$ and every $1 \leq i \leq n$:

$$\mathrm{Pr}_i(\mathbf{d}) := \sum_{\substack{N_i \overset{p}{\hookrightarrow} w: \\ ci_N(w) = \mathbf{d}}} p.$$

□

Consider again the parse tree in Fig. 6.3. On the right side of the figure we see the population development in $\mathcal{B}_G$ corresponding to the parse tree on the left. $\mathcal{B}_G$ is in fact the branching process we used for our introductory example.
We call a nonterminal N of an SCFG G reachable if there exists a parse tree where N occurs. Nonterminals that are not reachable can easily be detected (and removed) from an SCFG (see e.g. Hopcroft et al. [57]). We can now state the following easy fact (Etessami and Yannakakis [43]):

Lemma 5. *(Extinction probabilities and consistency of SCFGs)*
Let G be an SCFG having only reachable nonterminals. The extinction probabilities ψ of $\mathcal{B}_G$ are then all equal to 1 iff G is consistent. □

6.1.2 Stateless Probabilistic Pushdown Automata

We mention another closely related model, namely *stateless probabilistic pushdown automata* (see e.g. Esparza et al. [38]), a simple class of probabilistic programs with recursion (or dynamic process creation). Another notion of this class of programs are recursive Markov chains (Etessami and Yannakakis [43]). Stateless probabilistic pushdown automata (spPDA for short) are tuples $(\Gamma, \hookrightarrow, P)$, with $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ a stack alphabet, $\hookrightarrow \subseteq \Gamma \times \Gamma^*$ a transition relation, and $P : \hookrightarrow \to (0, 1]$ a function assigning probabilities to transitions. Again we write $\gamma \overset{P(\gamma)}{\hookrightarrow} w$ instead of $(\gamma, w) \in \hookrightarrow$. For every $\gamma \in \Gamma$, $\sum_{\gamma \overset{p}{\hookrightarrow} w} p = 1$ holds.
An important question for spPDAs is how to compute *termination probabilities*: starting with a given stack configuration $w \in \Gamma^*$, what is the probability of eventually reaching the configuration ϵ (the empty stack), if transitions are chosen according to P.

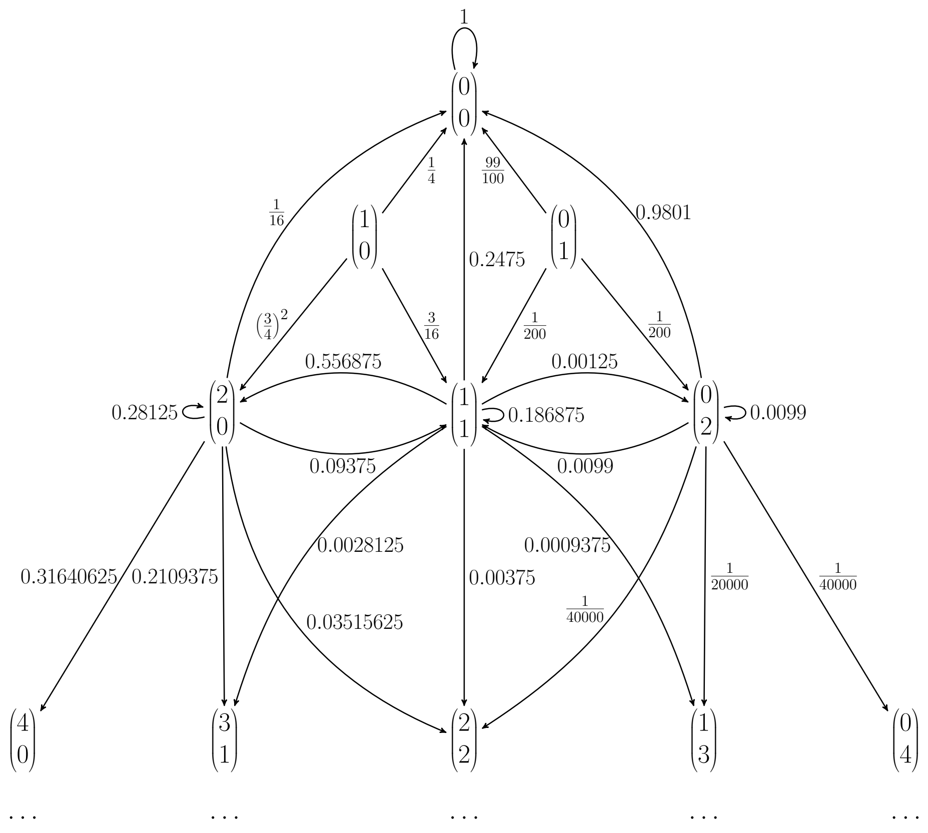
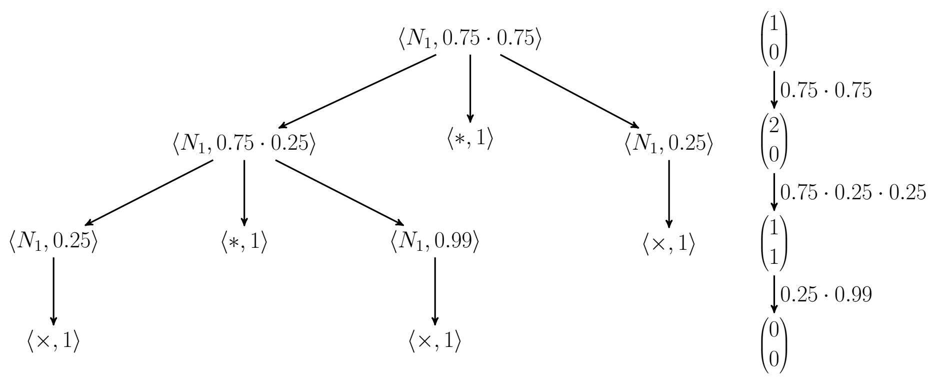


Figure 6.2: Fragment of the Markov chain belonging to the example MFBP.



$\langle N_1, 0.75 \cdot 0.75\rangle$
$\langle N_1, 0.75 \cdot 0.25\rangle$
$\langle *, 1\rangle$
$\langle N_1, 0.25\rangle$
$\langle N_1, 0.25\rangle$
$\langle *, 1\rangle$
$\langle N_1, 0.99\rangle$
$\langle \times, 1\rangle$
$\langle \times, 1\rangle$
$\langle \times, 1\rangle$
$\binom{1}{0}$
$0.75 \cdot 0.75$
$\binom{2}{0}$
$0.75 \cdot 0.25 \cdot 0.25$
$\binom{1}{1}$
$0.25 \cdot 0.99$
$\binom{0}{0}$

Figure 6.3: Derivation tree and corresponding population development of the SCFG example.

The similarity to the consistency problems of SCFGs is obvious. We give an analogous translation from spPDAs to MFBPs for this problem:

Definition 44. (MFBPs from spPDAs)
Let $\mathcal{A} = (\{\gamma_1, \dots, \gamma_n\}, \hookrightarrow, P)$ be an spPDA. For every $w \in \Gamma^*$ let

$$ci(w) := (|w|_{N_1}, |w|_{N_2}, \dots, |w|_{N_n}).$$

We define the n-type MFBP $\mathcal{B}_\mathcal{A} = (\mathrm{Pr}_1, \dots, \mathrm{Pr}_n)$ by setting, for every $\mathbf{d} \in \mathbb{N}^n$,

$$\mathrm{Pr}_i(\mathbf{d}) := \sum_{\substack{\gamma_i \overset{p}{\hookrightarrow} w: \\ ci(w) = \mathbf{d}}} p.$$

□

Again we can state a reduction (see also Esparza et al. [38]):

Lemma 6. *(Extinction probabilities and almost-sure termination of spPDAs)*
Let $\mathcal{A} = (\Gamma = \{\gamma_1, \dots, \gamma_n\}, \hookrightarrow, P)$ be a spPDA, and ψ the extinction probabilities of $\mathcal{B}_\mathcal{A}$. The termination probability of a given stack configuration $w \in \Gamma^*$ is $\prod_{i=1}^n (\psi_i^{|w|_{\gamma_i}})$. In particular, $\mathcal{A}$ is terminating with probability one from any given initial configuration iff $\psi = \overline{1}$. □

As an example we model the program mentionend in the introduction as an spPDA $\mathcal{A} = (\{\gamma\}, \hookrightarrow, P)$. The value of the variable k is represented by the number of elements on the stack, and $\hookrightarrow$ resp. P is given by the rules

$$\gamma \overset{p}{\hookrightarrow} \gamma\gamma$$
$$\gamma \xhookrightarrow{(1-p)} \epsilon.$$

The corresponding MFBP $\mathcal{B}_\mathcal{A} = (\mathrm{Pr}_1)$ has only one type and is defined by

$$\mathrm{Pr}_1((i)) := \begin{cases} p & \text{if } i = 2 \\ 1 - p & \text{if } i = 0 \\ 0 & \text{otherwise.} \end{cases}$$

We get $f[\mathcal{B}_{\mathcal{A}}]_{\mathbf{e}^1}(X_1) = pX_1^2 + (1-p)$. By solving $pX_1^2 + (1-p) = X_1$ we obtain the extinction probability $\psi_1 = \min(\frac{1-p}{p}, 1)$: for $p \leq \frac{1}{2}$ the program is almost-surely terminating (independent of the initial stack configuration, i.e. the value of `k`).

6.2 Probabilistic Systems of Polynomials

In this section we introduce *Probabilistic Systems of Polynomials*, which form a class of equation systems that are important for studying extinction probabilities of MFBPs. We need some notations and concepts from matrix theory.

6.2.1 Preliminaries

By $\mathbb{R}^{m\times n}$ we denote the set of real matrices with m rows and n columns. We write Id for the identity matrix. For a matrix $M \in \mathbb{R}^{m\times n}$, we denote the i-th component in the j-th column by $M_{i,j}$. Let now $A \in \mathbb{R}^{n\times n}$ be a square matrix. A vector $\mathbf{v}$ is an *eigenvector* of A with *eigenvalue* $\lambda \in \mathbb{R}$ if $A\mathbf{v} = \lambda\mathbf{v}$. We denote by $\rho(A)$ the *spectral radius* of A, i.e., the maximum of the absolute values of the eigenvalues. We also define the matrix star $A^* = \sum_{i=0}^{\infty} A^i = Id + \sum_{i=1}^{\infty} A^i$ for A. A matrix is *nonnegative* if all its entries are nonnegative. A nonnegative matrix $A \in \mathbb{R}^{n\times n}$ is *irreducible* if for every $k, l \in \{1, \ldots, n\}$ there exists an $i \in \mathbb{N}$ so that $(A^i)_{k,l} \neq 0$.

6.2.2 Definition and Properties

In Section 6.1 we have seen that the vector of extinction probabilities of an n-type MFBP $\mathcal{B}$ is the least nonnegative fixed point of a polynomial equation system with nonnegative coefficients. The polynomials $f[\mathcal{B}]_{\mathbf{e}^i}(X_1, \ldots, X_n)$ forming the left hand side of every equation also have the property that $f[\mathcal{B}]_{\mathbf{e}^i}(1, \ldots, 1) = 1$, i.e., their coefficients sum up to 1.
The vector $(f[\mathcal{B}]_{\mathbf{e}^1}(X_1, \ldots, X_n), \ldots, f[\mathcal{B}]_{\mathbf{e}^n}(X_1, \ldots, X_n))^\top$ is a *probabilistic systems of polynomials*:

Definition 45. (Probabilistic System of Polynomials)

Let

$$X_1 = f_1(X_1, \dots, X_n) \quad \dots \quad X_n = f_n(X_1, \dots, X_n),$$

be an equation system where the f_i are polynomials in the variables $X_1, \dots, X_n$ with positive real coefficients, and for every polynomial f_i the sum of its coefficients is *at most* 1. The vector $f := (f_1, \dots, f_n)^\top$ is called a *probabilistic system of polynomials* (PSP for short) and is identified with its induced function $f : \mathbb{R}^n \to \mathbb{R}^n$. If $X_1, \dots, X_n$ are the formal variables of f, we define $\overline{X} := (X_1, \dots, X_n)^\top$ and $\mathrm{Var}(f) := \{X_1, \dots, X_n\}$. The *degree* of f is the maximum of the degrees of $f_1, \dots, f_n$. PSPs of degree 0 (resp. 1 resp. >1) are called *constant* (resp. *linear* resp. *superlinear*). PSPs f where the degree of each f_i is at least 2 are called *purely superlinear.* $\square$

We write f' for the *Jacobi matrix* of f, i.e., $(f')_{i,j} = \frac{\partial f_i}{\partial X_j}$. We assume that f is represented as a list of polynomials, and each polynomial is a list of its monomials. If $S \subseteq \{X_1, \dots, X_n\}$, then f_S denotes the result of removing the polynomial $f_i(X_1, \dots, X_n)$ from f for every $x_i \notin S$; further, given $\mathbf{x} \in \mathbb{R}^n$ and $B \in \mathbb{R}^{n \times n}$, we denote by $\mathbf{x}_S$ and B_{SS} the vector and the matrix obtained from $\mathbf{x}$ and B by removing the entries with indices i such that $X_i \notin S$. The coefficients are represented as fractions of positive integers. The *size* of f is the size of that representation. For $\{\mathbf{x}, \mathbf{y}\} \subseteq \mathbb{R}^n$, we write $f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f'(\mathbf{x})\mathbf{y} + R_f(\mathbf{x}, \mathbf{y})$ for the Taylor expansion of f at $\mathbf{x}$. We often write R instead of R_f if the context is clear.

Dependence relation and SCCs. Given a PSP f, a variable X_i *depends directly* on a variable X_j if X_j "occurs" in f_i, more formally if $\frac{\partial f_i}{\partial X_j}$ is not the constant 0. A variable X_i *depends* on X_j if X_i depends directly on X_j or there is a variable X_k such that X_i depends directly on X_k and X_k depends on X_j. We often consider the *strongly connected components* (or SCCs for short) of the dependence relation. The SCCs of a PSP can be computed in linear time using e.g. Tarjan's algorithm (Tarjan [95]). They can be used to define a directed acyclic graph: its nodes are the SCCs and a transition from an SCCs S_1 to an SCC S_2 exists if $S_1 \neq S_2$ and there is a variable $X_i \in S_1$ that depends directly on a variable $X_j \in S_2$. We refer to this graph as the DAG (of the SCCs of f). A *bottom SCC S* is an SCC whose variables only depend on each other. An SCC S of a PSP f is *constant* resp. *linear* resp. *superlinear* resp. *purely superlinear* if the PSP $\tilde{f}$ has the respective property, where $\tilde{f}$ is obtained by restricting f to the S-components and replacing all variables not in S by the constant 1. A PSP is an *scPSP* if it is not constant and consists of only one SCC. Notice that a PSP f is an scPSP if and only if $f'(\overline{1})$ is irreducible.

Fixed points, post-fixed points, and pre-fixed points. A fixed point of a PSP f is a vector $\mathbf{x} \geq \overline{0}$ with $f(\mathbf{x}) = \mathbf{x}$. For every $\mathbf{v} \in [0,1]^n$, $f(\mathbf{v}) \in [0,1]^n$ holds, due to our assumptions on the coefficients of f. We can therefore define the restriction

$$f|_{[0,1]^n} : [0,1]^n \to [0,1]^n$$

of f (i.e., the function defined on $[0,1]^n$ and there coinciding with f). $f|_{[0,1]^n}$ is continuous in $[0,1]^n \to [0,1]^n$ with respect to the complete lattice $([0,1]^n, \leq)$ (recall Def. 3). We get using Kleene's theorem (Theorem 2) that there exists a least nonnegative fixed point μ_f of f (equal to the least fixed point of $f|_{[0,1]^n}$), i.e., $\mu_f \leq \mathbf{x}$ holds for every fixed point $\mathbf{x}$. Moreover, the sequence $\overline{0}, f(\overline{0}), f(f(\overline{0})), \ldots$ converges to μ_f. Vectors $\mathbf{x}$ with $\mathbf{x} \leq f(\mathbf{x})$ (resp. $\mathbf{x} \geq f(\mathbf{x})$) are called *pre-fixed* (resp. *post-fixed*) points (for f). Notice that we always have $\overline{0} \leq \mu_f \leq \overline{1}$ (this can also bee seen by the fact that μ_f is the least nonnegative post-fixed point of f and $\overline{1}$ is always a post-fixed point of f, using the Knaster-Tarski theorem). It is easy to detect and remove all components i with $(\mu_f)_i = 0$ by a simple round-robin method (see e.g. Esparza et al. [37]), which needs linear time in the size of f. We therefore assume in the following that $\mu_f \succ \overline{0}$ for all PSPs f we consider.

6.3 An Algorithm for Deciding whether $\psi = \overline{1}$

We have seen that for applications like deciding consistency of SCFGs or checking almost-sure termination of spPDAs it is crucial to know exactly whether the extinction probabilities of an MFBP are all equal to 1. This is equivalent to whether $\mu_f = \overline{1}$ holds for the corresponding PSP f constructed from the MFBP. We call this condition *consistency* of a PSP:

Definition 46. (Consistency of PSPs)
A PSP f is *consistent* if $\mu_f = \overline{1}$; otherwise it is *inconsistent*. Similarly, we call a component i of f consistent if $(\mu_f)_i = 1$. □

(here we borrow the term consistency from our introduction to SCFGs). We present a new algorithm for the consistency problem, i.e., the problem to check a PSP for consistency.

6.3.1 Checking Consistency using Linear Programming

It was proved in Etessami and Yannakakis [43] that consistency is checkable in polynomial time by reduction to Linear Programming (LP). We first observe that consistency of general PSPs can be reduced to consistency of scPSPs by computing the DAG of SCCs, and checking consistency SCC-wise (Etessami and Yannakakis [43]): Take any bottom SCC S, and check the consistency of f_S. (Notice that f_S is either constant or an scPSP; if constant, f_S is consistent iff $f_S = 1$, if an scPSP, we can check its consistency by assumption.) If f_S is inconsistent, then so is f, and we are done. If f_S is consistent, then we remove every f_i from f such that $x_i \in S$, replace all variables of S in the remaining polynomials by the constant 1, and iterate (choose a new bottom SCC, etc.). Note that this algorithm processes each polynomial at most once, as every variable belongs to exactly one SCC. It remains to reduce the consistency problem for scPSPs to LP. The first step is:

Theorem 11. *(Etessami and Yannakakis [43], Harris [54])*
An scPSP f is consistent iff $\rho(f'(\overline{1})) \leq 1$ (i.e., iff the spectral radius of the Jacobi matrix f' evaluated at the vector $\overline{1}$ is at most 1). □

The second step consists of observing that the matrix $f'(\overline{1})$ of an scPSP f is irreducible and nonnegative. It is shown in Etessami and Yannakakis [43] that $\rho(A) \leq 1$ holds for an irreducible and nonnegative matrix A iff the system of inequalities

$$A\mathbf{x} \geq \mathbf{x} + \overline{1} \ , \ \mathbf{x} \geq \overline{0} \tag{6.2}$$

is infeasible. However, no strongly polynomial algorithm for LP is known, and we are not aware that Eq. 6.2 falls within any subclass solvable in strongly polynomial time (Grötschel et al. [47]).

6.3.2 Our Algorithm

We provide a very simple, strongly polynomial time algorithm to check whether $\rho(f'(\overline{1})) \leq 1$ holds. We need some results from Perron-Frobenius theory (see e.g. Berman and Plemmons [12]).

Lemma 7. *(Facts from Perron-Frobenius theory)*
Let $A \in \mathbb{R}^{n\times n}$ be nonnegative and irreducible.

(1) $\rho(A)$ is a simple *eigenvalue of A.*

(2) There exists an eigenvector $\mathbf{v} \succ \overline{0}$ with $\rho(A)$ as eigenvalue.

(3) Every eigenvector $\mathbf{v} \succ \overline{0}$ has $\rho(A)$ as eigenvalue.

(4) For all $\alpha, \beta \in \mathbb{R} \setminus \{0\}$ and $\mathbf{v} > \overline{0}$: if $\alpha\mathbf{v} < A\mathbf{v} < \beta\mathbf{v}$, then $\alpha < \rho(A) < \beta$.

□

The following lemma is the key to the algorithm:

Lemma 8. *Let $A \in \mathbb{R}^{n\times n}$ be nonnegative and irreducible.*

(a) Assume there is $\mathbf{v} \in \mathbb{R}^n \setminus \{\overline{0}\}$ such that $(Id - A)\mathbf{v} = \overline{0}$. Then $\rho(A) \leq 1$ iff $\mathbf{v} \succ \overline{0}$ or $\mathbf{v} \prec \overline{0}$.

(b) Assume $\mathbf{v} = \overline{0}$ is the only solution of $(Id - A)\mathbf{v} = \overline{0}$. Then there exists a unique $\mathbf{x} \in \mathbb{R}^n$ such that $(Id - A)\mathbf{x} = \overline{1}$, and $\rho(A) \leq 1$ iff $\mathbf{x} \geq \overline{1}$ and $A\mathbf{x} < \mathbf{x}$.

□

Proof.

(a) From $(Id - A)\mathbf{v} = \overline{0}$ it follows $A\mathbf{v} = \mathbf{v}$. We see that $\mathbf{v}$ is an eigenvector of A with eigenvalue 1. So $\rho(A) \geq 1$.

($\Leftarrow$): As both $\mathbf{v}$ and $-\mathbf{v}$ are eigenvectors of A with eigenvalue 1, we can assume w.l.o.g. that $\mathbf{v} \succ \overline{0}$. By Lemma 7(3), $\rho(A)$ is the eigenvalue of $\mathbf{v}$, and so $\rho(A) = 1$.

($\Rightarrow$): Since $\rho(A) \leq 1$ and $\rho(A) \geq 1$, it follows that $\rho(A) = 1$. By Lemma 7 (1) and (2), the eigenspace of the eigenvalue 1 is one-dimensional and contains a vector $\mathbf{x} \succ \overline{0}$. So $\mathbf{v} = \alpha \cdot \mathbf{x}$ for some $\alpha \in \mathbb{R}, \alpha \neq 0$. If $\alpha > 0$, we have $\mathbf{v} \succ \overline{0}$, otherwise $\mathbf{v} \prec \overline{0}$.

(b) With the assumption and basic facts from linear algebra it follows that $(Id - A)$ has full rank and therefore $(Id - A)\mathbf{x} = \overline{1}$ has a unique solution $\mathbf{x}$. We still have to prove the second part of the conjunction:

($\Leftarrow$): Follows directly from Lemma 7 (4).

($\Rightarrow$): Let $\rho(A) \leq 1$. Assume for a contradiction that $\rho(A) = 1$. Then, by Lemma 7(1), the matrix A would have an eigenvector $\mathbf{v} \neq \overline{0}$ with eigenvalue 1, so $(Id - A)\mathbf{v} = \overline{0}$, contradicting the assumption. So we have, in fact, $\rho(A) < 1$. By standard matrix facts (see e.g. Berman and Plemmons [12]), this implies that $(Id - A)^{-1} = A^* = \sum_{i=0}^{\infty} A^i$ exists, and so we have $\mathbf{x} = (Id - A)^{-1}\overline{1} = A^*\overline{1} \geq \overline{1}$. Furthermore, $A\mathbf{x} = \sum_{i=1}^{\infty} A^i\overline{1} < \sum_{i=0}^{\infty} A^i\overline{1} = \mathbf{x}$. □

Theorem 11 directly yields an algorithm for checking the consistency of scPSPs, which we can extend to multiple SCCs in a similar way as described above for the LP-based approach:
Let $f(X_1, \ldots, X_n)$ be a PSP. The algorithm performs the following steps:

1. Compute the SCC-DAG of f, and choose any bottom SCC S (possible in linear time). If f_S is constant go to (2), otherwise go to (3).

2. *(f_S is constant):* If $f_S \neq 1$ then report “inconsistent” and terminate. Otherwise go to (4).

3. *(f_S is an scPSP):* Compute A as the Jacobi matrix f'_S evaluated at $\overline{1}$, i.e. $A = f'_S(\overline{1})$. Solve the system $(Id - A)\mathbf{v} = \overline{0}$ using Gaussian elimination and perform the following case distinction:

3.1. There is a vector $\mathbf{v} \neq \overline{0}$ such that $(Id - A)\mathbf{v} = \overline{0}$: If neither $\mathbf{v} \succ \overline{0}$ nor $\mathbf{v} \prec \overline{0}$, report "inconsistent" (according to Lemma 8(a)) and terminate. Otherwise go to (4).

3.2. $\mathbf{v} = \overline{0}$ is the only solution of $(Id - A)\mathbf{v} = \overline{0}$: solve $(Id - A)\mathbf{x} = \overline{1}$ using Gaussian elimination. If $\mathbf{x} \not\geq \overline{1}$ or $A\mathbf{x} \not< \mathbf{x}$ report "inconsistent" and terminate (according to Lemma 8(b)). Otherwise go to (4).

4. If $f_S = f$ report "consistent" and terminate. Otherwise remove for every $x_i \in S$ the polynomial f_i from f, replace all variables of S in the remaining polynomials by the constant 1, and go to (1).

Since Gaussian elimination of a rational n-dimensional linear equation system can be carried out in strongly polynomial time using $O(n^3)$ arithmetic operations (see e.g. Grötschel et al. [47]) we obtain:

Theorem 12. *(Consistency of PSPs)*
Let $f(X_1, \dots, X_n)$ be a PSP. There is a strongly polynomial time algorithm that uses $O(n^3)$ arithmetic operations and determines the consistency of f. □

We note that the reduction of an MFBP to a PSP can be done in linear time in the size of an representation of an MFBP. We get:

Corollary 1. *(Test for $\psi = \overline{1}$ for MFBPs)*
Let $\mathcal{B} = (Pr_1, \dots, Pr_n)$ be an n-type MFBP and let $\psi \in [0,1]^n$ be its extinction probabilities. We can decide in strongly polynomial time whether $\psi = \overline{1}$, using $O(n^3)$ arithmetic operations. □

6.3.3 Case Study: MFBPs with ψ being "almost" $\overline{1}$

In this section we illustrate some issues faced by algorithms that try to decide whether the extinction probabilities ψ_i for an MFBP are all 1 or, equivalently, try to solve the consistency problem for PSPs.

Consider the following family of MFBPs, which are given in form of their corresponding (sc)PSPs $h^{(n)}$, $n \geq 2$:

$$\begin{aligned} h^{(n)} = \Big(&0.5X_1^2 + 0.1X_n^2 + 0.4, \\ &0.01X_1^2 + 0.5X_2 + 0.49, \\ &0.01X_2^2 + 0.5X_3 + 0.49, \\ &\qquad\qquad \dots \\ &0.01X_{n-1}^2 + 0.5X_n + 0.49 \Big)^\top . \end{aligned}$$

It is not hard to show that $h^{(n)}(\mathbf{p}) \prec \mathbf{p}$ holds for $\mathbf{p} = (1-0.02^n, \dots, 1-0.02^{2n-1})^\top$, so we have $\mu_{h^{(n)}} \prec \overline{1}$ (see Theorem 16), i.e., the $h^{(n)}$ are inconsistent.

The tool PReMo (Wojtczak and Etessami [102]) relies on Java's floating-point arithmetic to compute approximations of the least fixed point of a PSP. We invoked PReMo for computing approximants of $\mu_{h^{(n)}}$ for different values of n between 5 and 100. Due to its fixed precision, PReMo's approximations for $\mu_{h^{(n)}}$ are ≥ 1 in all components if $n \geq 7$. This might lead to the wrong conclusion that $h^{(n)}$ is consistent.

Recall that the consistency problem can be solved by checking the feasibility of the system in Eq. (6.2) with $A = f'(\overline{1})$. We checked it with lp_solve (*lp_solve reference guide* [69]), a well-known LP tool using hardware floating-point arithmetic. The tool wrongly states that Eq. (6.2) has no solution for $h^{(n)}$-systems with $n > 10$. This is due to the fact that the solutions cannot be represented adequately using machine number precision.[1]

Finally, we also checked feasibility with Maple's Simplex package (*Maple* [71]), which uses exact arithmetic, and compared its performance with the implementation, also in Maple, of our consistency algorithm.[2] (We note that similar experiments were already performed in Esparza et al. [36]. However for the experimental

[1]The mentioned problems of PReMo and lp_solve are not due to the fact that the coefficients of $h^{(n)}$ cannot be properly represented using basis 2: The problems persist if one replaces the coefficients of $h^{(n)}$ by similar numbers exactly representable by machine numbers.

[2]We also tried to compare our algorithm with the standalone LP solver QSOpt_ex (*QSOpt_ex solver* [90]), which uses floating point arithmetic and then checks the result using exact arithmetic. The solver required < 1, and 9 seconds for checking feasibility of the $h^{(n)}$-systems with $n = 100, 200$, respectively. However, for all systems with $n > 200$ the tool crashed, throwing a floating point exception.

	$n = 100$	$n = 200$	$n = 400$	$n = 600$	$n = 1000$	$n = 1400$
Exact LP	1 sec	4 sec	23 sec	74 sec	450 sec	641 sec
Our algorithm	< 1 sec	< 1 sec	2 sec	5 sec	14 sec	33 sec

Table 6.1: Consistency checks for $h^{(n)}$-systems: Runtimes of different approaches.

evaluations in this section and Section 6.5 improved implementations of our algorithms and another testbed were used, including a newer version of Maple). Table 6.1 shows the results. Our algorithm clearly outperforms the LP approach. For more experiments see Section 6.5.

6.4 Approximating Extinction Probabilities with Inexact Arithmetic

We now turn to the problem of obtaining reliable upper and lower bounds for the extinction probabilities of an MFBP $\mathcal{B}$. We have shown in Theorem 10 that this problem is essentially equivalent to approximating the least nonnegative fixed point μ_f of a PSP f. It is shown in Etessami and Yannakakis [43] that such a μ_f may not be representable by radicals, so one can only approximate μ_f. We present an algorithm that solves the problem of computing *arbitrary precise bounds* (see Section 3.3) for the components of μ_f.

More precisely, the algorithm computes two sequences, $(\mathbf{lb}^{(i)})_{i\in\mathbb{N}}$ and $(\mathbf{ub}^{(i)})_{i\in\mathbb{N}}$, such that $\mathbf{lb}^{(i)} \leq \mu_f \leq \mathbf{ub}^{(i)}$ and $\lim_{i\to\infty} \mathbf{ub}^{(i)} - \mathbf{lb}^{(i)} = \overline{0}$. In words: $\mathbf{lb}^{(i)}$ and $\mathbf{ub}^{(i)}$ are lower and upper bounds on μ_f, respectively, and the sequences converge to μ_f. Moreover, they converge linearly, meaning that the *number of accurate bits* of $\mathbf{lb}^{(i)}$ and $\mathbf{ub}^{(i)}$ are linear functions of i. (The number of accurate bits of a vector $\mathbf{x}$ is defined as the greatest number k such that $|(\mu_f - \mathbf{x})_j|/|(\mu_f)_j| \leq 2^{-k}$ holds for all $j \in \{1, \ldots, n\}$.) These properties are guaranteed even though our algorithm uses inexact arithmetic: Our algorithm detects numerical problems due to rounding errors, recovers from them, and increases the precision of the arithmetic as needed. Increasing the precision dynamically is, e.g., supported by the GMP library (*GMP library* [46]) and computer algebra systems like Maple.

To approximate the least fixed point of a PSP, we first transform it into a certain normal form:

Definition 47. (Perfectly superlinar PSP)
A purely superlinear PSP f is called *perfectly superlinear* if every variable depends directly on itself and every superlinear SCC is purely superlinear. $\square$

The following theorem states that any PSP f can be made perfectly superlinear (the proof can be found in Appendix C):

Theorem 13. *(Normal form of PSPs)*
Let f be a PSP of size s. We can compute in time $O(n \cdot s)$ a perfectly superlinear PSP $\tilde{f}$ with $Var(\tilde{f}) = Var(f) \cup \{\tilde{X}\}$ of size $O(n \cdot s)$ such that $\mu_f = (\mu_{\tilde{f}})_{Var(f)}$. $\square$

Precision and Floating Assignments. We first explain our approach of combining exact and inexact arithmetic in more detail. Consider an elementary operation g, like multiplication, subtraction, etc., that operates on two input numbers x and y. The operation g could also return the first component of the solution of a nonsingular linear equation system, where the inputs $\mathbf{x}$ are the coefficients of the linear system. We can *compute $g(x, y)$ with increasing precision* if there is a procedure that on input x, y outputs a sequence $g^{(1)}(x, y), g^{(2)}(x, y), \ldots$ that converges to $g(x, y)$. We do not assume any requirements on the convergence speed of this procedure, in particular, we do not require that there is an i with $g^{(i)}(x, y) = g(x, y)$. This procedure, which we assume exists, allows to implement *floating assignments* of the form

$$z \leftsquigarrow g(x, y) \textbf{ such that } \phi(z)$$

with the following semantics: z is assigned the value $g^{(i)}(x, y)$, where $i \geq 1$ is the smallest index such that $\phi(g^{(i)}(x, y))$ holds. We say that the assignment is *valid* if $\phi(g(x, y))$ holds and ϕ involves only continuous functions (in the calculus sense) and strict inequalities. Our assumption on the arithmetic guarantees that (the computation underlying) a valid floating assignment terminates. As "syntactic sugar", more complex operations (e.g., linear equation solving) are also allowed

in floating assignments, as long as they can be decomposed into elementary operations.
In our opinion any implementation of arbitrary precision arithmetic should satisfy our requirement that the computed values converge to the exact result. For supporting this we cite the documentation of the GMP library (*GMP library* [46]): “Each function is defined to calculate with ‘infinite precision’ followed by a truncation to the destination precision, but of course the work done is only what’s needed to determine a result under that definition.”

Structure of the algorithm. Our algorithm receives as input a perfectly superlinear PSP f and an error bound $\epsilon > 0$, and returns vectors $\mathbf{lb}, \mathbf{ub}$ such that $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ and $\mathbf{ub} - \mathbf{lb} \leq \overline{\epsilon}$. A first initialization step requires us to compute a vector $\mathbf{x}$ with $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x})$, i.e., a “strict” pre-fixed point. This is done in the following section. In Section 6.4.2 we present the algorithm itself. Proofs of several properties are divided into Sections 6.4.2.1 – 6.4.2.4. The concluding correctness proof is given in Section 6.4.2.5.

6.4.1 Computing a Strict Pre-Fixed Point

Algorithm 2: Procedure `computeStrictPrefix`

```
Input: perfectly superlinear PSP $f$
Output: $\mathbf{x}$ with $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x}) \prec \overline{1}$
$\mathbf{x} \leftarrow \overline{0}$;
while $\overline{0} \not\prec \mathbf{x}$ do
    $Z \leftarrow \{i \mid 1 \leq i \leq n, f_i(\mathbf{x}) = 0\}$;
    $P \leftarrow \{i \mid 1 \leq i \leq n, f_i(\mathbf{x}) > 0\}$;
    $\mathbf{y}_Z \leftarrow \overline{0}$;
    $\mathbf{y}_P \leftarrow\!\!\!\sim f_P(\mathbf{x})$ such that $\overline{0} \prec \mathbf{y}_P \prec f_P(\mathbf{y}) \prec \overline{1}$;
    $\mathbf{x} \leftarrow \mathbf{y}$;
```

Algorithm 2 uses a floating assignment $\mathbf{y}_P \leftarrow\!\!\!\sim f_P(\mathbf{x})$, although it must also perform *exact* comparisons to obtain the sets Z and P and to decide exactly whether $\mathbf{y}_P \prec f_P(\mathbf{y})$ holds in the **such that** clause of the floating assignment. This is because, in spite of us performing such operations exactly, we do not want to use the *result* of exact computations as input for other computations, as this easily leads to an explosion in the required precision. For instance, the size of the exact result of $f_P(\mathbf{y})$ may be larger than the size of $\mathbf{y}$, while an approximation of smaller size may already satisfy the **such that** clause. In order to emphasize this, we *never* store the result of an exact numerical computation in a variable.

Theorem 14. *Algorithm 2 is correct and terminates after at most n iterations.* □

Proof.
We will prove the following invariant of the algorithm (i is the number of already performed loop iterations):

(a) $\overline{0} \leq \mathbf{x} \leq f(\mathbf{x})$;

(b) for all components j with $(f^i(\overline{0}))_j > 0$ we have $0 < \mathbf{x}_j < f_j(\mathbf{x})$.

The invariant implies that the loop terminates after at most n iterations, because $f^n(\overline{0}) \succ \overline{0}$ holds as $\mu_f \succ \overline{0}$. The invariant also implies that we have $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x})$ after the loop terminates.
So it remains to show the invariant. Part (a) clearly holds throughout the loop because for all components j either $0 = \mathbf{x}_j$ holds or $0 < \mathbf{x}_j < f_j(\mathbf{x})$ is guaranteed by the floating assignment. Assume inductively that the invariant holds after $i \geq 0$ iterations. It suffices to prove that (b) holds after $i+1$ iterations. Let $\mathbf{x}^{(i)}$ denote the value of $\mathbf{x}$ after i iterations. Let $(f^{i+1}(\overline{0}))_j > 0$. Then $f_j(\overline{0}) > 0$ or there is a monomial in f_j which consists only of variables X_k with $(f^i(\overline{0}))_k > 0$. In the second case we have, by induction hypothesis part (b), that $\mathbf{x}_k^{(i)} > 0$ holds for those variables X_k. In both cases it follows $f_j(\mathbf{x}^{(i)}) > 0$. Furthermore, we have by induction hypothesis part (a) that $f_j(\mathbf{x}^{(i)}) \leq f_j(f(\mathbf{x}^{(i)}))$ where, in fact, the inequality is strict because X_j depends on itself (as f is perfectly superlinear). We conclude that $0 < f_j(\mathbf{x}^{(i)}) < f_j(f(\mathbf{x}^{(i)}))$, and hence the floating assignment guarantees $0 < \mathbf{x}_j^{(i+1)} < f_j(\mathbf{x}^{(i+1)})$. So the invariant holds after $i+1$ iterations. □

6.4.2 Computing Lower and Upper Bounds

Algorithm 2 uses Kleene iteration $\overline{0}, f(\overline{0}), f(f(\overline{0})), \ldots$ to compute a strict pre-fixed point. One could, in principle, use the same scheme to compute lower bounds of μ_f, as this sequence converges to μ_f from below by Kleene's theorem. However, convergence of Kleene iteration is generally slow. It is shown in Etessami and Yannakakis [43] that for the 1-dimensional PSP f with $f(X) = 0.5X^2 + 0.5$ we have $\mu_f = 1$, and the i-th Kleene approximant $f^i(0)$ satisfies $f^i(0) \leq 1 - \frac{1}{i}$. Hence, Kleene iteration may converge only logarithmically, i.e., the number of accurate bits is a logarithmic function of the number of iterations.

In Etessami and Yannakakis [43] it was suggested to use Newton's method for faster convergence. In order to see how Newton's method can be used, observe that instead of computing μ_f, one can equivalently compute the least nonnegative zero of $f(\overline{X}) - \overline{X}$.

Given an approximant $\mathbf{x}$ of μ_f, Newton's method first computes $g^{(\mathbf{x})}(\overline{X})$, the first-order linearization of f at the point $\mathbf{x}$:

$$g^{(\mathbf{x})}(\overline{X}) = f(\mathbf{x}) + f'(\mathbf{x})(\overline{X} - \mathbf{x})$$

The next Newton approximant $\mathbf{y}$ is obtained by solving $\overline{X} = g^{(\mathbf{x})}(\overline{X})$; this motivates the definition of the *Newton operator*:

Definition 48. (Newton operator)
The *Newton operator* $\mathcal{N}_f$ is defined for $\mathbf{x} \in \mathbb{R}^n$ by

$$\mathcal{N}_f(\mathbf{x}) := \mathbf{x} + (Id - f'(\mathbf{x}))^{-1}(f(\mathbf{x}) - \mathbf{x}).$$

□

We usually drop the subscript of $\mathcal{N}_f$ if f is clear from the context. If $\boldsymbol{\nu}^{(0)} \leq \mu_f$ is any pre-fixed point of f, for instance $\boldsymbol{\nu}^{(0)} = \overline{0}$, we can define a *Newton sequence* $(\boldsymbol{\nu}^{(i)})_{i \in \mathbb{N}}$ by setting $\boldsymbol{\nu}^{(i+1)} = \mathcal{N}(\boldsymbol{\nu}^{(i)})$ for $i \geq 0$. It has been shown in Etessami and Yannakakis [43], Kiefer et al. [64] and Esparza et al. [37] that Newton sequences converge at least linearly to μ_f. Moreover, we have $\overline{0} \leq \boldsymbol{\nu}^{(i)} \leq f(\boldsymbol{\nu}^{(i)}) \leq \mu_f$ for all i.

These facts were shown only for Newton sequences that are computed exactly, i.e., without rounding errors. Unfortunately, Newton approximants are hard to compute exactly: since each iteration requires to solve a linear equation system whose coefficients depend on the results of the previous iteration, the size of

Algorithm 3: Procedure `calcBounds`

```
Input: perfectly superlinear PSP $f$, error bound $\epsilon > 0$
Output: vectors $\mathbf{lb}, \mathbf{ub}$ such that $\mathbf{lb} \le \mu_f \le \mathbf{ub}$ and $\mathbf{ub} - \mathbf{lb} \le \overline{\epsilon}$
$\mathbf{lb} \leftarrow$ computeStrictPrefix($f$);
$\mathbf{ub} \leftarrow \overline{1}$;
while $\mathbf{ub} - \mathbf{lb} \not\le \overline{\epsilon}$ do
    $\mathbf{x} \leftsquigarrow \mathcal{N}(\mathcal{N}(\mathbf{lb}))$ such that $f(\mathbf{lb}) + f'(\mathbf{lb})(\mathbf{x} - \mathbf{lb}) \prec \mathbf{x} \prec f(\mathbf{x}) \prec \overline{1}$;
    $\mathbf{lb} \leftarrow \mathbf{x}$;
    $Z \leftarrow \{i \mid 1 \le i \le n, f_i(\mathbf{ub}) = 1\}$;
    $P \leftarrow \{i \mid 1 \le i \le n, f_i(\mathbf{ub}) < 1\}$;
    $\mathbf{y}_Z \leftarrow \overline{1}$;
    $\mathbf{y}_P \leftsquigarrow f_P(f(\mathbf{ub}))$ such that $f_P(\mathbf{y}) \prec \mathbf{y}_P \prec f_P(\mathbf{ub})$;
    forall superlinear SCCs $S$ of $f$ with $\mathbf{y}_S = \overline{1}$ do
        $\mathbf{t} \leftarrow \overline{1} - \mathbf{lb}_S$;
        if $f'_{SS}(\overline{1})\mathbf{t} \succ \mathbf{t}$ then
            $\mathbf{y}_S \leftsquigarrow \overline{1} - \min\left\{1, \frac{\min_{i \in S}(f'_{SS}(\overline{1})\mathbf{t} - \mathbf{t})_i}{2 \cdot \max_{i \in S}(f_S(\overline{2}))_i}\right\} \cdot \mathbf{t}$ such that
            $f_S(\mathbf{y}) \prec \mathbf{y}_S \prec \overline{1}$;
    $\mathbf{ub} \leftarrow \mathbf{y}$;
```

the Newton approximants easily explodes. Therefore, we wish to use inexact arithmetic, but without losing the good properties of Newton's method (reliable lower bounds, linear convergence). Algorithm 3 accomplishes these goals, and additionally computes post-fixed points **ub** of f, which are upper bounds on μ_f. In the following we first give an overview over the parts of the algorithm.

Computing lower bounds. The lower bounds are stored in the variable **lb**. The first value of **lb** is not simply $\overline{0}$, but is computed by `computeStrictPrefix`(f), in order to guarantee the validity of the following floating assignments. We use Newton's method for improving the lower bounds because it converges fast (at least linearly) when performed exactly. In each iteration of the algorithm, *two* Newton steps are performed using inexact arithmetic. The intention is that two inexact Newton steps should improve the lower bound at least as much as one exact Newton step. While this may sound like a vague hope for small rounding errors, it can be rigorously proved thanks to the **such that** clause of the floating assignment in line 4. The proof involves two steps. The first step is to prove that $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$ is a (strict) post-fixed point of the function $g(\overline{X}) = f(\mathbf{lb}) + f'(\mathbf{lb})(\overline{X} - \mathbf{lb})$, i.e., $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$ satisfies the first inequality in the **such that** clause. For the second step, recall that $\mathcal{N}(\mathbf{lb})$ is the least fixed point of g. By Knaster-Tarski's theorem, $\mathcal{N}(\mathbf{lb})$ is also the least post-fixed point of g. So, our value $\mathbf{x}$, the inexact version of $\mathcal{N}(\mathcal{N}(\mathbf{lb}))$, satisfies $\mathbf{x} \geq \mathcal{N}(\mathbf{lb})$, and hence two inexact Newton steps are in fact at least as "fast" as one exact Newton step. Thus, the $\mathbf{lb}$ converge linearly to μ_f.

Computing upper bounds. The upper bounds $\mathbf{ub}$ are post-fixed points, i.e., $f(\mathbf{ub}) \leq \mathbf{ub}$ is an invariant of the algorithm. The algorithm computes the sets Z and P so that inexact arithmetic is only applied to the components i with $f_i(\mathbf{ub}) < 1$. In the P-components, the function f is applied to $\mathbf{ub}$ in order to improve the upper bound. In fact, f is applied twice in line 9, analogously to applying $\mathcal{N}$ twice in line 4. Here, the **such that** clause makes sure that the progress towards μ_f is at least as fast as the progress of one exact application of f would be. One can show that this leads to linear convergence to μ_f.

Obtaining a post-fixed point $\prec \overline{1}$. The rest of the algorithm (lines 10-13) deals with the problem that, given a post-fixed $\mathbf{ub}$, the sequence

$$\mathbf{ub}, f(\mathbf{ub}), f(f(\mathbf{ub})), \ldots$$

does not necessarily converge to μ_f. For instance, if $f(X) = 0.75X^2 + 0.25$, then $\mu_f = 1/3$, but $1 = f(1) = f(f(1)) = \cdots$. Therefore, the if-statement of Algorithm 3 allows to improve the upper bound from $\overline{1}$ to a post-fixed point less than $\overline{1}$, by exploiting the lower bounds $\mathbf{lb}$. This is illustrated in Figure 6.4 for a 2-dimensional scPSP f. The dotted lines indicate the curve of the points (X_1, X_2) satisfying $X_1 = 0.8X_1X_2 + 0.2$ and $X_2 = 0.4X_1^2 + 0.1X_2 + 0.5$. Notice that $\mu_f \prec \overline{1} = f(\overline{1})$. In Figure 6.4 (a) the shaded area consists of those points $\mathbf{lb}$ where $f'(\overline{1})(\overline{1} - \mathbf{lb}) \succ \overline{1} - \mathbf{lb}$ holds, i.e., the condition of line 12. One can show that μ_f must lie in the shaded area, so by continuity, any sequence converging to μ_f, in particular the sequence of lower bounds $\mathbf{lb}$, finally reaches the shaded area. In

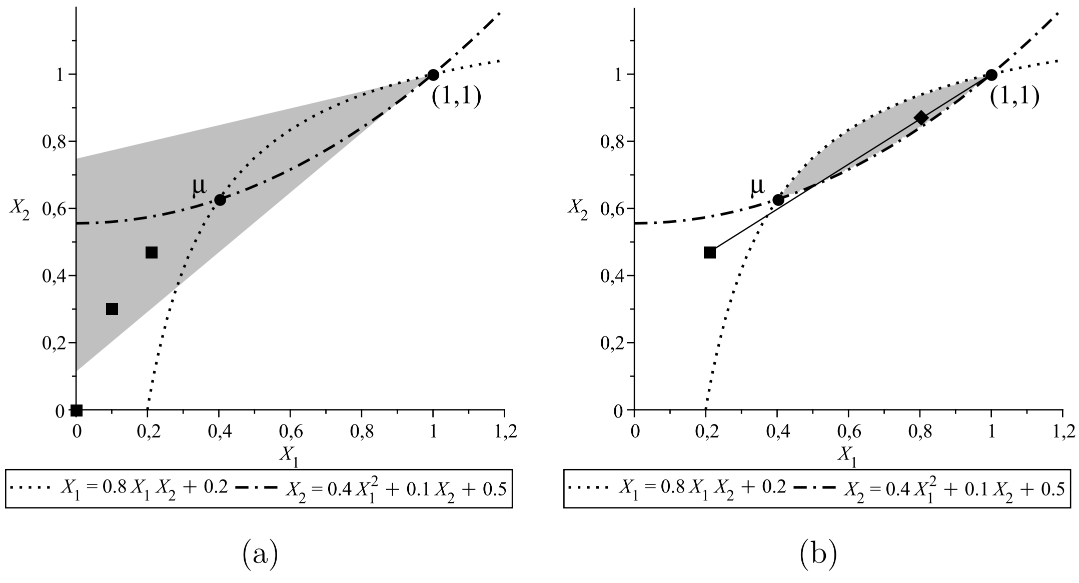


Figure 6.4: Computation of a post-fixed point less than $\overline{1}$.

Figure 6.4 (a) this is indicated by the points with the square shape. Figure 6.4 (b) shows how to exploit such a point **lb** to compute a post-fixed point $\mathbf{ub} \prec \overline{1}$ (post-fixed points are shaded in Figure 6.4 (b)): The post-fixed point **ub** (diamond shape) is obtained by starting at $\overline{1}$ and moving a little bit along the straight line between $\overline{1}$ and **lb**, cf. line 13. The sequence $\mathbf{ub}, f(\mathbf{ub}), f(f(\mathbf{ub})), \ldots$ now converges linearly to μ_f. Putting all together we obtain:

Theorem 15. *(Correctness of Algorithm 3)*
Algorithm 3 terminates and computes vectors $\mathbf{lb}, \mathbf{ub}$ *such that* $\mathbf{lb} \leq \mu_f \leq \mathbf{ub}$ *and* $\mathbf{ub} - \mathbf{lb} \leq \overline{\epsilon}$. *Moreover, the sequences of lower and upper bounds computed by the algorithm both converge linearly to* μ_f. □

Notice that Theorem 15 is about the convergence speed of the approximants, not about the time needed to compute them. To analyze the computation time, one would need stronger requirements on how floating assignments are performed. In summary this gives us a method to compute lower and upper bounds for the extinction probabilities ψ of a MFBP $\mathcal{B}$: We first construct the PSP f corresponding to $\mathcal{B}$ according to Theorem 10, then we normalize f and use it as input for Algorithm 3.

We divide the proof of Theorem 15 into several sections corresponding to the paragraphs in the explanation above. Missing proofs of auxiliary lemmas can be found in Appendix C. We first have to prove some additional properties of the computed sequences.

6.4.2.1 Characterizing Pre-Fixed Points and Post-Fixed Points

The lower and upper bounds computed by Algorithm 3 have a special feature: they satisfy $\mathbf{lb} \prec f(\mathbf{lb})$ and $\mathbf{ub} \geq f(\mathbf{ub})$. The following theorem guarantees that such points are in fact lower and upper bounds.

Theorem 16.
Let f be a perfectly superlinear PSP. Let $\overline{0} \leq \mathbf{x} \leq \overline{1}$. If $\mathbf{x} \prec f(\mathbf{x})$, then $\mathbf{x} \prec \mu_f$. If $\mathbf{x} \geq f(\mathbf{x})$, then $\mathbf{x} \geq \mu_f$. $\square$

In particular we need Theorem 16 for proving that Algorithm 3 computes correct results (i.e. for the proof of Theorem 15). It also provides the user a way of verifying that the computed bounds are indeed correct.
For proving Theorem 16 we start by characterizing the special case of linear PSPs:

Lemma 9.
Let f be a linear PSP. Then μ_f is the unique fixed point of f. $\square$

Proof.
Recall that $\mu_f \succ \overline{0}$. Assume that there is a fixed point $\mathbf{x}$ of f different from μ_f. Then, by the linearity of f, all points on the straight line through $\mathbf{x}$ and μ_f are fixed points of f. So there is a point $\mathbf{y} \geq \overline{0}$ on this straight line with $\mathbf{y}_i = 0$ for some $i \in \{1, \ldots, n\}$. This contradicts the fact that μ_f is the *least* fixed point of f. $\square$

The following lemma shows how to compute a post-fixed point that satisfies certain properties and is arbitrarily close to $\overline{1}$.

Lemma 10.
Let f be a perfectly superlinear PSP. Let $r \in \mathbb{R}$ with $0 < r < 1$. Let $\mathbf{x} = \mu_f + r(\overline{1} - \mu_f)$. Then $f(\mathbf{x}) \leq \mathbf{x}$. Furthermore, let $\mathbf{p} = f^n(\mathbf{x})$. Then $f(\mathbf{p}) \leq \mathbf{p}$ and $f_i(\mathbf{p}) < \mathbf{p}_i$ holds for all $i \in \{1, \ldots, n\}$ with $(\mu_f)_i < 1$. $\square$

Proof.
Letting $\mathbf{u}, \mathbf{v}$ be any vectors, we write $f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f'(\mathbf{u})\mathbf{v} + R(\mathbf{u}, \mathbf{v})$ for the Taylor expansion of f at $\mathbf{u}$. Then we have

$$\begin{aligned}&\mu_f + f'(\mu_f)(\overline{1} - \mu_f) + R(\mu_f, \overline{1} - \mu_f) \\ &= f(\mu_f + (\overline{1} - \mu_f)) = f(\overline{1}) \leq \overline{1} = \mu_f + (\overline{1} - \mu_f)\ ,\end{aligned}$$

so it follows $f'(\mu_f)\mathbf{d} + R(\mu_f, \mathbf{d}) \leq \mathbf{d}$ where $\mathbf{d} := \overline{1} - \mu_f$. Moreover, we have

$$\begin{aligned}f(\mathbf{x}) &= f(\mu_f + r\mathbf{d}) = f(\mu_f) + f'(\mu_f)r\mathbf{d} + R(\mu_f, r\mathbf{d}) \\ &= \mu_f + rf'(\mu_f)\mathbf{d} + R(\mu_f, r\mathbf{d}) \\ &\leq \mu_f + rf'(\mu_f)\mathbf{d} + rR(\mu_f, \mathbf{d}) && \text{(since } R(\mu_f, \cdot) \text{ is superlinear)} \\ &= \mu_f + r(f'(\mu_f)\mathbf{d} + R(\mu_f, \mathbf{d})) \\ &\leq \mu_f + r\mathbf{d} && \text{(from above)} \\ &= \mathbf{x}\ ,\end{aligned}$$

i.e., $\mathbf{x}$ is a post-fixed point, hence $\mu_f \leq \mathbf{x} \leq \overline{1}$. Consider the sequence $\mathbf{x} \geq f(\mathbf{x}) \geq f(f(\mathbf{x})) \geq \cdots$. Since every component depends directly on itself, we have for all components i that once $(f^j(\mathbf{x}))_i > (f^{j+1}(\mathbf{x}))_i$ holds for some j, we have $(f^k(\mathbf{x}))_i > (f^{k+1}(\mathbf{x}))_i$ for all $k \geq j$. On the other hand, it is easy to see that if $(f^j(\mathbf{x}))_i > (f^{j+1}(\mathbf{x}))_i$ for some j, then $(f^k(\mathbf{x}))_i > (f^{k+1}(\mathbf{x}))_i$ for some $k \leq n$. It follows that $f_i(\mathbf{p}) < \mathbf{p}_i$ holds for all components i for which there exists j with $(f^j(\mathbf{x}))_i > (f^{j+1}(\mathbf{x}))_i$. It remains to show that for all components i with $(\mu_f)_i < 1$ there exists j with $(f^j(\mathbf{x}))_i > (f^{j+1}(\mathbf{x}))_i$. Assume for a contradiction that this does not hold. Choose a variable X_i that violates the property (i.e., $(\mu_f)_i < 1$ and $\mathbf{x}_i = (f^j(\mathbf{x}))_i$ for all j) so that all components in lower SCCs satisfy the property. Let S denote the SCC of X_i and let g be the PSP obtained by restricting f to the S-components and replacing all variables in lower SCCs by the constant 1. Then for all X_j on which X_i depends we have $(\mu_f)_j = 1$, and so $\mathbf{x}_j = 1$. Furthermore, we have $(\mu_f)_S \prec \mathbf{x}_S \prec \overline{1}$. Notice that Lemma 9 guarantees that g is not linear, since both $(\mu_f)_S$ and $\mathbf{x}_S$ are fixed points of g. Hence, g is superlinear. For any vectors $\mathbf{u}, \mathbf{v}$ we write $g(\mathbf{u}+\mathbf{v}) = g(\mathbf{u}) + g'(\mathbf{u})\mathbf{v} + T(\mathbf{u}, \mathbf{v})$ for the Taylor expansion of g at $\mathbf{u}$. We have:

$$\begin{aligned}
r\mathbf{d}_S &= \mathbf{x}_S - (\mu_f)_S \\
&= g(\mathbf{x}_S) - (\mu_f)_S \\
&= g((\mu_f)_S + r\mathbf{d}_S) - (\mu_f)_S \\
&= (\mu_f)_S + g'((\mu_f)_S)r\mathbf{d}_S + T((\mu_f)_S, r\mathbf{d}_S) - (\mu_f)_S \\
&= g'((\mu_f)_S)r\mathbf{d}_S + T((\mu_f)_S, r\mathbf{d}_S).
\end{aligned}$$

Moreover, as $\mathbf{d}_S \succ \overline{0}$ and g is superlinear, the following inequality is strict in at least one component:

$$\begin{aligned}
g(\overline{1}) &= g\left((\mu_f)_S + \frac{1}{r}r\mathbf{d}\right) \\
&= g((\mu_f)_S) + g'((\mu_f)_S)\frac{1}{r}r\mathbf{d}_S + T\left((\mu_f)_S, \frac{1}{r}r\mathbf{d}\right) \\
&\geq (\mu_f)_S + \frac{1}{r}g'((\mu_f)_S)r\mathbf{d}_S + \frac{1}{r^2}T\left((\mu_f)_S, r\mathbf{d}\right) \\
&> (\mu_f)_S + \frac{1}{r}\left(g'((\mu_f)_S)r\mathbf{d}_S + T\left((\mu_f)_S, r\mathbf{d}\right)\right) \\
&= (\mu_f)_S + \frac{1}{r}r\mathbf{d}_S \qquad \text{(as computed above)} \\
&= (\mu_f)_S + \mathbf{d}_S = \overline{1}
\end{aligned}$$

This is the desired contradiction as $g(\overline{1}) \leq \overline{1}$ should hold since g is a PSP. □

6.4.2.2 Computing Upper Bounds

The following lemma is used for the proof of Theorem 16, but will also be essential to prove the convergence statement of Theorem 15: it states that, given a post-fixed point $\mathbf{p} \leq \overline{1}$, the Kleene sequence "from above" $\mathbf{p}, f(\mathbf{p}), f(f(f(\mathbf{p}))), \ldots$ converges linearly to μ_f:

Lemma 11. *Let f be a perfectly superlinear PSP. Let $f(\mathbf{p}) \leq \mathbf{p} \leq \overline{1}$ and $f_i(\mathbf{p}) < \mathbf{p}_i$ for all $i \in \{1, \ldots, n\}$ with $(\mu_f)_i < 1$. Then the sequence $(\mathbf{p}^{(i)})_{i\in\mathbb{N}}$ defined by*

$$\mathbf{p}^{(1)} := \mathbf{p} \textit{ and } \mathbf{p}^{(i+1)} := f(\mathbf{p}^{(i)}) \textit{ for } i \geq 1$$

converges linearly to μ_f. $\square$

Proof.
If $(\mu_f)_i = 1$ then $(\mu_f)_j = 1$ has to hold for every component j on which i depends. As μ_f is the least post-fixed point by Knaster-Tarski's theorem, we have $(f^k(\mathbf{p}))_i = 1$ for every $k \in \mathbb{N}$. Hence we can ignore the 1-components in our convergence proof and assume w.l.o.g. that $\mu_f \prec \overline{1}$ and with the assumptions $f(\mathbf{p}) \prec \mathbf{p}$. By the monotonicity of f and since every variable depends on itself, we get by a simple induction that $\mathbf{p}^{(i)} \succ \mathbf{p}^{(i+1)} \succ \mu_f$ for all $i \in \mathbb{N}$. This already shows that $(\mathbf{p}^{(i)})_{i\in\mathbb{N}}$ converges to some limit point. For every $\mathbf{u} \succ \mu_f$ with $\mathbf{u} \succ f(\mathbf{u})$ we write $\mathbf{u} = \mu_f + \mathbf{\Delta_u}$ and get:

$$\begin{aligned} f(\mathbf{u}) - \mu_f &= f(\mu_f + \mathbf{\Delta_u}) - \mu_f \\ &= f(\mu_f) + f'(\mu_f)\mathbf{\Delta_u} + R(\mu_f, \mathbf{\Delta_u}) - \mu_f \qquad \text{(Taylor expansion)} \end{aligned}$$

Since $R(\mu_f, \mathbf{\Delta_u})$ depends at least quadratically on $\mathbf{\Delta_u}$, one can write $R(\mu_f, \mathbf{\Delta_u}) = \tilde{R}(\mu_f, \mathbf{\Delta_u}) \cdot \mathbf{\Delta_u}$ for a nonnegative matrix $\tilde{R}(\mu_f, \mathbf{\Delta_u})$. Continuing the above equaliy, we obtain:

$$\begin{aligned} &= (f'(\mu_f) + \tilde{R}(\mu_f, \mathbf{\Delta_u}))\mathbf{\Delta_u} \\ &\prec \mathbf{\Delta_u} \qquad \text{(as } \mathbf{u} \succ f(\mathbf{u}).\text{)} \end{aligned}$$

Define $A(\mathbf{u}) := f'(\mu_f) + \tilde{R}(\mu_f, \mathbf{\Delta_u})$, so that we obtain

$$f(\mathbf{u}) - \mu_f = A(\mathbf{u}) \cdot \mathbf{\Delta_u} \prec \mathbf{\Delta_u}. \tag{6.3}$$

This holds especially for $\mathbf{u} = \mathbf{p}$. From the $\prec$-inequality in (6.3) follows that there exists $0 < \delta < 1$ such that

$$f(\mathbf{p}) - \mu_f = A(\mathbf{p})\mathbf{\Delta_p} \leq \delta\mathbf{\Delta_p}. \tag{6.4}$$

We now show for every $i \geq 0$ that $\mathbf{p}^{(i)} - \mu_f = \mathbf{\Delta_{p^{(i)}}} \leq \delta^i \mathbf{\Delta_p}$ by induction over i. This implies the linear convergence of $(\mathbf{p}^{(i)})_{i\in\mathbb{N}}$. The base case $i = 1$ is proved by (6.4). For $i > 1$ note that if $\overline{0} \leq \mathbf{u} \leq \mathbf{u}'$ and $\overline{0} \leq \mathbf{v} \leq \mathbf{v}'$, we have

$A(\mathbf{u})\mathbf{v} \leq A(\mathbf{u}')\mathbf{v}'$, since $A(\mathbf{u})$ is nonnegative if $\mathbf{u}$ is nonnegative.

$$\begin{aligned}
f^i(\mathbf{p}) - \mu_f &= f(\mathbf{p}^{(i-1)}) - \mu_f \\
&= A(\mathbf{p}^{(i-1)})(\mathbf{p}^{(i-1)} - \mu_f) && \text{(by (6.3))} \\
&\leq A(\mathbf{p})(\delta^{i-1}\boldsymbol{\Delta}_{\mathbf{p}}) && \text{(induction hypothesis)} \\
&= \delta^{i-1}A(\mathbf{p})\boldsymbol{\Delta}_{\mathbf{p}} \\
&\leq \delta^{i}\boldsymbol{\Delta}_{\mathbf{p}}. && (6.4)
\end{aligned}$$

□

Now we can prove Theorem 16.

Proof (of Theorem 16).
By Knaster-Tarski's theorem, μ_f is the least post-fixed point; the final statement of the theorem follows. It remains to show the first statement. By choosing the number r from Lemma 10 large enough we can find a post-fixed point $\mathbf{y}$ with $\mathbf{x} \prec \mathbf{y} \leq \bar{1}$. By Lemma 10 and Lemma 11 the sequence $\mathbf{y}, f(\mathbf{y}), f(f(\mathbf{y})), \ldots$ converges to μ_f. On the other hand, by repeatedly applying f to both sides of the inequality $\mathbf{x} \prec \mathbf{y}$ we obtain that $\mathbf{x} \prec f(\mathbf{x}) \leq f^i(\mathbf{x}) \leq f^i(\mathbf{y})$ holds for all $i \geq 0$. Since $(f^i(\mathbf{y}))_{i\in\mathbb{N}}$ converges to μ_f, we have $\mathbf{x} \prec \mu_f$. □

After characterizing pre-fixed points and post-fixed points, we now prove the remaining propositions corresponding to the sections in the explanation of the algorithm.

6.4.2.3 Computing Lower Bounds

The following lemma was essentially proved in Esparza et al. [37], Etessami and Yannakakis [43] (for the sake of completeness we give a proof in Appendix C):

Lemma 12.
Let f be a perfectly superlinear PSP and let $\mathbf{x} \prec f(\mathbf{x})$. Then

$$\mathcal{N}(\mathbf{x}) = \mathbf{x} + (f'(\mathbf{x}))^*(f(\mathbf{x}) - \mathbf{x}).$$

□

Lemma 12 allows to replace the matrix inverse $(Id - f'(\mathbf{x}))^{-1}$ with the matrix star $f'(\mathbf{x})^*$ as long as $\mathbf{x} \prec f(\mathbf{x})$ holds, which will be true whenever we compute $\mathcal{N}(\mathbf{x})$. The following lemmas are needed to show the validity of the floating assignment in line 4 of Algorithm 3, i.e. for proving that, for every computed $\mathbf{lb}$, $\mathbf{z} = \mathcal{N}(\mathcal{N}(\mathbf{lb}))$ satisfies

$$f(\mathbf{lb}) + f'(\mathbf{lb})(\mathbf{z} - \mathbf{lb}) \prec \mathbf{z} \prec f(\mathbf{z}) \prec \overline{1}.$$

First we state that the Newton operator preserves strict pre-fixed points:

Lemma 13.
Let f be perfectly superlinear. Let $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x}) \prec \overline{1}$ and $\mathbf{y} = \mathcal{N}(\mathbf{x})$. Then $f(\mathbf{x}) \prec \mathbf{y} \prec f(\mathbf{y}) \prec \overline{1}$. □

For the other condition of the floating assignment in line 4, we state that for a strict pre-fixed point $\mathbf{x}$ of f, $\mathcal{N}(\mathcal{N}(\mathbf{x}))$ is a *strict post-fixed* point of the linearization of f at $\mathbf{x}$:

Lemma 14.
Let f be perfectly superlinear. Let $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x})$. Let $\mathbf{z} = \mathcal{N}(\mathcal{N}(\mathbf{x}))$. Then $f(\mathbf{x}) + f'(\mathbf{x})(\mathbf{z} - \mathbf{x}) \prec \mathbf{z}$. □

Proof.
Recall hat for vectors $\mathbf{u}, \mathbf{v}$, we write $f(\mathbf{u} + \mathbf{v}) = f(\mathbf{u}) + f'(\mathbf{u})\mathbf{v} + R(\mathbf{u}, \mathbf{v})$ for the Taylor expansion of f at $\mathbf{u}$. We write $\mathbf{y} = \mathcal{N}(\mathbf{x})$ and $\boldsymbol{\Delta} = f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x})$. Notice that $\mathbf{y} = \mathbf{x} + \boldsymbol{\Delta}$. We have

$$\begin{aligned}
\mathbf{z} &= \mathbf{y} + f'(\mathbf{y})^*(f(\mathbf{y}) - \mathbf{y}) \\
&= \mathbf{x} + \boldsymbol{\Delta} + f'(\mathbf{y})^*(f(\mathbf{x} + \boldsymbol{\Delta}) - \mathbf{x} - \boldsymbol{\Delta}) \\
&= \mathbf{x} + \boldsymbol{\Delta} + f'(\mathbf{y})^*(f(\mathbf{x}) + f'(\mathbf{x})\boldsymbol{\Delta} + R(\mathbf{x}, \boldsymbol{\Delta}) - \mathbf{x} - \boldsymbol{\Delta}) \\
&= \mathbf{x} + \boldsymbol{\Delta} + f'(\mathbf{y})^*((f(\mathbf{x}) - \mathbf{x}) + f'(\mathbf{x})f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x}) - \boldsymbol{\Delta} + R(\mathbf{x}, \boldsymbol{\Delta})) \\
&= \mathbf{x} + \boldsymbol{\Delta} + f'(\mathbf{y})^*(\boldsymbol{\Delta} - \boldsymbol{\Delta} + R(\mathbf{x}, \boldsymbol{\Delta})) \\
&= \mathbf{x} + \boldsymbol{\Delta} + f'(\mathbf{y})^* R(\mathbf{x}, \boldsymbol{\Delta}) \; .
\end{aligned}$$

It follows

$$\begin{aligned}
f(\mathbf{x}) + f'(\mathbf{x})(\mathbf{z}-\mathbf{x}) &= f(\mathbf{x}) + f'(\mathbf{x})\left(\mathbf{\Delta} + f'(\mathbf{y})^* R(\mathbf{x},\mathbf{\Delta})\right) \\
&= \mathbf{x} + (f(\mathbf{x})-\mathbf{x}) + f'(\mathbf{x})f'(\mathbf{x})^*(f(\mathbf{x})-\mathbf{x}) + f'(\mathbf{x})f'(\mathbf{y})^* R(\mathbf{x},\mathbf{\Delta}) \\
&= \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{x})f'(\mathbf{y})^* R(\mathbf{x},\mathbf{\Delta}) \\
&\le \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{y})f'(\mathbf{y})^* R(\mathbf{x},\mathbf{\Delta}) \\
&\prec \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{y})f'(\mathbf{y})^* R(\mathbf{x},\mathbf{\Delta}) + R(\mathbf{x},\mathbf{\Delta}) \\
&= \mathbf{x} + \mathbf{\Delta} + f'(\mathbf{y})^* R(\mathbf{x},\mathbf{\Delta}) \\
&= \mathbf{z}\ .
\end{aligned}$$

For the $\prec$-inequality in this inequality chain, notice that, since $\mathbf{x} \prec f(\mathbf{x})$, we have $\mathbf{\Delta} \succ \overline{0}$, and since f is purely superlinear, we have $R(\mathbf{x},\mathbf{\Delta}) \succ \overline{0}$. □

For showing linear convergence, we show that $\mathcal{N}(\mathbf{x})$ is the *least* post-fixed point of the linearization of f at $\mathbf{x}$:

Lemma 15.
Let f be a PSP. Let $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x})$. Let $\mathbf{z}$ be with $f(\mathbf{x}) + f'(\mathbf{x})(\mathbf{z}-\mathbf{x}) \le \mathbf{z}$, i.e. $\mathbf{z}$ is a strict post-fixed point of the linearization of f at $\mathbf{x}$. Then $\mathcal{N}(\mathbf{x}) \le \mathbf{z}$ holds. □

We get that the computed $\mathbf{x}$ in the floating assignment of line 4 is at least as precise as $\mathcal{N}(\mathbf{x})$: $\mathbf{x}$ is a strict post-fixed point of the linearization of f at $\mathbf{lb}$, and $\mathcal{N}(\mathbf{lb})$ is the *least* such post-fixed point.

6.4.2.4 Obtaining a Post-Fixed Point $\prec \overline{1}$

The following lemma states the validity of the floating assignment in line 13 of Algorithm 3:

Lemma 16.
Let f be a purely superlinear PSP. Let $\overline{0} \prec \mathbf{t} \prec \overline{1}$ such that $f'(1)\mathbf{t} \succ \mathbf{t}$. Let

$$\mathbf{y} = \overline{1} - \min\left\{1, \frac{\min_{i\in\{1,\dots,n\}}(f'(\overline{1})\mathbf{t}-\mathbf{t})_i}{2\cdot\max_{i\in\{1,\dots,n\}}(f(\overline{2}))_i}\right\}\cdot\mathbf{t}\ .$$

Then $f(\mathbf{y}) \prec \mathbf{y} \prec \overline{1}$. □

Proof.
Recall that for vectors $\mathbf{u}, \mathbf{v}$, we write $f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f'(\mathbf{u})\mathbf{v} + R(\mathbf{u}, \mathbf{v})$ for the Taylor expansion of f at $\mathbf{u}$. Let $r = \min\left\{1, \frac{\min_{i\in\{1,\ldots,n\}}(f'(\overline{1})\mathbf{t}-\mathbf{t})_i}{2\cdot\max_{i\in\{1,\ldots,n\}}(f(\overline{2}))_i}\right\}$. Then we have:

$$\begin{aligned}
R(\overline{1}, -r\mathbf{t}) &\leq R(\overline{1}, r\mathbf{t}) \\
&\leq r^2 R(\overline{1}, \mathbf{t}) && \text{(degree of } R(\overline{1}, \cdot) \text{ at least 2)} \\
&\leq r^2 R(\overline{1}, \overline{1}) && (\mathbf{t} \leq \overline{1}) \\
&\leq r^2 f(\overline{2}) && (f(\overline{1}+\overline{1}) = f(\overline{1}) + f'(\overline{1})\overline{1} + R(\overline{1}, \overline{1})) \\
&\leq r^2 \max_{i\in\{1,\ldots,n\}}(f(\overline{2}))_i \cdot \overline{1} \\
&\leq r \cdot \frac{\min_{i\in\{1,\ldots,n\}}(f'(\overline{1})\mathbf{t}-\mathbf{t})_i}{2} \cdot \overline{1} && \text{(definition of } r) \\
&\leq \frac{r}{2} \cdot (f'(\overline{1})\mathbf{t} - \mathbf{t}) \\
&\prec r \cdot (f'(\overline{1})\mathbf{t} - \mathbf{t}) && (f'(\overline{1})\mathbf{t} \succ \mathbf{t})
\end{aligned}$$

Using this inequality we obtain

$$\begin{aligned}
f(\overline{1} - r\mathbf{t}) &= f(\overline{1}) + f'(\overline{1}) \cdot (-r\mathbf{t}) + R(\overline{1}, -r\mathbf{t}) \\
&\leq 1 - rf'(\overline{1})\mathbf{t} + R(\overline{1}, -r\mathbf{t}) && (f(\overline{1}) \leq \overline{1}) \\
&\prec 1 - rf'(\overline{1})\mathbf{t} + r \cdot (f'(\overline{1})\mathbf{t} - \mathbf{t}) && \text{(see above)} \\
&= 1 - r\mathbf{t}.
\end{aligned}$$

□

We still need two technical lemmas.

Lemma 17.
Let f be perfectly superlinear. Let $\mathbf{y} = f(\mathbf{x}) \leq \mathbf{x}$ and $f_i(\mathbf{x}) < \mathbf{x}_i$ for some $i \in \{1, \ldots, n\}$. Then $f(\mathbf{y}) \leq \mathbf{y}$ and $f_i(\mathbf{y}) < \mathbf{y}_i$. □

Proof.
We have $f(\mathbf{y}) = f(f(\mathbf{x})) \leq f(\mathbf{x}) = \mathbf{y}$ by the monotonicity of f. Moreover, since

each component depends on itself, the strict inequality $f_i(\mathbf{x}) < \mathbf{x}_i$ implies the strict inequality $f_i(f(\mathbf{x})) < f_i(\mathbf{x}) = \mathbf{y}_i$. □

To show that $\mathbf{ub}_i < 1$ eventually holds in the components i with $(\mu_f)_i < 1$, we prove

Lemma 18.
Let f be a purely superlinear PSP and $\mathbf{x} \geq \overline{0}, \mathbf{u} \succ \overline{0}$. Then

$$f'(\mathbf{x}+\mathbf{u})\mathbf{u} \succ f(\mathbf{x}+\mathbf{u}) - f(\mathbf{x}).$$

□

Proof.
It suffices to show $f_i(\mathbf{x}) - f_i(\mathbf{x}+\mathbf{u}) + (f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i > 0$ for every component i of f. We can write $f_i(\mathbf{x}+\mathbf{u})$ as $f_i(\mathbf{x}) + \int_0^1 (f'(\mathbf{x}+s\mathbf{u})\mathbf{u})_i\, ds$. Hence

$$\begin{aligned}
&f_i(\mathbf{x}) - f_i(\mathbf{x}+\mathbf{u}) + (f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i \\
&= f_i(\mathbf{x}) - f_i(\mathbf{x}) - \int_0^1 (f'(\mathbf{x}+s\mathbf{u})\mathbf{u})_i\, ds + (f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i \\
&= -\int_0^1 (f'(\mathbf{x}+s\mathbf{u})\mathbf{u})_i\, ds + (f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i \\
&= -\int_0^{1/2} (f'(\mathbf{x}+s\mathbf{u})\mathbf{u})_i\, ds - \int_{1/2}^1 (f'(\mathbf{x}+s\mathbf{u})\mathbf{u})_i\, ds + (f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i
\end{aligned}$$

For $0 \leq s \leq 1$, we have $(f'(\mathbf{x}+s\mathbf{u})\mathbf{u})_i \leq (f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i$, and for $0 \leq s \leq 1/2$, the inequality is strict, because $\mathbf{u} \succ \overline{0}$ and f is purely superlinear. Hence

$$\begin{aligned}
&-\int_0^{1/2} (f'(\mathbf{x}+s\mathbf{u})\mathbf{u})_i\, ds - \int_{1/2}^1 (f'(\mathbf{x}+s\mathbf{u})\mathbf{u})_i\, ds + (f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i \\
&> -\int_0^{\frac{1}{2}} (f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i\, ds - \int_{\frac{1}{2}}^1 (f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i\, ds + (f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i \\
&= -(f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i + (f'(\mathbf{x}+\mathbf{u})\mathbf{u})_i = 0.
\end{aligned}$$

□

6.4.2.5 Concluding Correctness Proof

By combing all obtained results we can finally prove Theorem 15. We restate the claim:

Theorem 15. *(Correctness of Algorithm 3)*
Algorithm 3 terminates and computes vectors $\mathbf{lb}, \mathbf{ub}$ *such that* $\mathbf{lb} \le \mu_f \le \mathbf{ub}$ *and* $\mathbf{ub} - \mathbf{lb} \le \bar{\epsilon}$*. Moreover, the sequences of lower and upper bounds computed by the algorithm both converge linearly to* μ_f*.* □

Proof (of Theorem 15).
The validity of the floating assignment in line 4 follows from Lemma 13 and Lemma 14. Next we show the convergence of the lower bounds. Let $(\mathbf{lb}^{(k)})_{k\in\mathbb{N}}$ be the sequence of the lower bounds $\mathbf{lb}$ in the algorithm, where $\mathbf{lb}^{(1)}$ is the result of `computeStrictPrefix`(f). Moreover, define an “exact” Newton sequence $\boldsymbol{\nu}^{(1)} = \mathbf{lb}^{(1)}$ and $\boldsymbol{\nu}^{(k+1)} = \mathcal{N}(\boldsymbol{\nu}^{(k)})$. We prove by induction that $\boldsymbol{\nu}^{(k)} \le \mathbf{lb}^{(k)}$. The induction base ($k = 1$) is trivial. Let $k \ge 1$. Notice that the floating assignment in line 4 guarantees $f(\mathbf{lb}^{(k)}) + f'(\mathbf{lb}^{(k)})\left(\mathbf{lb}^{(k+1)} - \mathbf{lb}^{(k)}\right) \le \mathbf{lb}^{(k+1)}$. Therefore, Lemma 15 assures $\mathcal{N}(\mathbf{lb}^{(k)}) \le \mathbf{lb}^{(k+1)}$. Hence we have

$$\begin{aligned}\boldsymbol{\nu}^{(k+1)} &= \mathcal{N}(\boldsymbol{\nu}^{(k)}) \\ &\le \mathcal{N}(\mathbf{lb}^{(k)}) \quad \text{(induction hypothesis, monotonicity of } \mathcal{N} \text{ (Esparza et al. [37]))} \\ &\le \mathbf{lb}^{(k+1)} \quad \text{(as argued above) .}\end{aligned}$$

So we have $\boldsymbol{\nu}^{(k)} \le \mathbf{lb}^{(k)}$ for all k. By the floating assignment in line 4, we have $\mathbf{lb}^{(k)} \prec f(\mathbf{lb}^{(k)})$, so $\mathbf{lb}^{(k)} \prec \mu_f$ by Theorem 16. As $(\boldsymbol{\nu}^{(k)})_{k\in\mathbb{N}}$ converges to μ_f, the sequence $(\mathbf{lb}^{(k)})_{k\in\mathbb{N}}$ converges to μ_f as well. In addition, it was shown in Esparza et al. [37], Kiefer et al. [64] that $(\boldsymbol{\nu}^{(k)})_{k\in\mathbb{N}}$ converges linearly to μ_f. As $\boldsymbol{\nu}^{(k)} \le \mathbf{lb}^{(k)}$, the same holds for $(\mathbf{lb}^{(k)})_{k\in\mathbb{N}}$.
Now we turn the upper bounds $\mathbf{ub}$. We prove the following invariants of the algorithm:

(a) $f(\mathbf{ub}) \le \mathbf{ub} \le \bar{1}$;

(b) for all components j with $\mathbf{ub}_i < 1$, we have $f_i(\mathbf{ub}) < \mathbf{ub}_i$.

Clearly, this holds at the beginning (when $\mathbf{ub} = \overline{1}$). The invariants are preserved by the assignment in line 8. Repeated application of Lemma 17 shows that the floating assignment in line 9 is valid and that the invariants are preserved. Lemma 16 implies that the floating assignment in line 13 are valid and preserve the invariants. Hence, the invariants hold.

Next we prove that for any component i with $(\mu_f)_i < 1$, we eventually have $\mathbf{ub}_i < 1$. Let us assume for the sake of a contradiction that there exists a component i with the property $P(i)$, where $P(i)$ means that $(\mu_f)_i < 1$ and $\mathbf{ub}_i = 1$ holds during the entire execution of the algorithm. Choose i "minimal" in the sense that for all variables X_j on which X_i depends we have that either X_i and X_j are in the same SCC, or $P(j)$ does not hold. Let S be the SCC of X_i and let X_j be any variable from $\mathrm{Var} \setminus S$ on which X_i depends. Since $P(i)$ holds, we must have $\mathbf{ub}_j = 1$ during the entire execution of the algorithm, because if $\mathbf{ub}_j < 1$ were true at some point, it would take at most n iterations before $\mathbf{ub}_i < 1$. As $P(j)$ cannot hold by the minimality of i, we have $(\mu_f)_j = 1$. Therefore, letting g denote the PSP obtained by restricting f to the S-components and replacing all variables from other SCCs by the constant 1, we have $\mu_g = (\mu_f)_S \prec \overline{1}$. Since $\mathbf{ub}_S = \overline{1}$ holds during the execution of the algorithm, we have $g(\overline{1}) = \overline{1}$, i.e., $\overline{1}$ is a fixed point of g. Therefore, by Lemma 9, g cannot be linear, as $\mu_g \prec \overline{1}$. Since f is perfectly superlinear, g must then be purely superlinear. Application of Lemma 18 (with $\mathbf{x} := \mu_g$ and $\mathbf{u} := \overline{1} - \mu_g$) yields

$$g'(\overline{1})(\overline{1} - \mu_g) \succ g(\overline{1}) - g(\mu_g) = \overline{1} - \mu_g \,.$$

Since the sequence of $\mathbf{lb}_S$ computed during the execution of the algorithm converges to μ_g, the continuity of $g'(\overline{1})$ implies that eventually $g'(\overline{1})(\overline{1} - \mathbf{lb}_S) \succ \overline{1} - \mathbf{lb}_S$ holds. But this means that the condition of line 12 is satisfied and, thus, the following assignment causes $\mathbf{ub}_S \prec \overline{1}$, contradicting our assumption that $P(i)$ holds. So we have shown that for any component i with $(\mu_f)_i < 1$, we eventually have $\mathbf{ub}_i < 1$.

Denote by $(\mathbf{ub}^{(k)})_{k \in \mathbb{N}}$ the sequence of upper bounds $\mathbf{ub}$ computed by the algorithm. It remains to show that this sequences converges linearly to μ_f. We have shown above that there exists k_0 such that for all $k \geq k_0$ we have that

$\mathbf{ub}_i^{(k+1)} \leq \mathbf{ub}_i^{(k)} < 1$ holds for all components i with $(\mu_f)_i < 1$. Choose a real number r with $0 < r < 1$ such that for the point $\mathbf{p} := \mu_f + r(\overline{1} - \mu_f)$ we have $\mathbf{ub}^{(k_0)} \leq \mathbf{p} \leq \overline{1}$ and the following is true for all components i: either $(\mathbf{ub}^{(k_0)})_i = 1$ or $\mathbf{ub}_i^{(k_0)} < \mathbf{p}_i < 1$. Define the sequence $(\mathbf{p}^{(k)})_{k \geq k_0}$ by setting $\mathbf{p}^{(k_0)} := \mathbf{p}$ and $\mathbf{p}^{(k+1)} := f(\mathbf{p}^{(k)})$ for all $k \geq k_0$. By Lemma 10 and Lemma 11, this sequence converges linearly to μ_f. To prove that the same holds for $(\mathbf{ub}^{(k)})_{k \in \mathbb{N}}$, it suffices to show that $\mathbf{ub}^{(k)} \leq \mathbf{p}^{(k)}$ holds for all $k \geq k_0$. We proceed by induction on k. The induction base ($k = k_0$) holds by definition of $\mathbf{p}^{(k_0)}$. Let $k \geq k_0$. Then we have:

$$\begin{aligned}
\mathbf{ub}^{(k+1)} &\leq f(\mathbf{ub}^{(k)}) && (\textbf{such that} \text{ clause of line 9})\\
&\leq f(\mathbf{p}^{(k)}) && (\text{induction hypothesis})\\
&= \mathbf{p}^{(k+1)} && (\text{definition of } \mathbf{p}^{(k+1)})
\end{aligned}$$

This completes the proof. $\square$

To summarize, Algorithm 3 computes provably and even verifiably correct lower and upper bounds, although exact computation is restricted to detecting numerical problems.

6.5 Case Study: A Neutron Branching Process

For testing the efficiency of our algorithms we consider a classical problem of nuclear physics: determining the critical mass or, equivalently, the critical radius of a perfect sphere of plutonium[1]. Roughly speaking, the critical radius is the smallest radius that will cause a nuclear explosion. More precisely, recall that the explosion is produced by a chain reaction: spontaneous fission of an atom liberates neutrons, whose collisions with other atoms induce further fissions etc. Following Harris [54], we model the ball by an MFBP describing the population of atoms fissioning at different distances from the ball's center. Initially there is one free neutron in the ball. A chain reaction occurs if its line of descendants does not go ultimately extinct (physically, this is identical to all atoms in the

[1]We assume room temperature, and so the density of plutonium is known.

ball fissioning in a very short time). Since the spontaneous fission rate is high (several hundred atoms per second per cm^3), even a small probability that one fission causes a chain reaction results in an explosion with large probability after a short time. So the critical radius is approximately given by the smallest radius such that $\psi < \overline{1}$.

Let us assume that the radius of the considered sphere is D, and that a neutron born at distance ξ from the center collides with an atom at distance η from the center with probability density $R(\xi, \eta)$. Let further p_k be the probability that a collision generates k neutrons ($k = 0$ means that no fission occurs). Harris uses the values $p_0 = 0.025$, $p_1 = 0.830$, $p_2 = 0.07$, $p_3 = 0.05$, $p_4 = 0.025$, $p_k = 0$ for $k > 4$, and also gives an expression for $R(\xi, \eta)$ (see Harris [54], p. 86).

The probability that a neutron starting at distance ξ collides with an atom at a distance in the interval $[a, b]$ (with $0 \leq a \leq b \leq D$) and generates k neutrons can be expressed as

$$\theta(\xi, a, b, k) := p_k \cdot \int_a^b R(\xi, \eta) \, d\eta.$$

By discretizing the interval $[0, D]$ into n segments we obtain an MFBP $\mathcal{B}(D, n) = (\mathrm{Pr}_1, \ldots, \mathrm{Pr}_n)$ with n types $1, \ldots, n$. An individual of type i represents a neutron whose distance from the center lies in between $(i-1)D/n$ and iD/n. The distributions $\mathrm{Pr}_i, 1 \leq i \leq n$ of the MFBP $\mathcal{B}(D, n)$ are given, for $\boldsymbol{c} \in \mathbb{N}^n$, by

$$\mathrm{Pr}_i(\boldsymbol{c}) = \begin{cases} \theta(\frac{(i-0.5)D}{n}, \frac{(j-1)D}{n}, \frac{jD}{n}, k) & \text{if } \boldsymbol{c}_j = k \geq 1 \text{ and } \boldsymbol{c}_\ell = 0 \text{ for } \ell \neq j \\ 1 - (1 - p_0) \cdot \int_0^D R(\frac{(i-0.5)D}{n}, \eta) \, d\eta & \text{if } \boldsymbol{c} = \overline{0} \\ 0 & \text{otherwise,} \end{cases}$$

and we use the methods we developed in the preceding sections to compute bounds for the extinction probability ψ_1 and to decide whether $(\psi_1, \ldots, \psi_n)^\top = \overline{1}$. Note that ψ_1 is the (approximative) probability that a neutron born in the centre does not cause an explosion.

Results. We used three different discretizations $n = 50, 100, 150$ for our experiments. We applied our consistency algorithm from Section 6.3 to check incon-

D	2			3			6			10		
n	50	100	150	50	100	150	50	100	150	50	100	150
inconsistent (yes/no)	n	n	n	y	y	y	y	y	y	y	y	y
Cons. check (Alg. Sec. 6.3)	< 1	2	6	< 1	1	3	< 1	1	3	< 1	1	3
Cons. check (Maple LP)	5	95	532	6	99	544	2	21	> 20min	< 1	7	> 20min
Approx. ψ_1 ($\epsilon = 10^{-3}$)	< 1	2	4	4	24	75	3	12	21	2	8	25
Approx. ψ_1 ($\epsilon = 10^{-4}$)	< 1	2	5	4	24	76	3	12	34	3	12	33
Cons. check (QSOpt_ex)	< 1	4	41	< 1	12	57	< 1	< 1	6	< 1	4	14

Table 6.2: Runtime in seconds of various algorithms on different values of D and n; we chose 20 minutes as a timeout.

sistency, i.e., to check whether an explosion occurs, and compare it to the LP approach using Maple's exact simplex package. The results are given in the first 3 rows of Table 6.2: Again our algorithm dominates the LP approach, although the polynomials are much denser than in the $h^{(n)}$-systems.

We additionally give running times of the LP approach if we replace Maple's simplex implementation by QSopt_ex (*QSOpt_ex solver* [90]), a standalone exact LP solver. It uses a mixture of floating point arithmetic and exact arithmetic (for checking the computed results) to speed up LP computations. Our consistency algorithm even outperforms this highly optimized LP solver. The speed differences are especially significant if D is close to the critical radius, i.e. if the systems are "almost" (in-)consistent.

We also implemented Algorithm 3 using Maple for computing lower and upper bounds on ψ_1 with two different values of the error bound ϵ. The runtime is given in the fourth and fifth rows. By setting the *Digits* variable in Maple we controlled the precision of Maple's software floating-point numbers for the floating assignments. In all cases starting with the standard value of 10, Algorithm 3 increased *Digits* at most twice by 5, resulting in a maximal *Digits* value of 20. We mention that Algorithm 3 computed an upper bound $\prec \overline{1}$, and thus proved inconsistency, after the first few iterations in all investigated cases, almost as fast as the consistency algorithm from Section 6.3. The numerical results, plotted in Figure 6.5, fit in well with the approximations given in Harris [54].

Computing approximations for the critical radius. From the data displayed in Figure 6.5 one can suspect that the critical radius (i.e., the smallest value of D for which the probability of a neutron in the centre causing an explo-

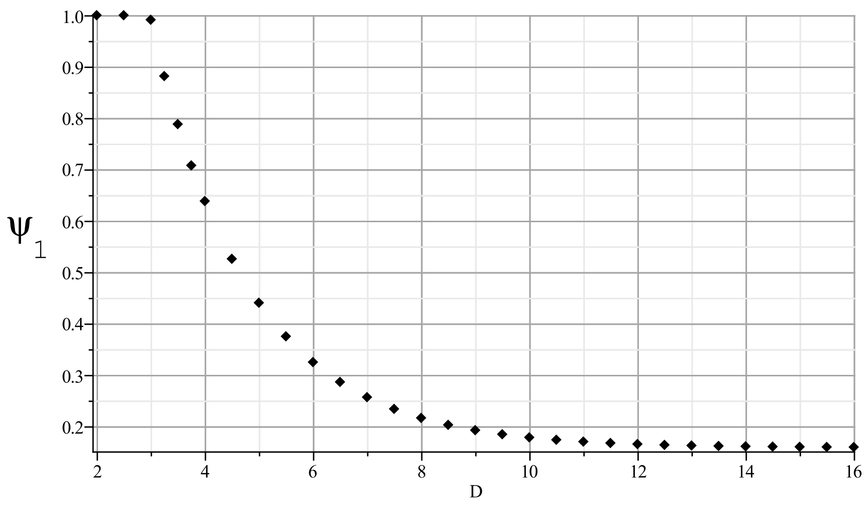


Figure 6.5: ψ_1 for different values of D, $n = 100$.

sion is still 1) lies somewhere between 2.7 and 3.

We take different discretizations $n = 25, 50, 75, 100, 150$ and combine our algorithm with binary search to determine the critical radius up to an error of 0.001, using our Maple implementation. During the search, the algorithm analyzes MBPs that get closer and closer to being critical. The running times of our algorithm for the last (and most expensive) binary search step that decreases the interval to 0.001 are given in Table 6.3. We found the critical radius to be in the interval $[2.981, 2.982]$ (using the finest discretization $n = 150$). Harris [54] estimates 2.9.

We also measured the time required for analyzing the MFBP in the last step of the binary search if we replace our algorithm by linear programming. Again we compared our algorithm to Maple's exact simplex package as well as the QSOpt_ex tool. Our approach outperforms both by at least an order of magnitude. This again indicates that the LP approach is less efficient for almost consistent PSPs.

n	25	50	75	100	150
Critical radius	2.9790	2.9809	2.9815	2.9815	2.9815
Precision	± 0.0005	± 0.0005	± 0.0005	± 0.0005	± 0.0005
Alg. Sec. 6.3	< 1	< 1	< 1	1	4
Exact LP (Maple Simplex)	< 1	6	32	108	588
Exact LP (QSOpt_ex solver)	< 1	< 1	4	14	72

Table 6.3: Runtimes in seconds for the last step of the binary search described in the text.

6.6 Conclusion

In this chapter we have studied extinction in multi-type finite branching processes. We discussed applications for different computer science formalisms like SCFGs and spPDAs. We have shown how to reduce the problem of computing extinction probabilities of MFBPs to computing fixed points of PSPs. We presented a new, simple, and efficient algorithm for checking the consistency of PSPs, which can be used for deciding whether the extinction probabilities of an MFBP are all equal to one. Our algorithm outperforms the previously existing LP-based method. In particular, it has strongly-polynomial complexity. We have also described the first algorithm that computes reliable lower and upper bounds for the vector of extinction probabilities of an MFBP, and showed that the sequence of bounds converges linearly. To achieve these properties without sacrificing efficiency, we used a novel combination of exact and inexact (floating-point) arithmetic. Experiments originating from concrete branching processes confirm the practicality of our approach.

Chapter 7

Summary and Outlook

In this work we have presented techniques for checking reachability properties of probabilistic programs and multi-type finite branching processes. Our main result can be summarized as follows: Reachability properties of probabilistic programs can be efficiently checked for important program classes, by using abstraction, reduction to nonprobabilistic problems, and numerical techniques, like combining inexact and exact arithmetic. We shortly provide the three key contributions of the thesis:

1. We have developed an extension of existing abstraction approaches for probabilistic programs. In particular, we have presented an algorithm for constructing abstract game arenas from probabilistic programs, which uses *arbitrary abstract domains* and *widenings* from the abstract interpretation framework. In contrast to existing approaches, abstract states do not have to form a partition of the state space, and domains with infinitely many elements and fast abstract operations can be used. We have shown advantages of this extension with the help of case studies.

2. We have presented a novel method for proving almost-sure termination of probabilistic programs using *patterns*. Our approach forms a complete proof method for finite and weakly finite programs, an important class of programs with possibly infinite state space. We have integrated state-of-the-art termination provers and LTL model checkers in our tool chain to generate suitable patterns automatically respectively semi-automatically,

and have shown how to use the found patterns to prove almost-sure termination by checking an instrumented program for sure termination.

3. Finally we have developed numerical methods for checking extinction probabilities of multi-type finite branching processes. We provided a strongly-polynomial algorithm for the problem of deciding whether all extinction probabilities of an MFBP are one. We also presented an algorithm for computing reliable upper and lower bounds for extinction probabilities that uses a mixture of exact and inexact arithmetic for efficiency.

However, much remains to do. We point out several directions for future work.

Concurrent programs with fairness constraints. In Chapter 4 we investigated probabilistic programs P with nondeterministic behaviour. We computed bounds for the *minimum* and *maximum* of reachability probabilities, taken over *all* possible strategies for $\mathcal{M}_P$, the MDP asociated with P. However, these values can be unrealistic in practice. Consider for example concurrent programs where the nondeterminism is caused by different decisions of a scheduler. Often one assumes *fair schedulers*, which e.g. guarantee that every task is scheduled infinitely often during a program run. This implies that several behaviours (i.e. strategies for $\mathcal{M}_P$) are not possible and should not be considered for computing *extremal* reachability values. This can be expressed by *fairness constraints*, for which many different variants exist (see e.g. Baier and Kwiatkowska [8], Francez [45]). A simple fairness constraint can be represented by a set G of guarded commands of a PGP P; a program run r then is fair with respect to G if the following holds for every guarded command $a \in G$: if r visits states infinitely often where g_a is enabled, then infinitely often a transition labeled by a is taken (this corresponds to so-called *strong action fairness*). In Baier et al. [11] a method is given for incorporating fairness constraints efficiently into quantitative model checking of MDPs. It seems promising to integrate fairness constraints into the game-based abstraction scenario. For this one might have to define a notion of fairness for stochastic 2-player game arenas (hereby taking the roles of the two players into account) and relate it to fairness constraints given for the probabilistic program P respectively its concrete MDP $\mathcal{M}_P$. This would extend the application area

and practicability of our programs considerably.
A similar extension seems possible for the pattern approach given in Chapter 4: we gave a possible class of patterns for arbitrary *nondeterministic* weakly finite programs in Section 5.5. However, other pattern classes might be more efficient for checking e.g. concurrent programs where nondeterministic behaviour is solely caused by choices of the process scheduler. In particular patterns for showing a.s.-termination with respect to fairness constraints appear to be worthwhile to explore in the future.

Precise results from value iteration. In our experiments from Chapter 4 we used value iteration relying on Java's floating point arithmetic capabilities for computing the extremal game values of abstract game arenas. To exclude potential rounding errors it seems wise to adapt our method of floating assignments from Chapter 6 to value iteration and to investigate what other numerical techniques could profit from our technique.

Combining and extending the techniques. We already sketched in Section 5.7 how termination information might help to make game-based abstraction more efficient. A way of automating the information transfer between the two approaches might be offered by fairness constraints. Let us recall a variant of the example program from Section 5.7:

```
int x = N; int y = 0;

a: (x > 0)  -> 0.5: x' = x-1 + 0.5: x' = x

b: (x <= 0) -> 0.5: y' = 0   + 0.5: y' = 1

reach: (y = 1)
```

The pattern approach might be used to prove that eventually the guarded command `b` is chosen. This information can be expressed as the fairness constraint $\{b\}$, and we can exclude all strategies that do not eventually choose the guarded command b with probability 1.
Another interesting direction for further research lies in the development of abstract domains that are particularly suited for building abstract game arenas,

taking products of predicate domains and numerical domains like intervals or octagons as a starting point. We saw in Section 4.8 an example for CEGAR-based predicate-abstraction beating e.g. the sole use of the interval domain, and also examples for the other way around. One can combine the sophisticated refinement methods offered by CEGAR with the flexibility of infinite domains and widenings; e.g. invariants computed by widenings might be added as predicates to the predicate domain. Finally, Algorithm 1 could be made more flexible by *merging* newly constructed states into existing ones. For this one could define an equivalence relation over abstract states, and merge equivalent states, similarly to the DAG-based approach in Gulavani et al. [50]. Also other structures of game arenas are possible (e.g. more similar to the structure of game arenas in Kattenbelt et al. [62]).

In our research we used different techniques for several problem variants of reachability problems for probabilistic systems. It became apparent that the developed approaches offer possibilities for extensions and cross-fertilization. Currently we are working on a tool that integrates our prototype implementations from Chapter 4 and Chapter 5 into a common framework, which supports a wide range of probabilistic programs and analysis options. Our hope for the future is to check properties of important program classes that are still out of reach for our methods today.

Appendix A

Missing Proofs of Chapter 4

We abbreviate the term “Induction hypothesis” by IH in the following.

A.1 Proof of Lemma 3, Part (1)

We complete the proof of several propositions used in the context of the proof of Lemma 3, part (1).

Lemma 19. *For all* $\Pi \in \mathit{Paths}^{\mathcal{G},(S_1,S_2)} \cap (QLab)^*(Q_1 \cup Q_2)$ *with* $\pi = T(\Pi)$ *it holds that*

$$\mathit{Pr}^{\mathcal{G},(S_1,S_2)}[\mathit{Cyl}(\Pi,\mathcal{G})] = \mathit{Pr}^{\mathcal{M},S}[\mathit{Cyl}(\pi,\mathcal{M})].$$

□

Proof.

We prove the claim by induction over the structure of Π.

In the case of $\Pi = s_0$,

$$\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(s_0,\mathcal{G})] = 1 = \Pr^{\mathcal{M},S}[\mathrm{Cyl}(\sigma_0,\mathcal{M})] = \Pr^{\mathcal{M},S}[\mathrm{Cyl}(T(s_0),\mathcal{M})]$$

holds. We consider the other possible forms of Π:

$\underline{\Pi = \hat{\Pi} \xrightarrow{d} \langle s, a, d\rangle \xrightarrow{i} d(i), a \in \mathcal{C}, i \in \mathbb{N}}$:

$\hat{\Pi}$ ends in the state $\langle s, a\rangle$. Using property (*) we get that $T(\hat{\Pi})$ ends in a state $\langle \sigma, a\rangle$, $\sigma \in \Sigma_{\mathcal{V}}$.

$$\begin{aligned}
&\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi} \xrightarrow{d} \langle s, a, d\rangle \xrightarrow{i} d(i), \mathcal{G})] \\
&= \Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi}, \mathcal{G})] \cdot S_2(\hat{\Pi})(d) \cdot p_a(i) && \text{(structure of } \mathcal{G}\text{)} \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(T(\hat{\Pi}), \mathcal{M})] \cdot S_2(\hat{\Pi})(d) \cdot p_a(i) && \text{(IH)} \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(T(\hat{\Pi}), \mathcal{M})] \cdot p_a(i) && \text{(Def. of } S_2\text{)} \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(T(\hat{\Pi}) \overset{i}{\Rightarrow} [\![c_a(i)]\!](\sigma), \mathcal{M})] && \text{(note above)} \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(T(\hat{\Pi} \xrightarrow{d} \langle s, a, d\rangle \xrightarrow{i} d(i)), \mathcal{M})] && \text{(Def. of } T\text{, case 2).}
\end{aligned}$$

$\underline{\Pi = \hat{\Pi} \xrightarrow{a} \langle s, a\rangle, a \in \mathcal{C}}$:

$$\begin{aligned}
&\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi} \xrightarrow{a} \langle s, a\rangle, \mathcal{G})] \\
&= \Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi}, \mathcal{G})] \cdot S_1(\hat{\Pi})(a) \\
&= \Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi}, \mathcal{G})] \cdot S(T(\hat{\Pi}))(a) && \text{(Def. of } S_1\text{, case 1)} \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(T(\hat{\Pi}), \mathcal{M})] \cdot S(T(\hat{\Pi}))(a) && \text{(IH)} \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(T(\hat{\Pi}) \overset{a}{\Rightarrow} \langle s, a\rangle, \mathcal{M})] && \text{(structure of } \mathcal{M}\text{)} \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(T(\hat{\Pi} \xrightarrow{a} \langle s, a\rangle), \mathcal{M})] && \text{(Def. of } T\text{).}
\end{aligned}$$

$\underline{\Pi = \hat{\Pi} \xrightarrow{\circ} q, \text{ with } q = \langle s, \circledcirc\rangle \text{ respectively } q = \circledcirc}$:

Then $T(\hat{\Pi})$ ends in $q' = \langle \sigma_f, \circledcirc\rangle$ (and then the second last state in $T(\hat{\Pi})$ must be in F) respectively $q' = \circledcirc$, due to property (*). We get

$$\begin{aligned}
&\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi} \overset{\circ}{\rightarrow} q, \mathcal{G})] \\
&= \Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi}, \mathcal{G})] && \text{(Def. of } S_1 \text{ resp. } S_2 \text{ in cases 2 and 3)} \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(T(\hat{\Pi}), \mathcal{M})] && \text{(IH)} \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(T(\hat{\Pi}) \overset{\circ}{\Rightarrow} q', \mathcal{M})] && \text{(structure of } \mathcal{M}\text{, note above)} \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(T(\Pi), \mathcal{M})].
\end{aligned}$$

□

Lemma 20. *T is bijective.* □

Proof. We first prove that T is injective: Let Π be a path in the domain of T, and let π be a prefix of $T(\Pi)$. Let Π' be a path such that Π is a prefix of Π (i.e. Π' is an extension of Π). Then π is a prefix of $T(\Pi')$, i.e. T *preserves* prefixes. Let now Π_1, Π_2 be two possible paths in the domain of T with $\Pi_1 \neq \Pi_2$. They have a maximal common prefix Π (at least s_0), so we can write them as:

$$\Pi_1 = \Pi \xrightarrow{\ell_1} q_1 \rightarrow \ldots \text{ and } \Pi_2 = \Pi \xrightarrow{\ell_2} q_2 \rightarrow \ldots.$$

Since we assume that $\Pi_1 \neq \Pi_2$, either $\ell_1 \neq \ell_2$ or $q_1 \neq q_2$ has to hold. However, if we inspect the definition of T, we see that the newly defined values of T differ among each other in all cases: in case 1 by the action labels of the transitions, in case 2 by the labels for probabilistic transitions; in case 3, T is defined for only one possible extension. The definition of $T(\Pi_1)$ and $T(\Pi_2)$ both rely on the definition of $T(\Pi)$ as an intermediate step, but as we can conclude from above, there exists prefixes Π' of Π_1 respectively Π'' of Π_2 with the same length and $T(\Pi') \neq T(\Pi'')$. With the fact that T preserves prefixes we get $T(\Pi_1) \neq T(\Pi_2)$, which proves the claim.

We now prove that T is also surjective: Let $\pi \in \mathrm{Paths}^{\mathcal{M},S}$. We show that there exists $\Pi \in \mathrm{Paths}^{\mathcal{G},(S_1,S_2)}$ such that $T(\Pi) = \pi$ via structural induction over π:

- $\pi = \sigma_0$: Choose $\Pi = s_0$.

- $\pi = \hat{\pi} \overset{a}{\Rightarrow} \langle \sigma, a \rangle$ for $a \in \mathcal{C}$: $\hat{\pi}$ has σ as last state. There exists $\hat{\Pi}$ with $T(\hat{\Pi}) = \hat{\pi}$ by induction hypothesis. $\hat{\Pi}$ has a $s \in Q_1 \setminus \{\circledcirc, \otimes\}$ as last state due to property (*), and $\sigma \in \gamma(s)$, hence $s \xrightarrow{a} \langle s, a \rangle$ holds in $\mathcal{G}$. By construction of S_1, $S_1(\hat{\Pi})(a) = S(T(\hat{\Pi}))(a) = S(\hat{\pi})(a) > 0$. By definition of T we get

$$T(\hat{\Pi} \xrightarrow{a} \langle s, a \rangle) = T(\hat{\Pi}) \overset{a}{\Rightarrow} \langle \sigma, a \rangle = \hat{\pi} \overset{a}{\Rightarrow} \langle \sigma, a \rangle = \pi.$$

  So we can choose $\Pi = \hat{\Pi} \xrightarrow{a} \langle s, a \rangle$.

- $\pi = \hat{\pi} \overset{a}{\Rightarrow} \langle \sigma, a \rangle \overset{i}{\Rightarrow} [\![c_a(i)]\!](\sigma)$ for $1 \leq i \leq |up_a|$: Let $\hat{\Pi}$ be with $T(\hat{\Pi}) = \hat{\pi} \overset{a}{\Rightarrow} \langle \sigma, a \rangle$ via induction hypothesis. $\hat{\Pi}$'s last state has the form $\langle s, a \rangle$ with $a \in \mathcal{C}$ due to property (*), and $\sigma \in \gamma(s)$ (by considering the second last state s in $\hat{\Pi}$). Let then $d = \beta(s, a, \sigma)$. By construction of S_1, $S_2(\hat{\Pi})(d) = 1$. We get:

$$\begin{aligned} T(\hat{\Pi} \xrightarrow{d} \langle s, a, d \rangle \xrightarrow{i} d(i)) &= T(\hat{\Pi}) \overset{i}{\Rightarrow} [\![c_a(i)]\!](\sigma) \\ &= \hat{\pi} \overset{a}{\Rightarrow} \langle \sigma, a \rangle \overset{i}{\Rightarrow} [\![c_a(i)]\!](\sigma) = \pi. \end{aligned}$$

  We choose $\Pi = \hat{\Pi} \xrightarrow{d} \langle s, a, d \rangle \xrightarrow{i} d(i)$.

- $\pi = \hat{\pi} \overset{i}{\Rightarrow} \sigma \overset{\circ}{\Rightarrow} \langle \sigma_f, \circ \rangle, i \in \mathbb{N}$:

  Then $\hat{\Pi}$ with $T(\hat{\Pi}) = \hat{\pi} \overset{i}{\rightarrow} \sigma$ exists by induction hypothesis. Furthermore the last state s of $\hat{\Pi}$ is contained in $Q_1 \setminus \{\circledcirc, \otimes\}$ and $\sigma \in \gamma(s)$, due to property (*). Note that $\sigma \in F$, and so $s \overset{\circ}{\rightarrow} \langle s, \circledcirc \rangle$ and $S_1(\hat{\Pi})(\circ) = 1$. By definition of T we get

  $$T(\hat{\Pi} \overset{\circ}{\rightarrow} \langle s, \circledcirc \rangle) = T(\hat{\Pi}) \overset{\circ}{\Rightarrow} \langle \sigma_f, \circledcirc \rangle = \hat{\pi} \overset{\circ}{\Rightarrow} \langle \sigma_f, \circledcirc \rangle = \pi.$$

  We choose $\Pi = \hat{\Pi} \overset{\circ}{\rightarrow} \langle s, \circledcirc \rangle$.

- $\pi = \hat{\pi} \overset{\circ}{\Rightarrow} \circledcirc$ is similar to the previous case.

$\square$

A.2 Proof of Lemma 3, Part (2)

We complete the proof of several propositions used in the context of the proof of Lemma 3, part (2).

Lemma 21.

S is well-defined. $\square$

Proof.

Let us assume that there exists a path π of minimal length such that $S(\pi)$ is defined differently at two different steps in the construction. Then $\pi \neq \sigma_0$ due to property (*) and the definitions in case 1 of the construction. π also does not end in $\langle \sigma_f, \circledcirc \rangle$ or σ_f, since in these cases there is only one choice for $S(\pi)$.

It therefore remains to show well-definedness for $S(\pi)$ with π having the shape

$$\pi = \hat{\pi} \overset{a}{\Rightarrow} \langle \hat{\sigma}, a \rangle \overset{\ell}{\Rightarrow} \sigma$$

with $\sigma \neq \sigma_f, \ell \in \mathbb{N}$. Both definitions of $S(\pi)$ have to occur in case 2, for two different (S_1, S_2)-possible paths Π_i, $i \in \{1, 2\}$, with

$$\Pi_i = \Pi'_i \overset{a_i}{\Rightarrow} \langle s'_i, a_i \rangle \xrightarrow{d_i} \langle s'_i, a_i, d_i \rangle \overset{\ell_i}{\Rightarrow} s_i$$

and $T(\Pi_i, \pi) > 0$. Following the construction of (both) Π_i backwards we get:

- $T(\Pi_1, \pi)$ and $T(\Pi_2, \pi)$ have both been defined in case 2; it follows $\ell_1 = \ell_2 = \ell$ and for $i \in \{1, 2\}$ that $T(\Pi'_i \xrightarrow{a_i} \langle s'_i, a_i \rangle, \hat{\pi} \overset{a_i}{\Rightarrow} \langle \hat{\sigma}, a \rangle) > 0$.
- Both values $T(\Pi'_i \xrightarrow{a_i} \langle s'_i, a_i \rangle, \hat{\pi} \overset{a_i}{\Rightarrow} \langle \hat{\sigma}, a \rangle)$ were defined in case 1; it follows $T(\Pi'_i, \hat{\pi}) > 0$, and $a_1 = a = a_2$.
- From $T(\Pi'_i, \hat{\pi}) > 0$ for both $i \in \{1, 2\}$ it follows that $S(\hat{\pi})$ is defined in case 1 for the cases (Π'_i). Since π is chosen minimal with multiple definitions, $\Pi'_1 = \Pi'_2$. This implies $d_1 = d_2$ and so $s_1 = s_2$, and so $\Pi_1 = \Pi_2$, which contradicts our assumption.

□

Lemma 22. *For every $\Pi \in Paths^{\mathcal{G},(S_1,S_2)}$ it holds that:*

$$Pr^{\mathcal{G},(S_1,S_2)}[Cyl(\Pi, \mathcal{G})] = \sum_{\pi \in Paths^{\mathcal{M},S}} T(\Pi, \pi).$$

□

Proof.

Note that, for a fixed path Π, all values $T(\Pi, \pi) > 0$ are defined at *exactly one* position during the construction; this allows us to give a proof via induction over the structure of Π. If $\Pi = s_0$, $\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(s_0, \mathcal{G})] = 1 = T(s_0, \sigma_0)$ holds. Now we consider the other possible forms of Π:

<u>$\Pi = \hat{\Pi} \xrightarrow{d} \langle s, a, d\rangle \xrightarrow{i} d(i), a \in \mathcal{C}, i \in \mathbb{N}$:</u>

$$\begin{aligned}
&\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi} \xrightarrow{d} \langle s, a, d\rangle \xrightarrow{i} d(i), \mathcal{G})] \\
&= \Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi}, \mathcal{G})] \cdot S_2(\hat{\Pi})(d) \cdot p_a(i) && \text{(structure of } \mathcal{G}) \\
&= \left(\sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi}, \pi)\right) \cdot S_2(\hat{\Pi})(d) \cdot p_a(i) && \text{(IH)} \\
&= \left(\sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi}, \pi)\right) \cdot \frac{\sum_{\pi' \in M_{a,d}} T(\hat{\Pi}, \pi')}{\sum_{\pi'' \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi}, \pi'')} \cdot p_a(i) && \text{(Def. of } S_2) \\
&= \sum_{\pi' \in M_{a,d}} T(\hat{\Pi}, \pi') \cdot p_a(i) \\
&= \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\Pi, \pi) && \text{(Def. of } T\text{, case 2).}
\end{aligned}$$

<u>$\Pi = \hat{\Pi} \xrightarrow{a} \langle s, a\rangle, a \in \mathcal{C}$:</u>

$$\begin{aligned}
&\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi} \xrightarrow{a} \langle s, a\rangle, \mathcal{G})] \\
&= \Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi}, \mathcal{G})] \cdot S_1(\hat{\Pi})(a) && \text{(structure of } \mathcal{G}) \\
&= \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi}, \pi) \cdot S_1(\hat{\Pi})(a) && \text{(IH)} \\
&= \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi} \xrightarrow{a} \langle s, a\rangle, \pi) && \text{(Def. of } T\text{, case 1).}
\end{aligned}$$

<u>$\Pi = \hat{\Pi} \xrightarrow{\circ} \langle s, \circledcirc\rangle$:</u>

Then $\mathrm{last}(\hat{\Pi}) \in Q_1 \setminus \{\circledcirc, \otimes\}$ holds; we get

$$\begin{aligned}
&\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi} \xrightarrow{\circ} q, \mathcal{G})] \\
&= \Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi}, \mathcal{G})] \cdot S_1(\hat{\Pi})(\circ) \\
&= \sum_{\pi' \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi}, \pi') \cdot S_1(\hat{\Pi})(\circ) && \text{(IH)} \\
&= \sum_{\pi' \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi} \xrightarrow{\circ} \langle s, \circledcirc\rangle, \pi' \overset{\circ}{\Rightarrow} \langle \sigma_f, \circledcirc\rangle) && \text{(Def. of } T\text{, case 1)} \\
&= \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi} \xrightarrow{\circ} \langle s, \circledcirc\rangle, \pi) && \text{(property (*)).}
\end{aligned}$$

The last step holds due to property (*): if $T(\hat{\Pi} \xrightarrow{\circ} \langle s, \circledcirc\rangle, \pi) > 0$ for a path π, then π ends in $\langle \sigma_f, \circledcirc\rangle$.

$\Pi = \hat{\Pi} \xrightarrow{\times} \otimes$ respectively $\Pi = \hat{\Pi} \xrightarrow{\circ} \circledcirc$:

The proofs of both cases are very similar, hence we only prove the case $\Pi = \hat{\Pi} \xrightarrow{\times} \otimes$. Then there exists $a \in \mathcal{C}$ such that $\hat{\Pi}$ ends in a state $\langle s, a\rangle$.

$$\begin{aligned}
&\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi} \xrightarrow{\times} \otimes, \mathcal{G})] \\
&= \Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi}, \mathcal{G})] \cdot S_2(\hat{\Pi})(\times) \\
&= \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi}, \pi) \cdot S_2(\hat{\Pi})(\times) && \text{(IH)} \\
&= \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi}, \pi) \cdot \left(1 - \frac{1}{\sum_{\pi' \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi}, \pi')} \cdot \sum_{\pi' \in M_a} T(\hat{\Pi}, \pi')\right) && \text{(Def. of } S_2\text{)} \\
&= \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi}, \pi) - \sum_{\pi' \in M_a} T(\hat{\Pi}, \pi').
\end{aligned}$$

The last term is equal to $\sum_{\pi \notin M_a} T(\hat{\Pi}, \pi')$, and we get with the definition of T in case 2 that $\sum_{\pi \notin M_a} T(\hat{\Pi}, \pi') = \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi} \xrightarrow{\times} \otimes, \pi)$. $\square$

Lemma 23. *For* every *S-possible path π in $\mathcal{M}$ it holds that*

$$Pr^{\mathcal{M},S}[Cyl(\pi, \mathcal{M})] = \sum_{\Pi \in Paths^{\mathcal{G},(S_1,S_2)}} T(\Pi, \pi).$$

$\square$

Proof. We prove the claim by induction over the structure of π.

$\underline{\pi = \sigma_0 :}$

We get $\sum_{\Pi \in \mathrm{Paths}^{\mathcal{G},(S_1,S_2)}} T(\Pi, \pi) = T(s_0, \sigma_0) = 1 = \mathrm{Pr}^{\mathcal{M},S}[\mathrm{Cyl}(\sigma_0, \mathcal{M})]$.

$\underline{\pi = \hat{\pi} \Rightarrow \hat{\sigma} \overset{a}{\Rightarrow} \langle \hat{\sigma}, a \rangle \overset{i}{\Rightarrow} \sigma,\ a \in \mathcal{C} :}$

Let $up_a = \langle \langle p_1, c_1 \rangle, \ldots, \langle p_k, c_k \rangle \rangle$ and $1 \leq i \leq k$. Let Π be such that $T(\Pi, \pi) > 0$. By using property (*) we can deduce that Π has the shape

$$\Pi = \hat{\Pi} \xrightarrow{a} \langle s', a \rangle \xrightarrow{d} \langle s', a, d \rangle \xrightarrow{i} s,$$

and so $T(\Pi, \pi)$ has been defined in case 2 of the construction. Due to the definition of T, $T(\Pi, \pi) = T(\hat{\Pi}, \hat{\pi}) \cdot S_1(\hat{\Pi})(a) \cdot p_i = T(\hat{\Pi}, \hat{\pi}) \cdot S(\hat{\pi})(a) \cdot p_i$. Assume on the other hand that $T(\tilde{\Pi}, \hat{\pi}) > 0$ for a path $\tilde{\Pi}$. a is enabled in the last state of $\tilde{\Pi}$ due to property (*). $\tilde{\Pi}$ has to satisfy the form corresponding to case 1. Then it follows that $S_1(\tilde{\Pi})(a) = S(\hat{\pi})(a)$, and so there exists a Π' such that $T(\Pi', \pi) = T(\tilde{\Pi}, \hat{\pi}) \cdot S(\hat{\pi})(a) \cdot p_i$. We get

$$\sum_{\Pi \in \mathrm{Paths}^{\mathcal{G},(S_1,S_2)}} T(\Pi, \pi) = \sum_{\Pi \in \mathrm{Paths}^{\mathcal{G},(S_1,S_2)}} T(\Pi, \hat{\pi}) \cdot S(\hat{\pi})(a) \cdot p_i, \text{ and conclude:}$$

$$
\begin{aligned}
&\Pr^{\mathcal{M},S}[\mathrm{Cyl}(\pi,\mathcal{M})] \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(\hat{\pi},\mathcal{M})] \cdot S(\hat{\pi})(a) \cdot p_i \\
&= \left(\sum_{\hat{\Pi} \in \mathrm{Paths}^{\mathcal{G},(S_1,S_2)}} T(\hat{\Pi},\hat{\pi}) \right) \cdot S(\hat{\pi})(a) \cdot p_i && \text{(IH)} \\
&= \sum_{\Pi \in \mathrm{Paths}^{\mathcal{G},(S_1,S_2)}} T(\Pi,\pi).
\end{aligned}
$$

$\underline{\pi = \hat{\pi} \overset{a}{\Rightarrow} \langle \sigma, a \rangle, a \in \mathcal{C} :}$

$$
\begin{aligned}
&\Pr^{\mathcal{M},S}[\mathrm{Cyl}(\pi,\mathcal{M})] \\
&= \Pr^{\mathcal{M},S}[\mathrm{Cyl}(\hat{\pi} \overset{a}{\Rightarrow} \langle \sigma, a \rangle,\mathcal{M})] \cdot S(\hat{\pi})(a) && \text{(structure of } \mathcal{M}) \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi},\hat{\pi}) \cdot S(\hat{\pi})(a) && \text{(IH)} \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi},\hat{\pi}) \cdot S_1(\hat{\Pi})(a) && (S_1(\hat{\Pi})(a) = S(\hat{\pi})(a)) \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi} \xrightarrow{a} \langle \mathrm{last}(\hat{\Pi}), a \rangle, \pi) && \text{(case dist.)} \\
&= \sum_{\Pi \in \mathrm{Paths}^{\mathcal{G},(S_1,S_2)}} T(\Pi,\pi) && \text{(property (*)).}
\end{aligned}
$$

$\underline{\pi = \hat{\pi} \Rightarrow \sigma \overset{\circ}{\Rightarrow} \langle \sigma_f, \circledcirc \rangle :}$

$$
\begin{aligned}
&\Pr^S[\mathrm{Cyl}(\pi, \mathcal{M})] \\
&= \Pr^S[\mathrm{Cyl}(\hat{\pi} \Rightarrow \sigma, \mathcal{M})] \cdot S(\pi)(\circ) && \text{(structure of } \mathcal{M}) \\
&= \left(\sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi}, \hat{\pi} \Rightarrow \sigma) \right) \cdot S(\pi)(\circ) && \text{(IH)} \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} \Big(T(\hat{\Pi}, \hat{\pi} \Rightarrow \sigma) \cdot S(\pi)(\circ) \Big) \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} \left(T(\hat{\Pi}, \hat{\pi} \Rightarrow \sigma) \cdot \left(S_1(\hat{\Pi})(\circ) + \sum_{a \in \mathcal{C} : \sigma \not\models a} S_1(\hat{\Pi})(a) \right) \right) && \text{(Def. of } T).
\end{aligned}
$$

We now have to distinguish the different cases for definitions of T: S_1 can choose an action a not enabled by σ or it can choose $\circ$. We can therefore write the last term from above as

$$
\sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} \left(T(\hat{\Pi} \xrightarrow{\circ} \langle \mathrm{last}(\hat{\Pi}), \circledcirc \rangle, \pi) + \sum_{a \in \mathcal{C} : \sigma \not\models a} T(\hat{\Pi} \xrightarrow{a} \langle \mathrm{last}(\hat{\Pi}), a \rangle, \pi) \right),
$$

which is equal to

$$
\sum_{\Pi \in \mathrm{Paths}^{(S_1,S_2)}} T(\Pi, \pi),
$$

again by the definition of T. $\square$

A.3 Proof of Lemma 3, Part (3)

In this section we comment on the completion of the proof of the third part of Lemma 3: given a strategy $S_1 \in \mathcal{S}_1(\mathcal{G})$ there exists a strategy $S \in \mathcal{S}(\mathcal{M}_P)$ such that

$$\mathrm{Pr}^{S}\mathrm{Reach}(\mathcal{M}_P, F) \geq \inf_{T \in S_2(\mathcal{G})} \mathrm{Pr}^{(S_1,T)}[\mathrm{Reach}(\mathcal{G}, \{\circledcirc\})].$$

We have already described the structure and function of $\mathcal{M}$ in the main text; again we use a function

$$T : \mathrm{Paths}^{\mathcal{G},(S_1,S_2)} \cap (Q\mathrm{Lab})^*(Q_1 \cup Q_2) \times \mathrm{Paths}^{\mathcal{M},S} \to [0,1],$$

serving the same purpose as in part (2) of the proof. It remains to describe the construction of S and S_2.

Definition of S and S_2.
Again we set $T(s_0, \sigma_0) := 1$ and use the following property that can easily be verified during the construction:

Property (*) for an (S_1, S_2)-possible path Π:
For all paths Π, π with $T(\Pi, \pi) > 0$: π is a (S)-possible path. Further the last states states of Π and π are related as follows:

1. Π ends in a state $s \in Q_1 \setminus \{\circledcirc, \otimes\}$ iff π ends in an action state $\sigma \neq \sigma_f$. Then $\sigma \in \gamma(s)$ holds.

2. Π ends in a state $\langle s, a\rangle$ with $a \in \mathcal{C}$ iff either

   - π ends in a state $\langle \sigma_r, \otimes\rangle$, and then also $\langle s, a\rangle \xrightarrow{\times} \otimes$ holds, or
   - π ends in a state $\langle \sigma, a\rangle$, and then $\sigma \in \gamma(s)$ holds.

3. If Π ends in $\langle s, \circledcirc\rangle \xrightarrow{\times} \langle s, \circledcirc\rangle$, then π ends in $\langle \sigma_r, \otimes\rangle$. If Π ends in $s \xrightarrow{\circ} \langle s, \circledcirc\rangle$, then π ends in $\langle \sigma_f, \circledcirc\rangle$ or in $\langle \sigma_r, \otimes\rangle$; in the later case $\Pi \xrightarrow{\times} \langle s, \circledcirc\rangle$ holds.

4. If Π ends in $\circledcirc$ then π ends in σ_f. If Π ends in $\otimes$ then π ends in σ_r.

5. The number of occurrences of Player 1 states in Π and the number of occurences of action nodes in π are equal. The number of occurrences of $\otimes$ in Π is equal to the number of occurrences of σ_r in π. The number of occurences of states $\langle s, \circledcirc\rangle$ in Π is equal to the number of occurrences of $\sigma_f, \sigma_r, \langle \sigma_f, \circledcirc\rangle$ and $\langle \sigma_r, \otimes\rangle$ in π.

□

For the construction we define several abbreviations: for $\Pi \in \mathrm{Paths}^{\mathcal{G},(S_1,S_2)}$, we set $t(\Pi) := \sum_{\pi:\pi\in\mathrm{Paths}\mathcal{M}_P,S} T(\Pi,\pi)$ and

$$t_f(\Pi) := \sum_{\substack{\pi\in\mathrm{Paths}^{\mathcal{M}_P,S}:\\ \mathrm{last}(\pi)=\langle\sigma_f,\circledcirc\rangle}} T(\Pi,\pi) \text{ and}$$
$$t_r(\Pi) := \sum_{\substack{\pi\in\mathrm{Paths}^{\mathcal{M}_P,S}:\\ \mathrm{last}(\pi)=\langle\sigma_r,\otimes\rangle}} T(\Pi,\pi).$$

Case 1: $\Pi = s_0$ respectively $\Pi = \hat{\Pi} \to s, s \in Q_1 \setminus \{\circledcirc, \otimes\}$.
Let $L = \{a \in \mathcal{C} \mid S_1(\Pi)(a) > 0\}$.
We give the following definitions for every $\pi \in \mathrm{Paths}^{\mathcal{G}}$ with $T(\Pi,\pi) > 0$: let σ be the last state of π with $\sigma \in \gamma(s)$ (by induction hypothesis, property (*)).
Let $G_\sigma = \{a \in \mathcal{C} \mid \sigma \models a\}$. For every $a \in \mathcal{C}$ holds: $\sigma \models a$ implies $s \xrightarrow{a} \langle s,a\rangle$. We use a case distinction:

- $\sigma \notin F$:

  (S) : For each $a \in G_\sigma$: Set $S(\pi)(a) := S_1(\Pi)(a)$.
  Set $S(\pi)(\times) := 1 - \sum_{a\in G_\sigma} S_1(\Pi)(a)$.

  (T) : For $a \in G_\sigma$: set $T(\Pi \xrightarrow{a} \langle s,a\rangle, \pi \overset{a}{\Rightarrow} \langle\sigma,a\rangle) := T(\Pi,\pi)\cdot S_1(\Pi)(a)$.
  For $a \in \mathcal{C}\setminus G_\sigma$: set $T(\Pi \xrightarrow{a} \langle s,a\rangle, \pi \overset{\times}{\Rightarrow} \langle\sigma_r,\otimes\rangle) := T(\Pi,\pi)\cdot S_1(\Pi)(a)$.
  Set $T(\Pi \xrightarrow{\circ} \langle s,\circledcirc\rangle, \pi \overset{\times}{\Rightarrow} \langle\sigma_r,\otimes\rangle) := T(\Pi,\pi)\cdot S_1(\Pi)(\circ)$.

- $\sigma \in F$:

  (S) : Set $S(\pi)(\circ) := S_1(\Pi)(\circ)$.
  Set $S(\pi)(\times) := 1 - S_1(\Pi)(\circ)$.

  (T) : No $a \in \mathcal{C}$ is enabled by σ (final states do not enable program guards).
  For $a \in \mathcal{C}$: set $T(\Pi \xrightarrow{a} \langle s,a\rangle, \pi \overset{\times}{\Rightarrow} \langle\sigma_r,\otimes\rangle) := T(\Pi,\pi)\cdot S_1(\Pi)(a)$.
  Set $T(\Pi \xrightarrow{\circ} \langle s,\circledcirc\rangle, \pi \overset{\circ}{\Rightarrow} \langle\sigma_f,\circledcirc\rangle) := T(\Pi,\pi)\cdot S_1(\Pi)(\circ)$.

(Note that, although there might be two paths $\Pi_1 \neq \Pi_2$ with $T(\Pi_i, \pi \overset{\times}{\Rightarrow} \langle \sigma_r, \otimes \rangle) > 0$, $i \in \{1,2\}$ according to the last definition, S is still well-defined: there is only one strategy choice for $S(\pi \overset{\times}{\Rightarrow} \sigma_r)$.)

Case 2: $\Pi = \hat{\Pi} \xrightarrow{a} \langle s, a \rangle, a \in \mathcal{C}$**.**

We give the following definitions for every $\pi \in \text{Paths}^{\mathcal{G}}$ with $T(\Pi, \pi) > 0$.
With property (*), two cases arise:

$(a):$ $\langle \sigma_r, \otimes \rangle$ last state of π: Then $\langle s, a \rangle \xrightarrow{\times} \otimes$ in $\mathcal{G}$ (property (*)).

$(T):$ Set $T(\Pi \xrightarrow{\times} \otimes, \pi \overset{\times}{\Rightarrow} \sigma_r) := T(\Pi, \pi)$.

$(b):$ $\langle \sigma, a \rangle$ last state of π: Then $\sigma \in \gamma(s)$; let $d = \beta(s, a, \sigma)$. Let $up_a = \langle \langle p_1, c_1 \rangle, \ldots, \langle p_k, c_k \rangle \rangle$.

$(T):$ Set for every $1 \leq i \leq k$:

$$T(\Pi \xrightarrow{d} \langle s, a, d \rangle \xrightarrow{i} d(i), \pi \overset{i}{\Rightarrow} [\![c_i]\!](\sigma)) := T(\Pi, \pi) \cdot p_i.$$

We finally define $S_2(\Pi)$:

$(S_2):$ Let M_a be the set of all paths π satisfying (b).
Set $S_2(\Pi)(\times) := 1 - \frac{1}{t(\Pi)} \cdot \sum_{\pi \in M_a} T(\Pi, \pi)$ and
for every possible sequence d:
let $M_{a,d}$ be the paths $\pi \in M_a$ having a last state $\langle \sigma, a \rangle$ s.t. $\beta(s, a, \sigma) = d$.
Set $S_2(\Pi)(d) := \frac{1}{t(\Pi)} \cdot \sum_{\pi \in M_{a,d}} T(\Pi, \pi)$.

Case 3a: $\Pi = \hat{\Pi} \xrightarrow{\circ} \langle s, \circledcirc \rangle$**.**

We carry out the following definitions for every $\pi \in \text{Paths}^{\mathcal{G}}$ with $T(\Pi, \pi) > 0$.
We know that π ends either with $\langle \sigma_f, \circledcirc \rangle$ or with $\langle \sigma_r, \otimes \rangle$, due to property (*).
Then there is only one choice for $S(\pi)$ (taking $\circ$ respectively $\times$).
It holds that $t(\Pi) = t_f(\Pi) + t_r(\Pi)$. If $t_f(\Pi) > 0$ we know due to property (*) and the definition of abstract game arenas that $\langle s, \circledcirc \rangle \xrightarrow{\times} \langle s, \circledcirc \rangle$ holds.

$(S_2):$ Set $S_2(\Pi)(\circ) := \frac{t_f(\Pi)}{t(\Pi)}$ and set $S_2(\Pi)(\times) := \frac{t_r(\Pi)}{t(\Pi)}$.

(T) : Set $T(\Pi \xrightarrow{\circ} \circledcirc, \pi \overset{\circ}{\Rightarrow} \sigma_f) := T(\Pi, \pi)$ if $\mathrm{last}(\pi) = \langle \sigma_f, \circledcirc \rangle$ and set $T(\Pi \xrightarrow{\times} \langle s, \circledcirc \rangle, \pi \overset{\times}{\Rightarrow} \sigma_r) := T(\Pi, \pi)$ otherwise.

Case 3b: $\Pi = \hat{\Pi} \rightarrow \langle s, \circledcirc \rangle \xrightarrow{\times} \langle s, \circledcirc \rangle$.

We carry out the following definitions for every $\pi \in \mathrm{Paths}^{\mathcal{G}}$ with $T(\Pi, \pi) > 0$. We know that π ends with σ_r, due to property (*).

(S_2) : Set $S_2(\Pi)(\times) := 1$.

(T) : Set $T(\Pi \xrightarrow{\times} \langle s, \circledcirc \rangle, \pi \overset{\times}{\Rightarrow} \sigma_r) := T(\Pi, \pi)$.

Case 4: $\Pi = \hat{\Pi} \xrightarrow{\times} \otimes$ respectively $\Pi = \hat{\Pi} \xrightarrow{\circ} \circledcirc$.

We carry out the following definitions for every $\pi \in \mathrm{Paths}^{\mathcal{G}}$ with $T(\Pi, \pi) > 0$. We know that π ends with σ_r respectively σ_f due to property (*).

(S) : Set $S(\pi)(\times) := 1$ respectively $S(\pi)(\circ) := 1$ (the only choice).

(T) : Set $T(\Pi \xrightarrow{\times} \otimes, \pi \overset{\times}{\Rightarrow} \sigma_r) := T(\Pi, \pi)$ respectively $T(\Pi \xrightarrow{\circ} \circledcirc, \pi \overset{\circ}{\Rightarrow} \sigma_f) := T(\Pi, \pi)$.

This concludes the construction.

S is well-defined using the same arguments as in the proof of part (2) and the additional comment in case 1. Again we can prove that for every $\Pi \in \mathrm{Paths}^{\mathcal{G},(S_1,S_2)}$ it holds that:

$$\mathrm{Pr}^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\Pi, \mathcal{G})] = \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\Pi, \pi).$$

The proof is essentially the same as the one of Lemma 22, i.e. we can prove it again using induction over the structure of $\Pi \in \mathrm{Paths}^{\mathcal{G},(S_1,S_2)}$. We only consider some cases that are nontrivial and substantially different from the proof in Lemma 22:

$\underline{\Pi = \hat{\Pi} \to s \xrightarrow{\circ} \langle s, \circledcirc \rangle \xrightarrow{\circ} \circledcirc}$:

$$
\begin{aligned}
&\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\Pi, \mathcal{G})] \\
&= \Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi} \to s \xrightarrow{\circ} \langle s, \circledcirc \rangle, \mathcal{G})] \cdot S_2(\hat{\Pi} \to s \xrightarrow{\circ} \langle s, \circledcirc \rangle)(\circledcirc) \\
&= \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi} \to s \xrightarrow{\circ} \langle s, \circledcirc \rangle, \pi) \cdot \frac{t_f(\hat{\Pi} \to s \xrightarrow{\circ} \langle s, \circledcirc \rangle)}{t(\hat{\Pi} \to s \xrightarrow{\circ} \langle s, \circledcirc \rangle)} && \text{(IH, case 3a)} \\
&= t_f(\hat{\Pi} \to s \xrightarrow{\circ} \langle s, \circledcirc \rangle) \\
&= \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\Pi, \pi) && \text{(Def. of } T)
\end{aligned}
$$

$\underline{\Pi = \hat{\Pi} \xrightarrow{a} \langle s, a \rangle \xrightarrow{\times} \otimes, a \in \mathcal{C}}$:

$$
\begin{aligned}
&\Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi} \xrightarrow{a} \langle s, a \rangle \xrightarrow{\times} \otimes, \mathcal{G})] \\
&= \Pr^{\mathcal{G},(S_1,S_2)}[\mathrm{Cyl}(\hat{\Pi} \xrightarrow{a} \langle s, a \rangle, \mathcal{G})] \cdot S_2(\hat{\Pi} \xrightarrow{a} \langle s, a \rangle)(\times) \\
&= \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\hat{\Pi} \xrightarrow{a} \langle s, a \rangle, \pi) \cdot S_2(\hat{\Pi} \xrightarrow{a} \langle s, a \rangle)(\times) && \text{(IH)} \\
&= t(\hat{\Pi} \xrightarrow{a} \langle s, a \rangle, \pi) \cdot \left(1 - \frac{1}{t(\hat{\Pi} \xrightarrow{a} \langle s, a \rangle)} \cdot t_r(\hat{\Pi} \xrightarrow{a} \langle s, a \rangle)\right) && \text{(Def. of } S_2\text{, case 2)} \\
&= t_r(\hat{\Pi} \xrightarrow{a} \langle s, a \rangle) && \text{(Def. of } T\text{, case 2)} \\
&= \sum_{\pi \in \mathrm{Paths}^{\mathcal{M},S}} T(\Pi, \pi) && \text{(Def. of } T \text{ and } t_r).
\end{aligned}
$$

We can prove that for every S-possible path π in $\mathcal{M}$ it holds that

$$\Pr^{\mathcal{M},S}[\mathrm{Cyl}(\pi, \mathcal{M})] = \sum_{\Pi \in \mathrm{Paths}^{\mathcal{G},(S_1,S_2)}} T(\Pi, \pi).$$

by using again a similar proof as for Lemma 23, i.e. we prove it by induction over the structure of a S-possible path π. We again give only some differing nontrivial cases here:

$\underline{\pi = \hat{\pi} \overset{i}{\Rightarrow} \sigma \overset{\circ}{\Rightarrow} \langle \sigma_f, \circledcirc \rangle, i \in \mathbb{N}:}$ Then $\sigma \in F$ due to property (*).

$$\begin{aligned}
&\Pr^S[\mathrm{Cyl}(\pi, \mathcal{M})] \\
&= \Pr^S[\mathrm{Cyl}(\hat{\pi} \overset{i}{\Rightarrow} \sigma, \mathcal{M})] \cdot S(\hat{\pi} \overset{i}{\Rightarrow} \sigma)(\circ) \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi}, \hat{\pi} \overset{i}{\Rightarrow} \sigma) \cdot S(\hat{\pi} \Rightarrow \sigma)(\circ) && \text{(IH)} \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi}, \hat{\pi} \overset{i}{\Rightarrow} \sigma) \cdot S_1(\hat{\Pi})(\circ) && \text{(Def. of } T\text{, prop. (*))} \\
&= \sum_{\Pi \in \mathrm{Paths}^{(S_1,S_2)}} T(\Pi, \pi).
\end{aligned}$$

$\underline{\pi = \hat{\pi} \overset{i}{\Rightarrow} \sigma \overset{\times}{\Rightarrow} \langle \sigma_r, \otimes \rangle, i \in \mathbb{N}:}$ Let us first assume that $\sigma \notin F$:

$$\begin{aligned}
&\Pr^S[\mathrm{Cyl}(\pi, \mathcal{M})] \\
&= \Pr^S[\mathrm{Cyl}(\hat{\pi} \overset{i}{\Rightarrow} \sigma, \mathcal{M})] \cdot S(\hat{\pi} \overset{i}{\Rightarrow} \sigma)(\times) && \text{(structure of } \mathcal{M}\text{)} \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi}, \hat{\pi} \overset{i}{\Rightarrow} \sigma) \cdot S(\hat{\pi} \overset{i}{\Rightarrow} \sigma)(\times) && \text{(IH)} \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi}, \hat{\pi} \overset{i}{\Rightarrow} \sigma) \cdot \left(1 - \sum_{a \in G_\sigma} S_1(\hat{\Pi})(a)\right) && \text{(Def. of } S \text{)} \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi}, \hat{\pi} \overset{i}{\Rightarrow} \sigma) \cdot \left(\sum_{a \notin G_\sigma} S_1(\hat{\Pi})(a) + S_1(\hat{\Pi})(\circ)\right) \\
&= \sum_{\Pi \in \mathrm{Paths}^{(S_1,S_2)}} T(\Pi, \pi) && \text{(Def. of } T\text{).}
\end{aligned}$$

For $\sigma \in F$, no $a \in \mathcal{C}$ is enabled by σ (this is a basic assumption of Problem 3).

$$\begin{aligned}
&\Pr^S[\mathrm{Cyl}(\pi, \mathcal{M})] \\
&= \Pr^S[\mathrm{Cyl}(\hat{\pi} \overset{i}{\Rightarrow} \sigma, \mathcal{M})] \cdot S(\hat{\pi} \overset{i}{\Rightarrow} \sigma)(\times) && \text{(structure of } \mathcal{M}\text{)} \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi}, \hat{\pi} \overset{i}{\Rightarrow} \sigma) \cdot S(\hat{\pi} \overset{i}{\Rightarrow} \sigma)(\times) && \text{(IH)} \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi}, \hat{\pi} \overset{i}{\Rightarrow} \sigma) \cdot (1 - S_1(\hat{\Pi})(\circ)) && \text{(Def. of } S\text{)} \\
&= \sum_{\hat{\Pi} \in \mathrm{Paths}^{(S_1,S_2)}} T(\hat{\Pi}, \hat{\pi} \overset{i}{\Rightarrow} \sigma) \cdot \sum_{a \in \mathcal{C}} S_1(\hat{\Pi})(a) \\
&= \sum_{\Pi \in \mathrm{Paths}^{(S_1,S_2)}} T(\Pi, \pi) && \text{(Def. of } T\text{)}
\end{aligned}$$

A similar argument as in the proof for part (2) concludes the proof of this part, by letting $F_{\mathcal{G}}$ the set of (S_1, S_2)-possible paths Π in $\mathcal{G}$ such that Π ends in $\circledcirc$, $F_{\mathcal{M}}$ the set of S-possible runs π such that π ends in σ_f. $\square$

Appendix B

Missing Proofs of Chapter 5

Theorem 7, (1). *Let $P = (\mathcal{L}, I, \hookrightarrow, \mathit{label}, \bot, \top)$ be a PIP.*

(1) Let Φ be a pattern. The set of Φ-conforming runs has probability 1. In particular, if Φ is terminating, then P is a.s.-terminating.

$\square$

Proof (of Theorem 7 (1)).

Let $P = (\mathcal{L}, I, \hookrightarrow, \text{label}, \bot, \top)$, and $\mathcal{M}_P = (Q, \to, \text{Lab}_P)$ its corresponding MDP. Let $\Phi = C^* w_1 C^* w_2 C^* \ldots$; the set of runs conforming to Φ are denoted by $\text{Runs}(\Phi)$. We first prove that $\text{Runs}(\Phi)$ is measurable, i.e. contained in $\mathfrak{S}^{\mathcal{M}}_P$. Let $I_C \in \mathfrak{S}^{\mathcal{M}_P}$ be the set of runs r containing infinitely many coin tosses, i.e. $\bar{r}|_C \in C^\omega$. For $w_1, w_2, \ldots w_i$, $i \geq 1$, we define the set $M(w_1, w_2, \ldots, w_i)$ by

$$M(w_1, w_2, \ldots, w_i) := \{r \in \text{Runs}^{\mathcal{M}_P} \mid \bar{r}|_C \in C^* w_1 C^* w_2 C^* \ldots C^* w_i C^\omega\}.$$

$M(w_1, w_2, \ldots, w_i)$ is measurable: let $\text{NC}(i) \in \mathfrak{S}^{\mathcal{M}_P}$ be the set of all runs r whose

i-th label is not in C, and let $F(i,c) \in \mathfrak{S}^{\mathcal{M}_P}$ be the set of runs that have c as i-th label. Set

$$G(b^-, b^+, c_1 \ldots c_k) = \bigcup_{b^- \leq a_1 < \ldots < a_k < b^+} \Bigg(\bigcap_{\substack{l > a_1 \\ \wedge l \notin \{a_2, \ldots, a_k\}}} \mathrm{NC}(l) \cap \bigcap_{1 \leq j \leq k} F(a_j, c_j) \Bigg) \in \mathfrak{S}^{\mathcal{M}_P},$$

for $\{b^-, b^+, k\} \subseteq \mathbb{N}$ and $c_1 \ldots c_k \in C^k$, the set of runs where between the b^--th and the b^+-th label the subword $c_1 \ldots c_k$ can be observed. $M(w_1, w_2, \ldots, w_n)$ can be written as

$$I_C \cap \bigcup_{1 \leq b_1^- < b_1^+ < b_2^- < \ldots < b_n^- < b_n^+} G(b_1^-, b_1^+, w_1) \cap \ldots \cap G(b_n^-, b_n^+, w_n) \in \mathfrak{S}^{\mathcal{M}_P}.$$

Since

$$\mathrm{Runs}(\Phi) = \Big(\mathrm{Runs}^{\mathcal{M}_P} \setminus I_C\Big) \cup \bigcap_{i \geq 1} M(w_1, w_2, \ldots, w_i) \in \mathfrak{S}^{\mathcal{M}_P}$$

we conclude that $\mathrm{Runs}(\Phi)$ is also measurable.

Next we show that $\mathrm{Pr}^S[\mathrm{Runs}(\Phi)] = 1$ for every strategy S. We abbreviate Pr^S by Pr in the following.

We first show that for every prefix $w_1, w_2, w_3, \ldots, w_i$ of $(w_i)_{i \in \mathbb{N}}$, $\mathrm{Pr}[M(w_1, \ldots, w_i)] = \mathrm{Pr}[I_C]$ holds, i.e., the set of runs that visit probabilistic states infinitely often, but are *not* $C^* w_1 C^* w_2 C^* \ldots C^* w_i C^\omega$-conforming, have probability zero.

For proving this we write $w = w_1 w_2 \ldots w_i$. Let $n = |w|$. $M(w) \subseteq M(w_1, w_2, \ldots, w_i)$ holds for all i. It suffices to show that $\mathrm{Pr}[M(w)] = \mathrm{Pr}[I_C]$, since this implies with $I_C \supseteq M(w_1, \ldots, w_i)$ that $\mathrm{Pr}[I_C] = \mathrm{Pr}[M(w_1, w_2, \ldots, w_i)]$.

Let $V(j)$ be the (measurable) set of runs that visit a probabilistic state at least

j times, and let

$$B(j) = V(j \cdot n) \cap (\mathrm{Runs}^{\mathcal{M}_P} \setminus M(w))$$

be the set of runs r that visit a probabilistic state at least $j \cdot n$ times, and w is no substring of $\bar{r}|_C$. Since there are only finitely many probabilistic locations in P, there exists a minimal probability $p_{\min} > 0$ such that for every transition $q \xrightarrow{p',c} q'$ with $c \in \{0, 1\}$ in in $\mathcal{M}_P$ it holds that $p' \geq p_{\min}$. We write $\mathrm{NI}(w)$ ("not visited initially") for the set of runs r such that $\bar{r}|_C$ does not start with w. Note that $\Pr[\mathrm{NI}(w) \mid V(n)] \leq (1 - p_{\min}^n)$: the probability that a run visiting probabilistic states at least n times does *not* observe w in the beginning is at most $1 - p_{\min}^{|w|}$. We get

$$\begin{aligned}
&\Pr[B(1)] \\
&\leq \Pr[\mathrm{NI}(w) \cap V(n)] \\
&\leq \Pr[\mathrm{NI}(w) \mid V(n)] \cdot \Pr[V(n)] \\
&\leq (1 - p_{\min}^n) \cdot \Pr[V(j)] && \text{(note above)} \\
&\leq (1 - p_{\min}^n),
\end{aligned}$$

i.e., after visiting probabilistic states at least n times, the probability p of *not* seeing the sequence w is at most $(1 - p_{min}^n) < 1$. By repeating the observation inductively we obtain $\Pr[B(j)] \leq (1 - p_{\min}^n)^j$. It holds that $B(j) \supseteq B(j + 1)$ for all j. Then

$$\Pr[\bigcap_{j \geq 1} B(j)] = \lim_{j \to \infty} \Pr[B(j)] \leq \lim_{j \to \infty} (1 - p_{min}^n)^j = 0. \qquad \text{(B.1)}$$

We can write $M(w) = I_C \setminus \bigcap_{j\geq 0} B(j)$. Hence

$$\begin{aligned}
\Pr[M(w_1w_2\dots w_i)] &= \Pr[I_C \setminus \bigcap_{j\geq 1} B(j)] && \text{(Def. of } B(\cdot))\\
&= \Pr[I_C] - \Pr[\bigcap_{j\geq 1} B(j) \cap I_C]\\
&= \Pr[I_C] && \text{(Eq. B.1).}
\end{aligned}$$

Now $\Pr[I_C \setminus M(w_1,\dots,w_i)] = \Pr[I_C] - \Pr[M(w_1,\dots,w_i)] = 0$. We can write

$$I_C \setminus \bigcap_{i\geq 1} M(w_1,\dots,w_i) = I_C \cap \bigcup_{i\geq 1} I_C \setminus M(w_1,\dots,w_i).$$

For every $i \geq 1$, $I_C \setminus M(w_1,\dots,w_i)$ is a null set, thus the countable union $\bigcup_{i\geq 1} I_C \setminus M(w_1,\dots,w_i)$ is also a null set, and $\Pr[\bigcap_{i\geq 0} M(w_1,\dots,w_i)] = \Pr[I_C]$ holds (*). We conclude:

$$\begin{aligned}
&\Pr[\mathrm{Runs}(\Phi)]\\
&= \Pr[\mathrm{Runs}^{\mathcal{M}_P} \setminus I_C] + \Pr[\bigcap_{i\geq 0} M(w_1,\dots,w_i)]\\
&= \Pr[\mathrm{Runs}^{\mathcal{M}_P} \setminus I_C] + \Pr[I_C] && (*)\\
&= 1.
\end{aligned}$$

$\square$

Theorem 9. *(Response pattern as a proof method)*

Let P be a PIP in normal form.

(1) Let Φ be a response pattern. The set of Φ-conforming runs has probability 1 *for every strategy S for $\mathcal{M}_P$. In particular, if P has a terminating response pattern, then P is a.s.-terminating.*

(3) If P is a.s.-terminating and finite with $n < \infty$ reachable states in $\mathcal{M}_P$, then there exists a response R of length in $\mathcal{O}(n^2)$ such that $(AC)^*R(AC)^\omega$ is terminating for P and $((AC)^*R)^\omega$ is a simple terminating response pattern for P.

(2) If P is a.s.-terminating and weakly finite, then the universal response pattern is terminating for P.

$\square$

Proof.

Let $P = (\mathcal{L}, I, \hookrightarrow, \text{label}, \bot, \top)$. The MDP corresponding to P is denoted by $\mathcal{M}_P = (Q_A, Q_D, \text{Init}, \to, \text{Lab}_A, \text{Lab}_P)$. Let $\Phi = (AC)^*R_1(AC)^*R_2\ldots$, with R_i a response for all $i \geq 1$. We call the set of Φ-corresponding runs $\text{Runs}(\Phi)$. For responses R_1, R_2 of length n_1 and n_2, respectively, and a word $w \in (AC)^+$, we set $w \circ R_1 := \{wr \mid r \in R_1\}$ and $R_1 \circ R_2 := \{r \circ R_2 \mid r \in R_1\}$. $R_1 \circ R_2$ is a response of length $n_1 + n_2$. We set $G := A \cup C$. Recall that, since P is in normal form, for every run r in $\mathcal{M}_P$, $\bar{r}|_G$ is a prefix of a word in $(AC)^\omega$.

Part (1):

We first prove that $\text{Runs}(\Phi)$ is measurable (the proof is very similar to the one of Theorem 7 (1)). Let $I_G \in \mathfrak{S}^{\mathcal{M}_P}$ be the set of runs r with $\bar{r}|_G \in G^\omega$. For $R_1, R_2, \ldots R_i$, $i \geq 1$, we define the set $M(R_1, R_2, \ldots, R_i)$ by

$$M(R_1, R_2, \ldots, R_i) := \{r \in \mathrm{Runs}(\mathcal{M}_P) \mid \bar{r}|_G \in G^* R_1 C^* R_2 G^* \ldots G^* R_i G^\omega\}.$$

$M(R_1, R_2, \ldots, R_i)$ is measurable: the set of runs r such that

$$\bar{r}|_M \in G^* w_1 G^* w_2 G^* \ldots G^* w_i G^\omega$$

for $(w_1, \ldots, w_i) \in R_1 \times \ldots \times R_i$ is measurable, which can be proved as in the proof of Theorem 7 (1). $M(R_1, R_2, \ldots, R_i)$ is the finite union of all these sets and thus is also measurable. Again, since

$$\mathrm{Runs}(\Phi) = \Big(\mathrm{Runs}(\mathcal{M}_P) \setminus I_G\Big) \cup \bigcap_{i \geq 0} M(R_1, R_2, \ldots, R_i) \in \mathfrak{S}^{\mathcal{M}_P}$$

we conclude that $\mathrm{Runs}(\Phi)$ is measurable.

Let S be a strategy for $\mathcal{M}_P$. We show that $\mathrm{Pr}^S[\mathrm{Runs}(\Phi)] = 1$, again reusing ideas from the proof of Theorem 7. We abbreviate Pr^S again by Pr.

For every prefix $R_1, R_2, R_3, \ldots, R_i$ of $(R_i)_{i \in \mathbb{N}}$, we show that $\mathrm{Pr}[M(R_1, \ldots, R_i)] = \mathrm{Pr}[I_G]$ holds, i.e., the set of runs that visit probabilistic states infinitely often, but are *not* conforming to $(AC)^* R_1 (AC)^* \ldots (AC)^* R_i (AC)^\omega$, have probability zero.

For proving this we write $R = R_1 \circ R_2 \circ \ldots R_i$. Let n be the length of R. $M(R) \subseteq M(R_1, R_2, \ldots, R_i)$ holds for all i. Again it suffices to show that $\mathrm{Pr}[M(R)] = \mathrm{Pr}[I_G]$, since this implies with $I_G \supseteq M(R_1, \ldots, R_i)$ that

$$\mathrm{Pr}[I_G] = \mathrm{Pr}[M(R_1, R_2, \ldots, R_i)].$$

We reuse the definition of the sets of runs $V(j)$ that visit a probabilistic state at least j times, and set

$$B(j) = V(j \cdot n) \cap (\mathrm{Runs}(\mathcal{M}_P) \setminus M(R))$$

for the set of runs r that visit a probabilistic state at least $j \cdot n$ times, and no $w \in R$ is a substring of $\bar{r}|_G$. Again we use the fact that there exists a minimal probability $p_{min} > 0$ such that for every transition $q \xrightarrow{p',c} q'$ in $\mathcal{M}_P$, $c \in \{0,1\}$, $p' \geq p_{\min}$ holds. For $x \in A^*$ we write $SC(x)$ ("strategy choice") for the set of runs r such that $\bar{r}|_A$ starts with x. For $x \neq x'$ with $x, x' \in A^*$ having the same length,

$$SC(x) \cap SC(x') = \emptyset. \tag{B.2}$$

Let $\mathrm{NI}(w)$ be again the set of runs r such that $\bar{r}|_C$ does not start with $w \in C^*$. If the strategy S initially chooses actions according to the actions in $\bar{w}|_A$, we get similarly to the proof of Theorem 7 (1):

$$\Pr[\mathrm{NI}(\bar{w}|_C) \mid \mathrm{SC}(\bar{w}|_A) \cap V(n)] \leq (1 - p^n_{\min}). \tag{B.3}$$

It also holds that

$$B(1) \subseteq V(n) \cap \bigcup_{w \in R} \left(\mathrm{NI}(\bar{w}|_C) \cap \mathrm{SC}(\bar{w}|_A)\right).$$

With this we obtain

$$
\begin{aligned}
&\Pr^S[B(1)] \\
&\leq \sum_{w\in R} \Pr[\bigcup_{w\in R} \mathrm{NI}(\bar{w}|_C) \cap SC(\bar{w}|_A) \cap V(n)] \\
&\leq \sum_{w\in R} \Pr[\mathrm{NI}(\bar{w}|_C) \mid \mathrm{SC}(\bar{w}|_A) \cap V(n)] \cdot \Pr[\mathrm{SC}(\bar{w}|_A) \cap V(n)] \\
&\leq \sum_{w\in R} (1 - p_{\min}^n) \cdot \Pr^S[SC(\bar{w}|_A) \cap V(j)] && \text{(Eq. B.2, Eq. B.3)} \\
&\leq (1 - p_{\min}^n).
\end{aligned}
$$

Again we can see that after visiting probabilistic states at least n times, the probability of *not* seeing at least *one* of the $w \in R$ is at most $(1 - p_{min}^n) < 1$. In $B(j)$, we repeat this experiment at least j times (at positions $1, n, 2n, \ldots$) and get again $\Pr[B(j)] \leq (1 - p_{min}^n)^j$. Now we proceed exactly as in the proof of Theorem 7, substituting I_C by I_G and $M(w_1, \ldots w_i)$ by $M(R_1, \ldots, R_i)$, and obtain $\Pr[I_G] = \Pr^S[\bigcap_{i\geq 0} M(R_1, \ldots, R_i)]$. We conclude

$$
\begin{aligned}
&\Pr[\mathrm{Runs}(\Phi)] \\
&= \Pr[\mathrm{Runs}(\mathcal{M}_P) \setminus I_G] + \Pr[\bigcap_{i\geq 0} M(R_1, \ldots, R_i)] \\
&= \Pr[\mathrm{Runs}(\mathcal{M}_P) \setminus I_G] + \Pr[I_G] \\
&= 1.
\end{aligned}
$$

Part (2):

We reintroduce several notations from the proof of Theorem 7 and generalize them to accomodate nondeterminism. We redefine the term “ending up” as follows:

$q \in Q$ *ends up in* a state $q' \in Q$ *following* $w = x_1 x_2 \dots x_m \in G^*$ iff

$$q \xrightarrow{\tau^* x_1 \tau^* x_2 \tau^* \dots \tau^* x_m \tau^*} q' \text{ and either}$$

$$q' = \langle \ell, \sigma \rangle \text{ with } \ell \text{ } \textit{probabilistic} \text{ or } \textit{nondeterministic}, \text{ or } q' = \top.$$

Again, if such a q' exist, it is unique, since all transition choices are resolved. For every reachable state $q \in Q_A$ and every sequence $w \in (AC)^*$ holds that either: (i) q ends up in a state $\langle \ell, \sigma \rangle$ with ℓ probabilistic or nondeterministic following w, or (ii) q ends up in $\top$ following a prefix of w. Otherwise there exists a state q' from which no state with probabilistic or nondeterministic location or $\top$ is reachable any more, which contradicts that P is a.s.-terminating. Note that there always exists a strategy that is able to cause the initial state of $\mathcal{M}_P$ to end up in q' with nonzero probability, using the nondeterministic choices given in w. We show that for every state $q \in Q_A$ and every sequence $s_1 \dots s_n \in A^n$ there exists a $c_1 c_2 \dots c_n$ such that q ends up in $\top$ following a prefix of $s_1 c_1 \dots s_n c_n$. Assume for the sake of contradiction that there exists $q \in Q_A$ and a sequence $s_1 \dots s_n \in A^n$ for which no $c_1 \dots c_n$ exists with the property described above. We can then construct a strategy S that assures:

(i) q is reached with probability > 0, and

(ii) every run that reaches q never reaches $\top$.

The probability of reaching $\top$ is then smaller than 1, contradicting the assumption that P is a.s.-terminating. Since q is reachable in $\mathcal{M}_P$, there exists a cycle-free path π from the initial state q_0 to q. For all proper path prefixes of π ending in a nondeterministic state, S selects the corresponding choices contained in π with probability 1, and thus we reach q with probability > 0. This proves (i). For (ii), let π be a path having the form $\pi = \pi' \to q_1 \xrightarrow{l_1} q_2 \xrightarrow{l_2} \dots \xrightarrow{l_m} q_m$, with $m \geq 1$ and $q_1 = q$, such that π' does not contain q. We define $S(\pi)$ as follows: let π_r be the path obtained from $q_1 \xrightarrow{l_1} q_2 \xrightarrow{l_2} \dots \xrightarrow{l_m} q_m$ by removing all possible cycles. π_r then contains $k < n$ nondeterministic states (there are only n states in total). Set $S(\pi)(s_k) = 1$. Then there is no path starting from a reachable state $\pi' \to q$ in $\mathcal{M}_P[S]$ that reaches $\top$ (more exactly, that reaches a state $\pi'' \to \top$), contradicting the assumption that P is a.s.-terminating: after reaching q, S can avoid reaching $\top$ entirely.

We now select a $c_1 \dots c_n \in C^n$ with the property described above for each $q \in Q_A$ and $s_1 \dots s_n \in A^n$, and define $tr(q, s_1 \dots s_n) := s_1c_1 \dots s_nc_n$. We set

$$R(q) := \{tr(q, w) \mid w \in A^n\}.$$

Note that every $R(q)$ is a response, and for every $w \in R(q)$, q ends up in $\top$ following a prefix of w. We say that a response with this property *leads* q to $\top$. We construct now a sequence $R^{(0)}, R^{(1)}, \dots, R^{(m)}$ using the following algorithm. Set $R^{(0)} := \{\epsilon\}$ and $i := 1$.

1. Pick a $q_i' \in Q_A$ that does end up in a state $q_i \neq \top$ following a $w \in R^{(i-1)}$. If no such q_i exists set $R := R^{(i-1)}$ and terminate.

2. Set $R^{(i)} := (R^{(i-1)} \setminus \{w\}) \cup w \circ R(q_i)$. Set $i := i + 1$ and go to (1).

We show that for every i, if $w \in R^{(i)}$, $|w| \leq n^2$. This implies termination of the algorithm.

Let $w \in R^{(i)}$. Let $q_1', \dots, q_m'$ be the states selected in part (1) of the algorithm such that $w = w_1 w_2 \dots w_m$ with $w_j \in R(q'_j)$ for $1 \leq j \leq m$. We define a family of sets by:

- $Q^{(0)} = Q_A \cup \{\top\}$,
- for every $j \geq 1$, $Q^{(j)}$ is the set of states consisting of $\top$ and all states $\hat{q}$ such that there exists a $q \in Q_A$ that ends up in $\hat{q}$ following $w_1 \dots w_j$.

For every $j \geq 1$, $|Q^{(j)}| \geq 1$. We now prove that $Q^{(j)}$ contains at most $n - j$ states. This is true for $Q^{(0)}$. For $j > 0$, note that q'_j is chosen such that w_j or one of its prefixes leads a state q in $Q^{(j-1)}$ to $\top$. That implies $|Q^{(j)}| < |Q^{(j-1)}|$, and therefore the property (recall that every state ends up in at most one state following a sequence).
Thus m has to be smaller than n, and $|w| \leq n^2$, since $R(q)$ has length n for all q. Note that for every $w, w' \in R^{(i)}$ for all $i \geq 0$, if $w \neq w'$ then $w|_A \neq w'|_A$. Hence after termination of the procedure, we can replace every $w \in R$ such that $|w| = k \cdot n < n^2$ by $w \circ R'$, with R' an arbitrary response of length $(n - k) \cdot n$, to obtain equal length of all words in R, which then forms a response of length n^2. For every $w \in R$, every state of Q_A ends up in $\top$ after following a prefix of w. We can conclude that every run r with $\bar{r}|_G$ a prefix of a word in $(AC)^* R (AC)^\omega$ is terminating, and thus $(AC)^* R (AC)^\omega$ is a terminating pattern, and so also $((AC)^* R)^\omega$.

Part (3):

The proof proceeds analogously to the one of Theorem 7, part (3): Let $\sigma_1, \sigma_2, \dots$ be a (countable or infinite) enumeration of the states in I. With Part (2) we obtain for each $i \geq 1$ a response R_i such that $(AC)^* R_i (AC)^\omega$ is a terminating pattern for P, if the only starting state considered is σ_i. By its definition, the universal pattern is a subset of $(AC)^*R_i(AC)^\omega$ for every $i \geq 1$, so it is also terminating.

$\square$

Appendix C

Missing Proofs of Chapter 6

Lemma 24. *($\mathcal{M}[\mathcal{B}]$ is well-defined for every MFBP $\mathcal{B}$)*
For every MFBP $\mathcal{B} = (Pr_1, Pr_2, \ldots, Pr_n)$ with n types, $\mathcal{M}[\mathcal{B}]$ defined in Def. 40 forms a Markov chain. □

Proof.
Let $\mathcal{B} = (\mathrm{Pr}_1, \mathrm{Pr}_2, \ldots, \mathrm{Pr}_n)$ and $\mathcal{M}[\mathcal{B}] = (\mathbb{P}, \rightarrow, \{\tau\})$, We have to verify conditions (1), (2) of Definition 8. As before we set $\mathbb{P} = \mathbb{N}^n$. Given states $\mathbf{x}$ and $\mathbf{y}$ of $\mathcal{M}[\mathcal{B}]$, there is at most one transition $(\mathbf{x}, p, \tau, \mathbf{y})$ in $\rightarrow$. For proving (2), we first note that

$$\sum_{\mathbf{w}\in\mathbb{P}} g(\mathbf{w}, i, k) = 1 \tag{C.1}$$

for every i and k. We compute for a given $\mathbf{x} \in Q$:

$$
\begin{aligned}
&\sum_{(\mathbf{x},p,k,\mathbf{y}) \in \to} p \\
&= \sum_{\mathbf{y} \in \mathbb{P}} \sum_{\substack{(\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}) \in \mathbb{P}^n: \\ \mathbf{w}^{(1)}+\ldots+\mathbf{w}^{(n)}=\mathbf{y}}} \left(\prod_{i=1}^{n} g(\mathbf{w}^{(i)}, i, \mathbf{x}_i) \right) && \text{(Def. 40)} \\
&= \sum_{(\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}) \in \mathbb{P}^n} \left(\prod_{i=1}^{n} g(\mathbf{w}^{(i)}, i, \mathbf{x}_i) \right) && \text{(Sum over arbitrary } \mathbf{y}) \\
&= \sum_{\mathbf{w}^{(1)} \in \mathbb{P}} g(\mathbf{w}^{(1)}, 1, \mathbf{x}_1) \cdot \sum_{\mathbf{w}^{(2)},\ldots,\mathbf{w}^{(n)}} \left(\prod_{i=2}^{n} g(\mathbf{w}^{(i)}, i, \mathbf{x}_i) \right) \\
&= 1 \cdot \sum_{\mathbf{w}^{(2)},\ldots,\mathbf{w}^{(n)}} \prod_{i=2}^{n} g(\mathbf{w}^{(i)}, i, \mathbf{x}_i) && \text{(Eq. C.1)} \\
&= 1 && \text{(Apply Eq. C.1 iteratively).}
\end{aligned}
$$

$\square$

Theorem 13.
Let f be a PSP of size s. We can compute in time $O(n \cdot s)$ a perfectly superlinear PSP $\tilde{f}$ with $\mathit{Var}(\tilde{f}) = \mathit{Var}(f) \cup \{\tilde{X}\}$ of size $O(n \cdot s)$ such that $\mu_f = (\mu_{\tilde{f}})_{\mathit{Var}(f)}$. $\square$

Proof.

In a first step, we add to the equation system $\overline{X} = f(\overline{X})$ an $(n+1)$-st equation $\tilde{X} = \frac{1}{3}\tilde{X}^2 + \frac{2}{3}$. The least solution of this equation is $\tilde{X} = 1$. $\{\tilde{X}\}$ now forms a purely superlinear bottom SCC. We now take all components f_i that are not yet superlinear and multiply a monomial of f_i by $\tilde{X}$. For instance, if $f_i = \frac{1}{4}X_j + \frac{1}{3}X_k$, then we replace f_i with $\frac{1}{4}X_j\tilde{X} + \frac{1}{3}X_k$. This does not change the least fixed point in the non-$\tilde{X}$-components. We call the resulting PSP again f for simplicity. Notice that f is now purely superlinear.

In a second step we make sure that all superlinear SCCs are purely superlinear. For this, we repeatedly apply a certain operation: Let X_j be a variable that occurs in a monomial m of a component f_i, i.e., there is a monomial $\tilde{m}$ with $m = X_j \cdot \tilde{m}$. The operation that replaces the monomial m in f_i with $0.5 \cdot m + 0.5 \cdot f_j \cdot \tilde{m}$ is called *substituting* (an occurrence of) X_j. It is easy to see that applying this operation to a PSP yields a PSP with the same set of fixed points. Notice that substituting does not change the dependency relation between the variables.

In order to make all superlinear SCCs purely superlinear, we apply a sequence of substituting operations. Take a superlinear SCC S which is not purely superlinear and let $g(S)$ denote the PSP obtained by restricting f to the S-components and replacing all variables which are not in S by the constant 1. Since S is superlinear and not purely superlinear, the PSP $g(S)$ is, by definition, superlinear and not purely superlinear. So there exist variables $X_i, X_j \in S$ such that X_i directly depends on X_j in $g(S)$, and $g(S)_i$ is linear, and $g(S)_j$ is superlinear. Substitute the corresponding occurrence of X_j in f_i. This makes $g(S)_i$ superlinear. By proceeding this way, at most n substituting operations suffice to make all superlinear SCCs purely superlinear.

To make f perfectly superlinear, it remains to make each variable directly depend on itself. We achieve that by replacing, for all variables X, the polynomial f_X with $0.5 f_X + 0.5 X$. It is easy to see that f has the same least fixed point, the sum of the coefficients is still at most 1 in all components, and no new variable dependencies are created by this operation except that every variable now depends directly on itself. So, this operation makes f perfectly superlinear.

The bottleneck of this whole procedure are the substituting operations. Notice that computing the DAG of SCCs can be done in time $O(s)$ with Tarjan's algorithm. The size of each single polynomial at the end of the substituting procedure is $O(s)$, so the total size of the resulting PSP is $O(n \cdot s)$. □

Lemma 12.

Let f be a perfectly superlinear PSP and let $\mathbf{x} \prec f(\mathbf{x})$. Then

$$\mathcal{N}(\mathbf{x}) = \mathbf{x} + (f'(\mathbf{x}))^*(f(\mathbf{x}) - \mathbf{x}).$$

□

Proof (of Lemma 12).

By Proposition 16 we have $\mathbf{x} \prec \mu_f$. For such points $\mathbf{x}$ it was shown in Esparza et al. [37], Etessami and Yannakakis [43] that $\rho(f'(\mathbf{x})) < 1$. By standard matrix facts (Berman and Plemmons [12]), the matrix star A^* exists if and only if $\rho(A) < 1$. Furthermore, if A^* exists, it is equal to $(\mathrm{Id} - A)^{-1}$. Hence, $(Id - f'(\mathbf{x}))^{-1} = f'(\mathbf{x})^*$, and the statement follows. □

Lemma 13.

Let f be perfectly superlinear. Let $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x}) \prec \overline{1}$ and $\mathbf{y} = \mathcal{N}(\mathbf{x})$. Then $f(\mathbf{x}) \prec \mathbf{y} \prec f(\mathbf{y}) \prec \overline{1}$. □

Proof.

By Lemma 12 we have $\mathcal{N}(\mathbf{x}) = \mathbf{x} + f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x})$. Write $\boldsymbol{\Delta} = f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x})$, i.e., $\mathbf{y} = \mathbf{x} + \boldsymbol{\Delta}$. As every variable depends directly on itself, we have $f'(\mathbf{x})(f(\mathbf{x}) -$

$\mathbf{x}) \succ \overline{0}$. Consequently,

$$\begin{aligned} f(\mathbf{x}) &\prec \mathbf{x} + (f(\mathbf{x}) - \mathbf{x}) + f'(\mathbf{x})(f(\mathbf{x}) - \mathbf{x}) \\ &= \mathbf{x} + \sum_{i=0,1} f'(\mathbf{x})^i (f(\mathbf{x}) - \mathbf{x}) \\ &\leq \mathbf{x} + \sum_{i=0}^{\infty} f'(\mathbf{x})^i (f(\mathbf{x}) - \mathbf{x}) \\ &= \mathbf{x} + \mathbf{\Delta} \\ &= \mathbf{y} \; . \end{aligned}$$

Letting $\mathbf{u}, \mathbf{v}$ be any vectors, we write $f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f'(\mathbf{u})\mathbf{v} + R(\mathbf{u}, \mathbf{v})$ for the Taylor expansion of f at $\mathbf{u}$. Notice that $R(\mathbf{x}, \mathbf{\Delta}) \succ \overline{0}$, because $\mathbf{x} \succ \overline{0}$ and f is purely superlinear. Hence we have

$$\begin{aligned} \mathbf{y} &= \mathbf{x} + \mathbf{\Delta} \\ &\prec \mathbf{x} + \mathbf{\Delta} + R(\mathbf{x}, \mathbf{\Delta}) \\ &= \mathbf{x} + f'(\mathbf{x})^* (f(\mathbf{x}) - \mathbf{x}) + R(\mathbf{x}, \mathbf{\Delta}) \\ &= \mathbf{x} + (f(\mathbf{x}) - \mathbf{x}) + f'(\mathbf{x}) f'(\mathbf{x})^* (f(\mathbf{x}) - \mathbf{x}) + R(\mathbf{x}, \mathbf{\Delta}) \\ &= f(\mathbf{x}) + f'(\mathbf{x})\mathbf{\Delta} + R(\mathbf{x}, \mathbf{\Delta}) \\ &= f(\mathbf{x} + \mathbf{\Delta}) \\ &= f(\mathbf{y}) \; . \end{aligned}$$

By Etessami and Yannakakis [43], Kiefer et al. [64] we have $\mathbf{y} \leq \mu_f$. By the monotonicity of f it follows that $f(\mathbf{y}) \leq f(\mu_f) = \mu_f$. Using the monotonicity of f once more and the fact that every variable depends directly on itself, we obtain $\mathbf{y} \prec f(\mathbf{y}) \prec f(f(\mathbf{y})) \leq f(\mu_f) = \mu_f$. As $\mu_f \leq \overline{1}$, it follows $f(\mathbf{y}) \prec \overline{1}$. $\square$

Lemma 15.
Let f be a PSP. Let $\overline{0} \prec \mathbf{x} \prec f(\mathbf{x})$. Let $\mathbf{z}$ be with $f(\mathbf{x}) + f'(\mathbf{x})(\mathbf{z} - \mathbf{x}) \leq \mathbf{z}$, i.e. $\mathbf{z}$ is a strict post-fixed point of the linearization of f at $\mathbf{x}$. Then $\mathcal{N}(\mathbf{x}) \leq \mathbf{z}$ holds. $\square$

Proof.

We write $\mathbf{y} = \mathcal{N}(\mathbf{x})$ and $\mathbf{\Delta} = f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x})$. Notice that $\mathbf{y} = \mathbf{x} + \mathbf{\Delta}$. We have

$$
\begin{aligned}
&(Id - f'(\mathbf{x}))\,(\mathbf{z} - \mathbf{y}) \\
&= \mathbf{z} - \mathbf{x} - \mathbf{\Delta} + f'(\mathbf{x})(\mathbf{x} + \mathbf{\Delta} - \mathbf{z}) \\
&= \mathbf{z} - f(\mathbf{x}) + (f(\mathbf{x}) - \mathbf{x}) - f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x}) + f'(\mathbf{x})f'(\mathbf{x})^*(f(\mathbf{x}) - \mathbf{x}) \\
&\qquad + f'(\mathbf{x})(\mathbf{x} - \mathbf{z}) \\
&= \mathbf{z} - f(\mathbf{x}) + f'(\mathbf{x})(\mathbf{x} - \mathbf{z}) \\
&\geq \overline{0} \quad \text{(by assumption)}\ .
\end{aligned}
$$

It follows that

$$
\begin{aligned}
\mathbf{z} - \mathbf{y} &= (Id - f'(\mathbf{x}))^{-1}\,(Id - f'(\mathbf{x}))\,(\mathbf{z} - \mathbf{y}) \\
&= f'(\mathbf{x})^*\,(Id - f'(\mathbf{x}))\,(\mathbf{z} - \mathbf{y}) \geq f'(\mathbf{x})^*\overline{0} = \overline{0}\ ,
\end{aligned}
$$

i.e. $\mathcal{N}(\mathbf{x}) = \mathbf{y} \leq \mathbf{z}$. $\square$

References

[1] J. Anderson, P. Tataru, J. Staines, J. Hein, and R. Lyngso. „Evolving stochastic context-free grammars for RNA secondary structure prediction“. In: *BMC Bioinformatics* 13.1 (2012), p. 78.

[2] K. Apt and D. Kozen. „Limits for automatic verification of finite-state concurrent systems“. In: *Information Processing Letters* 22 (1986), pp. 307–309.

[3] T. Arons, A. Pnueli, and L.D. Zuck. „Parameterized Verification by Probabilistic Abstraction“. In: *FoSSaCS*. 2003, pp. 87–102.

[4] K. B. Athreya and P. E. Ney. *Branching Processes*. Springer, 1972.

[5] R. Bagnara, K. Dobson, P. M. Hill, M. Mundell, and E. Zaffanella. „Grids: A domain for analyzing the distribution of numerical values“. In: *LOPSTR*. 2007, pp. 219–235.

[6] R. Bagnara, P. M. Hill, and E. Zaffanella. „The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and

Verification of Hardware and Software Systems“. In: *Science of Computer Programming* 72 (2008).

[7] C. Baier and J.-P. Katoen. *Principles of model checking.* MIT Press, 2008, pp. I–XVII, 1–975.

[8] C. Baier and M. Z. Kwiatkowska. „Model Checking for a Probabilistic Branching Time Logic with Fairness“. In: *Distributed Computing* 11.3 (1998), pp. 125–155.

[9] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. „Model-checking algorithms for continuous-time Markov chains“. In: *Software Engineering, IEEE Transactions on* 29.6 (2003), pp. 524 –541.

[10] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. „Performance evaluation and model checking join forces“. In: *Commun. ACM* 53.9 (2010), pp. 76 –85.

[11] C. Baier, M. Groesser, and F. Ciesinski. „Quantitative Analysis under Fairness Constraints“. In: *ATVA*. Vol. 5799. 2009, pp. 135–150.

[12] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences.* SIAM, 1994.

[13] D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar. „The software model checker BLAST“. In: *STTT* 9.5-6 (2007), pp. 505–525.

[14] A. Bianco and L. de Alfaro. „Model checking of probabilistic and nondeterministic systems“. In: *FSTTCS*. Springer Verlag, 1995, pp. 499–513.

[15] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. „Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software“. In: *The Essence of Computation: Essays Dedicated to Neil D. Jones*. 2002, pp. 85–108.

[16] G. Bracha and S. Toueg. „Asynchronous consensus and broadcast protocols“. In: *Journal of ACM* 32 (4 1985), pp. 824–840.

[17] *Case studies for Kattenbelt, Kwiatkowska, Norman, and Parker [. 63]*. URL: `http://www.prismmodelchecker.org/files/vmcai09/`.

[18] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. „Trading Memory for Randomness“. In: *QEST*. 2004, pp. 206–217.

[19] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. „Counterexample-Guided Abstraction Refinement“. In: *CAV*. Vol. 1855. Lecture Notes in Computer Science. 2000, pp. 154–169.

[20] A. Condon. „On Algorithms for Simple Stochastic Games“. In: *Volume 13 of DIMACS Series in Discr. Math. and Theor. Comp. Sci.* AMS, 1993, pp. 51–73.

[21] A. Condon. „The Complexity of Stochastic Games“. In: *Inf. Comput.* 96.2 (1992), pp. 203–224.

[22] B. Cook, A. Podelski, and A. Rybalchenko. „Terminator: Beyond Safety“. In: *CAV*. 2006, pp. 415–418.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill Book Company, 2001.

[24] C. Courcoubetis and M. Yannakakis. „The complexity of probabilistic verification“. In: *Journal of ACM* 42.4 (July 1995).

[25] P. Cousot and R. Cousot. „Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints“. In: *POPL*. 1977, pp. 238–252.

[26] P. Cousot and R. Cousot. „Systematic design of program analysis frameworks“. In: *POPL*. 1979, pp. 269–282.

[27] P. Cousot and N. Halbwachs. „Automatic discovery of linear restraints among variables of a program“. In: *POPL*. 1978, pp. 84–97.

[28] P. Cousot, P. Ganty, and J.-F. Raskin. „Fixpoint-Guided Abstraction Refinements“. In: *SAS*. 2007, pp. 333–348.

[29] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen. „Reduction and Refinement Strategies for Probabilistic Analysis“. In: *PAPM-PROBMIV*. 2002, pp. 57–76.

[30] J.-P. Dedieu. „Estimations for the Separation Number of a Polynomial System“. In: *Journal of Symbolic Computation* 24.6 (1997), pp. 683 –693. ISSN: 0747-7171.

[31] A. Di Pierro, C. Hankin, and H. Wiklicky. „On Probabilistic Techniques for Data Flow Analysis“. In: *Electr. Notes Theor. Comput. Sci.* 190.3 (2007), pp. 59–77.

[32] J. Esparza and A. Gaiser. *Probabilistic abstractions with arbitrary domains*. Tech. rep. Available at `http://arxiv.org/abs/1106.1364`. Technische Universität München, 2011.

[33] J. Esparza and A. Gaiser. „Probabilistic Abstractions with Arbitrary Domains“. In: *SAS*. 2011, pp. 334–350.

[34] J. Esparza and S. Schwoon. „A BDD-based Model Checker for Recursive Programs“. In: *CAV*. Lecture Notes in Computer Science. 2001, pp. 324–336.

[35] J. Esparza, A. Gaiser, and S. Kiefer. *Computing Least Fixed Points of Probabilistic Systems of Polynomials*. Tech. rep. Available at `http://arxiv.org/abs/0912.4183`. Technische Universität München, 2009.

[36] J. Esparza, A. Gaiser, and S. Kiefer. „Computing Least Fixed Points of Probabilistic Systems of Polynomials“. In: *STACS*. 2010, pp. 359–370.

[37] J. Esparza, S. Kiefer, and M. Luttenberger. „Convergence Thresholds of Newton’s Method for Monotone Polynomial Equations“. In: *STACS*. 2008, pp. 289–300.

[38] J. Esparza, A. Kučera, and R. Mayr. „Model Checking Probabilistic Pushdown Automata“. In: *LICS 2004*. IEEE Computer Society, 2004, pp. 12–21.

[39] J. Esparza, A. Gaiser, and S. Kiefer. *Proving Termination of Probabilistic Programs Using Patterns*. Tech. rep. Available at `http://arxiv.org/abs/1204.2932`. 2012.

[40] J. Esparza, A. Gaiser, and S. Kiefer. „Proving Termination of Probabilistic Programs Using Patterns“. In: *CAV*. 2012.

[41] J. Esparza, A. Kučera, and R. Mayr. *Quantitative Analysis of Probabilistic Pushdown Automata: Expectations and Variances*. Tech. rep. FIMU-RS-2005-07. Masaryk University, 2005.

[42] K. Etessami and M. Yannakakis. „Recursive Markov chains, stochastic grammars, and monotone systems of non-linear equations“. In: *Journal of ACM* (2009).

[43] K. Etessami and M. Yannakakis. „Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations“. In: *Journal of the ACM* 56.1 (2009), pp. 1–66.

[44] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. „Impossibility of distributed consensus with one faulty process“. In: *Journal of ACM* 32.2 (Apr. 1985).

[45] N. Francez. *Fairness*. Texts and monographs in computer science. Springer, 1986.

[46] *GMP library*. URL: `http://gmplib.org`.

[47] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1993.

[48] B. S. Gulavani and S. K. Rajamani. „Counterexample Driven Refinement for Abstract Interpretation“. In: *TACAS*. 2006, pp. 474–488.

[49] B. S. Gulavani, S. Chakraborty, A. V. Nori, and S. K. Rajamani. „Automatically Refining Abstract Interpretations“. In: *TACAS*. 2008, pp. 443–458.

[50] B. S. Gulavani, S. Chakraborty, A. V. Nori, and S. K. Rajamani. „Refining abstract interpretations“. In: *Inf. Process. Lett.* 110.16 (July 2010), pp. 666–671.

[51] P. Haccou, P. Jagers, and V.A. Vatutin. *Branching Processes: Variation, Growth, and Extinction of Populations*. Cambridge studies in American literature and culture. Cambridge University Press, 2007.

[52] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. „PASS: Abstraction Refinement for Infinite Probabilistic Models“. In: *TACAS*. 2010, pp. 353–357.

[53] T. Han, J.-P. Katoen, and B. Damman. „Counterexample Generation in Probabilistic Model Checking“. In: *IEEE Trans. Software Eng.* 35.2 (2009), pp. 241–257.

[54] T. E. Harris. *The theory of branching processes*. Berlin: Springer, 1963.

[55] H. Hermanns, B. Wachter, and L. Zhang. „Probabilistic CEGAR“. In: *CAV*. 2008, pp. 162–175.

[56] G. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. First. Addison-Wesley Professional, 2003.

[57] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 2003.

[58] A. Hun, D. Dill, A. Drexler, and C. Yang. „Higher-level specification and verification with BDDs“. In: *CAV*. 1993, pp. 82–95.

[59] B. Jeannet and A. Miné. „Apron: A Library of Numerical Abstract Domains for Static Analysis“. In: *CAV*. 2009, pp. 661–667.

[60] M. Kattenbelt. „Automated Quantitative Software Verification“. PhD thesis. Oxford University, 2011.

[61] M. Kattenbelt. *QProver: Quantitative ANSI-C model checker*. URL: `http://www.prismmodelchecker.org/qprover/`.

[62] M. Kattenbelt, M. Z. Kwiatkowska, G. Norman, and D. Parker. „A game-based abstraction-refinement framework for Markov decision processes“. In: *Form. Methods Syst. Des.* 36 (3 2010), pp. 246–280.

[63] M. Kattenbelt, M. Z. Kwiatkowska, G. Norman, and D. Parker. „Abstraction Refinement for Probabilistic Software“. In: *VMCAI*. 2009, pp. 182–197.

[64] S. Kiefer, M. Luttenberger, and J. Esparza. „On the Convergence of Newton's Method for Monotone Systems of Polynomial Equations“. In: *STOC*. 2007, pp. 217–226.

[65] D. Kozen. „Semantics of probabilistic programs“. In: *Journal of Computer and System Sciences* 22 (1981), pp. 328–350.

[66] M. Kwiatkowska, G. Norman, and D. Parker. „PRISM 4.0: Verification of Probabilistic Real-time Systems“. In: *CAV*. 2011, pp. 585–591.

[67] D. Lehmann and S. Shelah. „Reasoning with time and chance“. In: *Automata, Languages and Programming*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1983, pp. 445–457.

[68] C. Limongelli and R. Pirastu. „Exact solution of linear equation systems over rational number by parallel p-adic arithmetic“. In: *CONPAR*. Springer, 1994.

[69] *lp_solve reference guide*. URL: `http://lpsolve.sourceforge.net/5.5/`.

[70] C. D. Manning and H. Schuetze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[71] *Maple*. URL: `http://www.maplesoft.com/`.

[72] A. McIver and C. Morgan. „Developing and Reasoning About Probabilistic Programs in pGCL“. In: *PSSE*. 2004, pp. 123–155.

[73] A. McIver, C. Morgan, and Thai Son Hoang. „Probabilistic termination in B“. In: *ZB2003*. Springer, 2003.

[74] A. Miné. „The octagon abstract domain“. In: *Higher-Order and Symbolic Computation* 19.1 (2006), pp. 31–100.

[75] A. Miné. „Weakly Relational Numerical Abstract Domains“. PhD thesis. École polytechnique, 2004.

[76] D. Monniaux. „Abstract Interpretation of Probabilistic Semantics“. In: *SAS*. 2000, pp. 322–339.

[77] D. Monniaux. „Abstract Interpretation of Programs as Markov Decision Processes“. In: *SAS*. 2003, pp. 237–254.

[78] D. Monniaux. „An Abstract Analysis of the Probabilistic Termination of Programs“. In: *SAS*. Springer, 2001, pp. 111–126.

[79] S. S. Muchnick. *Advanced Compiler Design and Implementation.* Morgan Kaufmann, 1997.

[80] T. Nakata. „On the Expected Time for Herman's Probabilistic Self-Stabilizing Algorithm“. In: *Theoretical Computer Science* 349.3 (2005), pp. 475 –483.

[81] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999.

[82] D. Parker. „Implementation of Symbolic Model Checking for Probabilistic Systems“. PhD thesis. University of Birmingham, 2002.

[83] A. Pnueli. „On the Extremely Fair Treatment of Probabilistic Algorithms“. In: *STOC*. 1983, pp. 278–290.

[84] A. Pnueli and L.D. Zuck. „Probabilistic verification“. In: *Inf. Comput.* 103 (1 1993), pp. 1–29.

[85] A. Podelski and A. Rybalchenko. „ARMC: The Logical Choice for Software Model Checking with Abstraction Refinement“. In: *PADL*. 2007, pp. 245–259.

[86] A. Podelski and A. Rybalchenko. „Transition Invariants“. In: *LICS*. 2004, pp. 32–41.

[87] A. Podelski and A. Rybalchenko. „Transition Invariants and Transition Predicate Abstraction for Program Termination“. In: *TACAS*. 2011, pp. 3–10.

[88] *PRISM model checker homepage.*
URL: http://www.prismmodelchecker.org/.

[89] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley-Interscience, 1994.

[90] *QSOpt_ex solver.*
URL: http://www2.isye.gatech.edu/~wcook/qsopt/ex/.

[91] J. S. Rosenthal. *A first look at rigorous probability theory.* 2. ed. Singapore [u.a.]: World Scientific, 2006. XVI, 219.

[92] A. Rybalchenko. „Temporal verification with transition invariants". PhD thesis. Universität des Saarlandes, 2005.

[93] R. Segala and N. A. Lynch. „Probabilistic Simulations for Probabilistic Processes". In: *Nordic Journal of Computing* 2.2 (1995), pp. 250–273.

[94] M. Sharir, A. Pnueli, and S. Hart. „Verification of Probabilistic Programs". In: *SIAM Journal on Computing* 13.2 (1984), pp. 292–314.

[95] R. Tarjan. „Depth-First Search and Linear Graph Algorithms". In: *SIAM Journal on Computing* 1.2 (1972), pp. 146–160.

[96] *The PRISM Language - Semantics.*
URL: http://www.prismmodelchecker.org/doc/semantics.pdf.

[97] M. Y. Vardi. „Automatic Verification of Probabilistic Concurrent Finite-State Programs". In: *FOCS.* 1985, pp. 327–338.

[98] B. Wachter. „Refined Probabilistic Abstraction". PhD thesis. Universität des Saarlandes, 2011.

[99] B. Wachter and L. Zhang. „Best Probabilistic Transformers“. In: *VMCAI*. 2010, pp. 362–379.

[100] B. Wachter, L. Zhang, and H. Hermanns. „Probabilistic Model Checking Modulo Theories“. In: *QEST*. 2007, pp. 129–140.

[101] G. Winskel. *The formal semantics of programming languages: an introduction*. Cambridge, MA, USA: MIT Press, 1993.

[102] D. Wojtczak and K. Etessami. „PReMo: An Analyzer for Probabilistic Recursive Models“. In: *TACAS*. 2007, pp. 66–71.