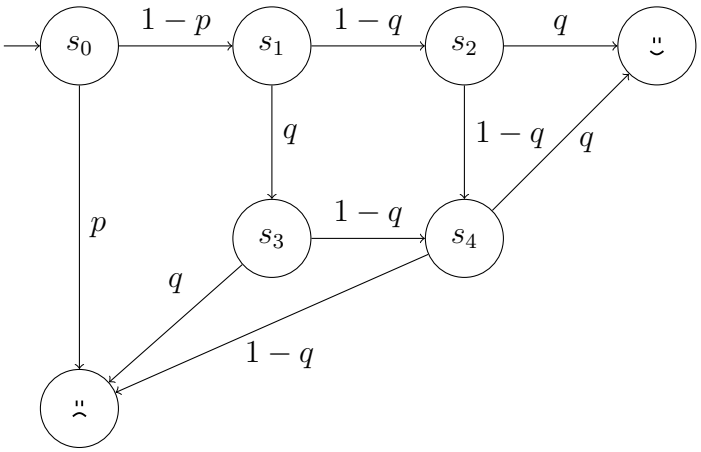
Monotonicity in Markov Chains

Jip Spel



UNIVERSITY OF TWENTE.

Formal Methods and Tools

Faculty of Electrical Engineering, Mathematics and Computer Science

University of Twente

In cooperation with:
Chair for Software Modeling and Verification
RWTH Aachen University

Supervisors:
Prof. Dr. Ir. Joost-Pieter Katoen
Prof. Dr. Marielle Stoelinga
Sebastian Junges M.Sc.

Location:
Enschede & Aachen

Time Frame:
November 2017– May 2018

If you weren't you, then we'd all be
A BIT LESS WE

— Piglet, Winnie the Pooh

Dedicated to and in loving memory of Judith Wijnhoven
1964 – 2017

Abstract

Markov chains (MCs) are an excellent formalism to capture the behaviour of systems that are governed by randomized behaviour. They are used in computer science, engineering, mathematics, and biology. MCs require fixed distributions, but often these probabilities are not precisely known.

Parametric MCs (pMCs) allow for changing specific sets of distributions in the MC. One way to find the values for parameters is parameter synthesis. Based on the pMC and the specification, the parameter values for which the system meets these requirements are needed to be calculated. We investigate the effect of changing the parameter values. In particular, we observe that parameters often have a monotone effect on the probability that a given system state is reached.

We want to exploit this monotone effect to improve the analysis on the behaviour of systems. To that end, we provide a formal framework to verify efficiently whether these systems are monotone. The framework consists of two layers: the foundation, and the top layer. In the foundation we define a set of monotone pMCs through the composition of predefined building blocks. In the top layer, we show that structures in high level descriptions of systems naturally map to the building blocks of the foundation.

If you don't know anything about computers,
just remember that they are machines that
do exactly what you tell them
but often surprise you in the result

— Richard Dawkins [1]

Acknowledgements

I would first like to thank my supervisors Sebastian, Joost-Pieter and Marielle for their time reading (and re-reading) my earlier versions of this master's thesis. Especially I thank Sebastian, the door to his office was always open whenever I had a question about my research or writing.

Next, I thank everyone at the Chair for Software Modeling and Verification at RWTH Aachen University for their support and all the social events, in particular the hours we spent playing kicker.

I would also like to acknowledge Sybe and Meike for their time to proof-read this thesis and their advice.

Finally, I must express my gratitude to my family, my boyfriend, Leonie, study friends, and Skeuvel for their support through the process of researching and writing this thesis. This accomplishment would not have been possible without you.

Contents

List of Tables

List of Figures

Listings

Acronyms and Notation

DTMC	discrete-time Markov chain
MC	Markov chain
MDP	Markov decision process
PA	Process algebra
pMC	parametric Markov chain
`pWhile`	Probabilistic While language
$\uparrow_p$	Monotone increasing in p
$\uparrow$	Monotone increasing in any parameter
$\downarrow_p$	Monotone decreasing in p
$\downarrow$	Monotone decreasing in any parameter
$\boldsymbol{\times}_p$	Not monotone in p
$\boldsymbol{\times}$	Not monotone in any parameter
$\mathbf{?}_p$	Do not know if monotone in p
?	Do not know if monotone in any parameter

Part I

Background

Chapter 1

Introduction

Probabilistic behaviour occurs in several kinds of systems. For instance systems containing randomized algorithms and communication protocols have probabilistic state changes. Moreover, the unreliable or unpredictable behaviour in computer networks is viewed as probabilistic behaviour. The occurrence of probabilistic behaviour has led to research on formal methods for the specification and verification of probabilistic systems. For instance, the bounded re-transmission protocol [4] (BRP) could be analyzed through formal methods. BRP is meant to transfer a file in a reliable manner, properties such as "the probability that the file is transferred correctly if messages are lost with a probability 0.05" could be analyzed through formal methods.

Markov chains (MCs) are often used to describe probabilistic models. If all transitions between states are probabilistic, the system can be modeled with discrete-time Markov chains (DTMCs).

***Example* 1.1**
We consider someone tossing two biased coins. Let the first coin toss head with probability $\frac{1}{4}$ and the second coin toss head with probability $\frac{1}{3}$. Figure 1.1 shows the associated DTMC, in which the person first tosses the first biased coin, and then tosses the second biased coin. At state s_2, tails is thrown twice, at state s_4 and s_5 heads is thrown once and tails is thrown once. At state s_6, heads is thrown twice. ∗

Parametric MCs (pMCs) are used when probabilities of a probabilistic model aren't known in advance, they generalize DTMCs by allowing parametric probabilities instead of fixed probabilities. E.g. in biochemical reaction networks [5] in which the rates of the reactions are either unknown or estimated with a possible measurement error. In these biochemical reaction networks, one still wants to show the robustness of the chemical network.

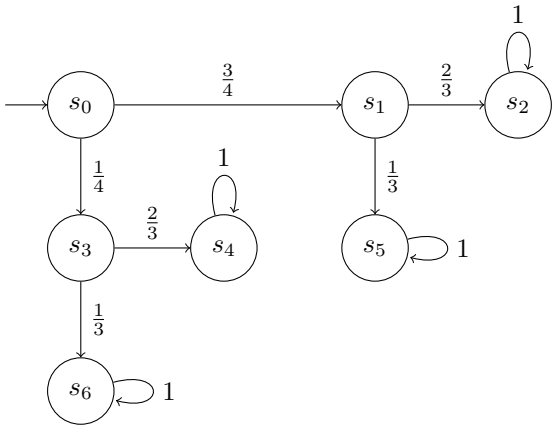


Figure 1.1: DTMC $\mathcal{D}$ of tossing two different biased coins

1.1 Motivation

For many properties, reachability analysis is the key procedure [6]. This reachability probability is often monotone in one or more parameters. When it is monotone increasing, this means that increasing the value of a parameter will always either increase the reachability probability or not change the reachability probability. An example of a *reachability problem* is: "find any valuation of the parameter values, for which the model full fills a reachability property."

Also in the Model Repair problem [7], pMCs are used. A general formulation for the Model Repair problem is: given a probabilistic system $\mathcal{M}$ and a probabilistic temporal logic formula ϕ such that $\mathcal{M}$ fails to satisfy ϕ, the Model Repair problem is to find a $\mathcal{M}'$ that satisfies ϕ and differs only from $\mathcal{M}$ by having transition probabilities being tuned with parameters. This tuning means that these probabilities can be changed, and transitions can be added or removed, but the state space remains the same. The tuning should be done such that the cost of modifying $\mathcal{M}$, such that it satisfies ϕ, should be minimized.

***Example* 1.2**
Recall the DTMC $\mathcal{D}$ in Figure 1.1 of a person tossing two biased coins. When the tossing distributions of the coins are not known in advance, we can replace the probabilities by parameters p and q. Figure 1.2 on the next page shows the pMC we obtain by replacing the probabilities through parameters. ∗

There are two main ways to find parameters meeting a given probabilistic temporal logic formula. The first way to find parameters is by *fitting* [8], whereby with the use of experiments the parameters are estimated. However, when e.g. a system component randomly fails, fitting isn't applicable, since too many experiments are required to get a proper estimate of the parameter values. Also

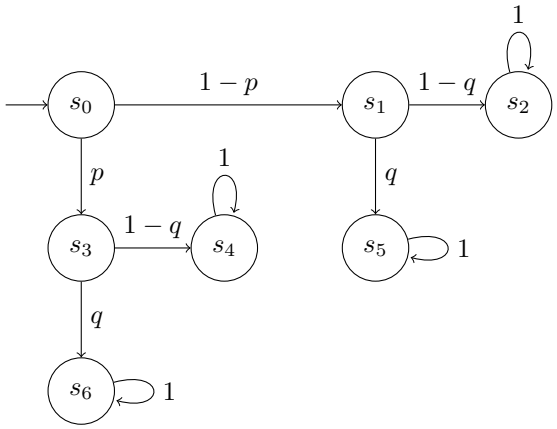


Figure 1.2: pMC $\mathcal{M}$ of tossing two different parametric coins

in the example of biochemical reaction networks, fitting is problematic, because of the included measurement errors. Also to answer questions "What is the maximal tolerable failure probability of component X while ensuring that the specifications hold?", fitting isn't suitable.

Another way to find the parameters is *parameter synthesis*. Based on the pMC and the requirements, the parameters for which the system meets these requirements are needed to be calculated. This is either done with the use state elimination to find the range of parameter values for which a system meets a given requirement or with parameter lifting. With parameter lifting, regions of parameter values are checked with the reachability property.

We observe that many pMCs are monotone in one or more of their parameters. This monotonicity can be used to solve reachability problems. Furthermore, we observe that for small pMCs, monotonicity can easily be determined. Our hypothesis is that a pMC which is known to be monotone in one or more of its parameters, can be constructed from simple, clearly monotone, pMCs.

1.1.1 *Problem statement*

We want to find monotonicity in pMCs without analyzing the whole rational function and apply this on structures in high level descriptions.

1.2 Contribution

The thesis gives a formal framework to deduce monotonicty of MCs. The framework consists of two layers. The first layer is the foundation and defines a set of monotonic pMCs as a composition of predefined building blocks. The second layer, the top layer, shows that common structures in high level description map

naturally to the building blocks in the foundation.

In the foundation, we show how to obtain monotonicity from the structure of pMCs. To this end, we provide the following:

- A mapping from a process in PA to a pMC.
- Theorems on monotonicity in pMCs.

The theorems in the foundation are proven by function calculus.

In the top layer, we show how the results of the foundation are used to obtain monotonicity from the structure of a `pWhile` program. We show the following results:

- A mapping from a `pWhile` program to a process in PA.
- Theorems on monotone structures in `pWhile`.

The theorems are proven using the mapping from `pWhile` program to a process, the mapping from a process to a pMC, and the theorems in the foundation.

We rewrite several case studies to be in `pWhile` and show how the obtained results are used to deduce monotonicity for these case studies.

1.3 Structure

First, Chapter 2 recaps important notions and fixed notation. Chapter 3, describes relevant work related to pMCs and parameter synthesis. In Chapter 4 we elaborate on the problem description and approach in this research. After this, in Part II, we describe the foundation on monotonic pMCs. Part III lifts the results to pMCs constructed from `pWhile`. Finally, we conclude this thesis in Part IV by giving a summary and proposing future work.

Chapter 2

Preliminaries

In this section we introduce the definitions and notations on Markov chains and Markov decision processes. Secondly, we introduce the theory of parameter lifting. Thirdly, we introduce some notations on rational functions and monotone functions. Finally, we introduce `pWhile` in which the case studies are written.

2.1 Markov Chains

A *discrete-time Markov chain* (DTMC) is a transition system in which transitions to successor states depend on probabilistic choices. Furthermore, the probability of moving from one state to another, only depends on the current state. This is known as the memoryless property. We describe DTMCs as a directed graph where the nodes of the graph are the different states. The transitions are described by a probability matrix and states are labelled with the atomic propositions which hold in that state.

Definition 2.1.1 (Discrete-time Markov Chain [9])
A *discrete-time Markov Chain* (DTMC) is a tuple $\mathcal{D} = (S, s_0, \mathcal{P}, AP, L)$ where

- S is a finite set of *states*
- $s_0 \in S$ is the *initial state*
- $\mathcal{P}\colon S \times S \mapsto [0,1]$ is a *probability matrix* such that for all $s \in S$: $\sum_{t \in S} \mathcal{P}(s,t) = 1$
- AP is a set of *atomic propositions*
- $L\colon S \mapsto 2^{AP}$ is a *labelling function* which gives the atomic propositions that hold in a state. $*$

***Example* 2.1**
Figure 2.1 shows a DTMC with states $\{s_0, s_1, s_2\}$. s_0 is the initial state and with probability $\frac{1}{4}$ the DTMC evolves to state s_1 and with $\frac{3}{4}$ the DTMC evolves to state s_2. ∗

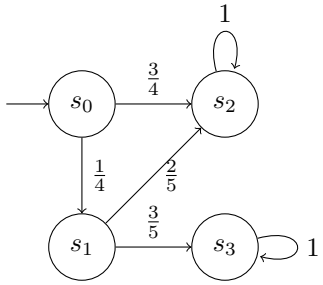


Figure 2.1: Example of a DTMC $\mathcal{D}$

A *finite path* in $\mathcal{D}$ is a sequence of states $\pi = s_0 s_1 \dots s_n \in S^n$. The probability of following a finite path π in a MC is the product of the probabilities of moving from one state to the next along π. A finite path is called *infeasbile* when the probability of following that path is 0.

Definition 2.1.2 (Probability of a path)
Let $\mathcal{D} = (S, s_I, \mathcal{P}, L)$ be a DTMC. Given *path* $\pi = s_0 s_1 \dots s_n \in S^n$, the probability of taking this path π is given by:

$$\mathcal{P}(\pi) = \mathcal{P}(s_0, s_1) \cdot \mathcal{P}(s_1, s_2) \cdot \ldots \cdot \mathcal{P}(s_{n-1}, s_n) \qquad *$$

***Example* 2.2**
Consider DTMC $\mathcal{D}$ from Figure 2.1. A possible finite path to follow in this DTMC is:

$$\pi = s_0 s_1 s_2$$

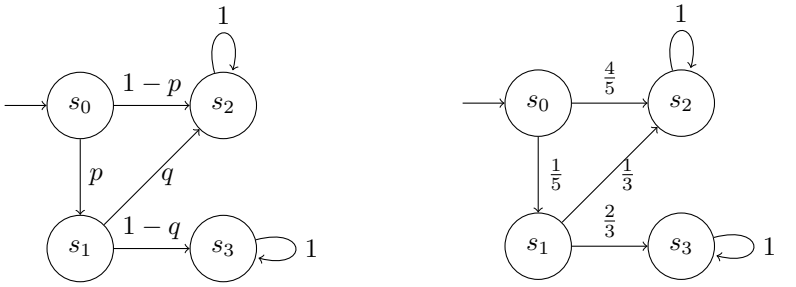
The probability of following this path is $\mathcal{P} = \frac{1}{4} \cdot \frac{2}{5} = \frac{1}{10}$. Note that e.g. $\pi = s_0 s_1 s_2 s_3$ is an infeasible path, as s_3 cannot be reached from s_2, the probability will be 0. ∗

In a *parametric (discrete-time) MC* (pMC) the entries of the probability matrix are given by rational functions. A rational function f over a set of parameters $V = \{x_1, \dots, x_n\}$ is a fraction $f(x_1, \dots, x_n) = \frac{g_1(x_1,\dots,x_n)}{g_2(x_1,\dots,x_n)}$ of two polynomials g_1 and g_2 over parameters V. Let $\mathbb{Q}_V = \{\frac{g_1}{g_2} | g_1, g_2 \in \mathbb{Z}[x_1, \dots, x_n] \wedge g_2 \neq 0\}$ denote the set of rational functions over V.

Definition 2.1.3 (Parametric Markov chain [10])
A *parametric* MC (pMC) is a tuple $\mathcal{D} = (S, s_0, \mathcal{P}, AP, L, V)$ where

- S, s_0, AP and L are defined as in Definition 2.1.1

- $V = \{x_1, x_2, \ldots, x_n\}$ is a *finite set of parameters* on domain $\mathbb{R}$
- $\mathcal{P}$ is the *probability matrix* $S \times S \to \mathbb{Q}_V$. ∗



(a) pMC $\mathcal{D}$ (b) induced pMC $\mathcal{D}_{\{p \to \frac{1}{5}, q \to \frac{1}{3}\}}$

Figure 2.2: Example of a pMC and an induced pMC

***Example* 2.3**
Figure 2.2a shows a pMC $\mathcal{D}$ with parameter x. The probability of moving from s_0 to s_1 is denoted by x, and the probability of moving from s_0 to s_2 by $1-x$. ∗

A *valuation* u is a partial function $u\colon V \to \mathbb{R}$. $Dom(u)$ denotes the domain of u. Valuation u is *total* when $Dom(u) = V$. Given a rational function f, a set of parameters $X \subseteq V$ and valuation u, $f[X/u]$ is the rational function obtained from f by substituting every occurrence of $x \in X \cap Dom(u)$ with $u(x)$. Applying a valuation on the rational function in a pMC results in the following definition of an induced pMC.

Definition 2.1.4 (Induced pMC [10])
Let $\mathcal{D} = (S, s_0, \mathcal{P}, AP, L, V)$ be a pMC. The pMC $\mathcal{D}_u$ induced by a valuation u is defined as $\mathcal{D}_u = (S, s_0, \mathcal{P}_u, AP, L, V_u)$ where

- $V_u = V \setminus Dom(u)$
- $\mathcal{P}_u\colon S \times S \to \mathbb{Q}_{V \setminus Dom(u)}$ is given by $\mathcal{P}_u(s, s') = \mathcal{P}(s, s')[Dom(u)/u]$ ∗

Definition 2.1.5 (Well-defined valuation)
A total valuation u is *well-defined* for pMC $\mathcal{D} = (S, s_0, \mathcal{P}, AP, L, V)$ if for the induced pMC $\mathcal{D}_u = (S_u, s_0, \mathcal{P}_u, AP, L)$ it holds that:

$$\mathcal{P}_u\colon S_u \times S_u \to [0,1] \text{ with for all } s \in S_u, \sum_{t \in S_u} \mathcal{P}(s,t) = 1 \quad *$$

Definition 2.1.6 (Graph preserving valuation)
A total valuation u for pMC $\mathcal{D}$ is called *graph preserving* if it is well-defined and it holds that:

$$\forall s, s' \in S. \mathcal{P}(s, s') \neq 0 \to \mathcal{P}(s, s')[u] > 0 \quad *$$

***Example* 2.4**
Recall pMC $\mathcal{D}$ from Figure 2.2a on the preceding page. For $\mathcal{D}$ valuation $u\colon \{p \to \frac{1}{5}, q \to \frac{1}{3}\}$, is a well-defined and graph preserving valuation. Figure 2.2b shows the MC induced by u. ∗

2.2 Markov Decision Process

A Markov decision process (MDP) is a transition system in which transitions to successor states are non-deterministic choices over probability distributions over states, so it can be viewed as a MC extended with non-deterministic transitions.

Definition 2.2.1 (Markov decision process [11])
A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, s_0, \mathcal{P}, AP, L, Act)$ where

- S, s_0, AP and L are defined as in Definition 2.1.1
- Act is a finite set of actions
- $\mathcal{P}\colon S \times Act \times S \mapsto \mathbb{Q} \cap [0,1]$ where for all states $s \in S$ and actions $\alpha \in Act$: $\sum_{s' \in S} \mathcal{P}(s, \alpha, s') \in \{0, 1\}$ and $Act(s) \neq \emptyset$. ($Act(s) = \{\alpha \in Act | \exists s' \in S. \mathcal{P}(s, \alpha, s') \neq 0\}$) ∗

An action α is *enabled* in state $s \in S$ when $\exists s' \in S. \mathcal{P}(s, \alpha, s') \neq 0$.

To reason about the probabilities in a MDP, we need a way to handle the non-determinism. Therefore, *schedulers* are introduced, a scheduler chooses in each state $s \in S$ an enabled action $\alpha \in Act(s)$, which is performed.

Definition 2.2.2 (Scheduler)
A *scheduler* for MDP $\mathcal{M} = (S, s_0, \mathcal{P}, L, Act)$ is a function $\sigma : S \to Act$ with $\sigma(s) \in Act(s)$ for all $s \in S$. ∗

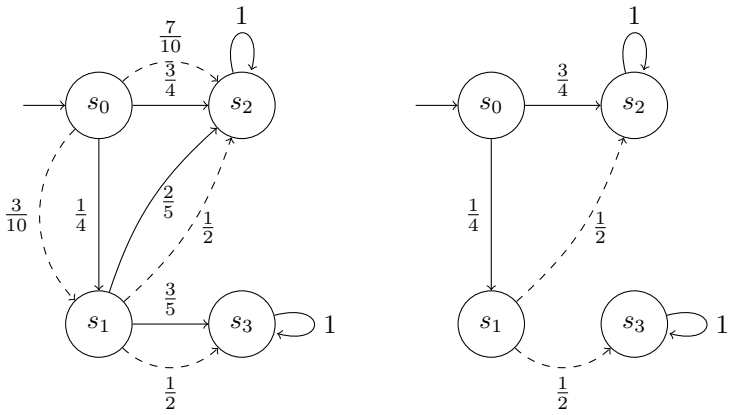
Imposing a scheduler σ on a MDP $\mathcal{M}$ resolves the non-determinism in $\mathcal{M}$ and yields MC $\mathcal{M}_\sigma$, in which the transition probabilities for a state s in $\mathcal{M}_\sigma$ equal those of taking action $\sigma(s)$ in state s in $\mathcal{M}$.

Definition 2.2.3 (MC of an MDP induced by a scheduler [6])
Let $\mathcal{M} = (S, s_0, \mathcal{P}, AP, L, Act)$ a MDP and σ a scheduler for $\mathcal{M}$. Markov chain $\mathcal{M}_\sigma$ induced by applying scheduler σ on MDP $\mathcal{M}$ is given by $\mathcal{M}_\sigma = (S, s_0, \mathcal{P}_\sigma, AP, L, Act)$ with

$$\mathcal{P}_\sigma(s, s') = \mathcal{P}(s, \sigma(s), s') \text{ for all } s, s' \in S$$ ∗

***Example* 2.5**
Figure 2.3a on the following page shows a MDP $\mathcal{M}$ in which at both state s_0 and s_1 a non-deterministic choice occurs. One can either take the dashed or the non-dashed transitions, which both have different transition probabilities. Taking

scheduler σ, which takes the non-dashed transitions at state s_0 and the dashed transitions at state s_1 yields into MC $\mathcal{M}_\sigma$ which van be found in Figure 2.3b ∗



(a) MDP $\mathcal{M}$ (b) MC $\mathcal{M}_\sigma$

Figure 2.3: Example of a MDP and MC of this MDP induced by a scheduler σ

2.3 Parameter Lifting

Parameter lifting is introduced by Quatmann et al. [11]. The main idea of parameter lifting is considering a specific *region* of the possible values of the parameters. For this region one attempts to determine whether the whole region can be considered safe or unsafe based on a given reachability property. A region is *well-defined* when all valuations in the region are well-defined. To determine whether a region is safe or unsafe, a sound over-approximation of the problem is used.

Definition 2.3.1 (Region [11])
Given a set of parameters $V = \{x_1, \ldots, x_n\}$ and *rational parameter bounds* $B(x_i) = \{b_1, b_2\}$. The parameter bounds induce a *parameter interval* $I(x_i) = [b_1, b_2]$ with $b_1 \leq b_2$. The set of valuations $\{u | \forall x_i \in V. u(x_i) \in I(x_i)\}$ is called a *region* (for V). ∗

***Example* 2.6**
Recall pMC $\mathcal{D}$ from Figure 2.2a on page 8. For this pMC $V = \{p, q\}$, and $p, q \in [0, 1]$. A possible parameter bound is given by:

$$B(p) = \{0.3, 0.7\} \text{ and } B(q) = \{0.2, 0.4\}$$

This yields intervals:

$$I(p) = [0.3, 0.7] \text{ and } I(q) = [0.2, 0.4]$$

Figure 2.4 on the next page shows region r imposed by these intervals on the parameters. ∗

First, all parameter dependencies in pMDP $\mathcal{M}$ are removed by *relaxation*. Sec-

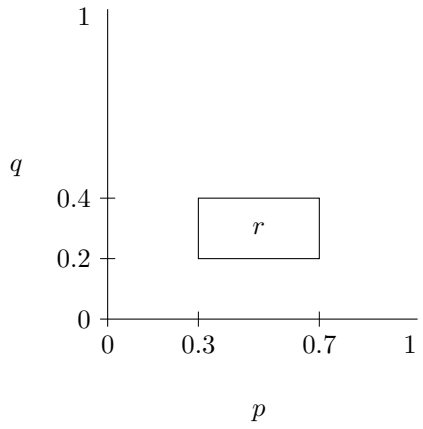


Figure 2.4: Region r of intervals $I(p) = [0.3, 0.7]$ and $I(q) = [0.2, 0.4]$

ondly, the parameter transitions are *substituted* by non-deterministic choices on the upper bound and lower bound of the valuations of the parameters in the region. This results in a MDP on which different schedulers are used to obtain the maximal and minimal reachability probabilities. With these reachability probabilities we claim that a region is either safe, unsafe or unknown.

Definition 2.3.2 (Relaxation [11])
The *relaxation* of pMC $\mathcal{D} = (S, s_0, \mathcal{P}, AP, L, V)$ is the pMC $rel(\mathcal{D}) = (S, s_0, \mathcal{P}', AP, L, rel_{\mathcal{D}}(V))$ with:

- $rel_{\mathcal{D}}(V) = \{x_i^s | x_i \in V, s \in S\}$
- $\mathcal{P}'(s, s') = \mathcal{P}(s, s')[x_1, \ldots, x_n / x_1^s, \ldots, x_n^s]$ *

Definition 2.3.3 (Substitution)
An MDP $sub_r(\mathcal{M}) = (S, s_0, \mathcal{P}_{sub}, AP, L, Act_{sub})$ is the *parameter-substitution* of a pMC $\mathcal{M} = (S, s_0, \mathcal{P}, AP, L, V)$ and a region r when:

- $Act_{sub} = \bigcup_{s \in S} \{v \colon V_s \to \mathbb{R} | v(x_i) \in B(x_i)\}$
- $\mathcal{P}_{sub}(s, v, s') = \begin{cases} \mathcal{P}(s, s')[v] & if\ v \in Act_{sub}(s), \\ 0 & otherwise. \end{cases}$ *

***Example* 2.7**
Figure 2.5a on the following page shows a pMC $\mathcal{D}$ with param p used both on the transitions from s_0 and the transitions from s_1. Figure 2.5b shows the relaxed pMC $rel(\mathcal{D})$ in which parameter p is split in two seperate parameters p^{s_0} and p^{s_1}. Figure 2.5c shows the parameter-substitution of $rel(\mathcal{D})$ with region r of intervals $I(p^{s_0}) = [0.3, 0.7]$ and $I(p^{s_1}) = [0.2, 0.4]$ *

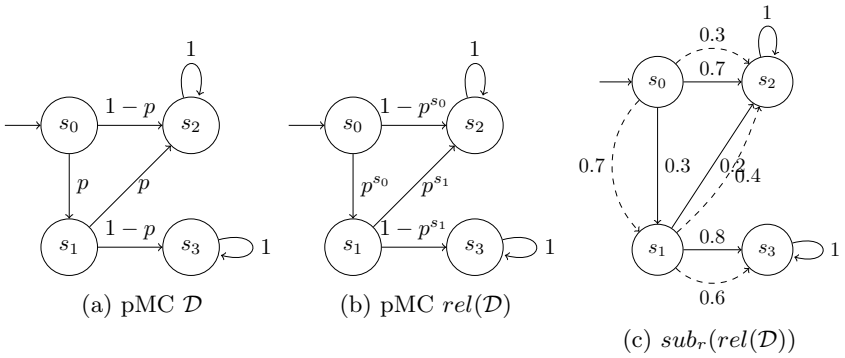


Figure 2.5: Example on relaxation and substitution

2.4 Functions

This section introduces the notations on functions used throughout this thesis. Given a function f, we denote by $f{\uparrow}$ that f is monotone increasing and with $f{\downarrow}$ that f is monotone decreasing, we denote by $f{\uparrow_p}$ that f is monotone increasing in p and with $f{\downarrow_p}$ that f is monotone decreasing in p. We assume all functions are continuous on the given domain.

All definitions of monotone decreasing functions are similar to the ones for monotone increasing, however, $\leq$ will be replaced with $\geq$ and vice versa.

Monotone functions

A univariate function $f\colon \mathbb{R} \to \mathbb{R}$ is called *monotone* if and only if it is either entirely *non-increasing* or entirely *non-decreasing*.

Definition 2.4.1 (Univariate monotone function)
Let $f\colon \mathbb{R} \to \mathbb{R}$: f is *monotone* when either $f{\uparrow}$ or $f{\downarrow}$. $*$

Definition 2.4.2 (Univariate monotone increasing function)
Let $f\colon \mathbb{R} \to \mathbb{R}$: $f{\uparrow} \leftrightarrow \forall x, x' \in \mathbb{R}.(x \leq x' \to f(x) \leq f(x'))$ $*$

From function theory we know that:

$$f{\uparrow} \leftrightarrow \forall x \in \mathbb{R}.f'(x) \geq 0$$

A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is called *monotone in parameter* x_i if and only if it is either entirely non-increasing or entirely non-decreasing in parameter x_i. A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is called monotone if and only if it is either entirely non-increasing or entirely non-decreasing in all of its parameters.

Definition 2.4.3 (Multivariate monotone function)
Let $f : \mathbb{R}^n \to \mathbb{R}$.

$$f \text{ is } \textit{monotone in parameter } x_i \in \mathbb{R} \text{ when either } f{\uparrow_{x_i}} \text{ or } {\downarrow_{x_i}}$$

$$f \text{ is } \textit{monotone} \text{ when either } f{\uparrow} \text{ or } f{\downarrow} \qquad *$$

From function theory we know that:

$$f{\uparrow_{x_i}} \leftrightarrow \forall \vec{x} \in \mathbb{R}^n . \frac{\partial}{\partial x_i} f(\vec{x}) \geq 0$$
$$f{\uparrow} \leftrightarrow \forall i \in [1 \ldots n] . f{\uparrow_{x_i}}$$

Critical Points and Turning Points

The *critical points* of univariate function f are the values within its the domain in which the derivative of f is 0. These points can either be *turning points* or *inflection points*. A *local extreme value* always occurs at a turning point. This is tested with the second derivative test.

Remark. The second derivative test is inconclusive when the second derivative is 0, in these cases a higher derivative test can be used.

Definition 2.4.4 (Critical points univariate function)
Let $f : \mathbb{R} \to \mathbb{R}$ be continuous.

$$v \in \mathbb{R} \text{ is a } \textit{critical point} \text{ for } f \leftrightarrow f'(v) = 0 \qquad *$$

Definition 2.4.5 (Second derivative test)
Given twice differentiable function f with critical point v. Then:

$$f''(v) < 0 \implies f \text{ has a local maximum at } v$$
$$f''(v) > 0 \implies f \text{ has a local minimum at } v$$
$$f''(v) = 0 \implies \text{ the test for } f \text{ is inconclusive} \qquad *$$

***Example* 2.8**
Figure 2.6 shows the critical points of functions f_1 and f_2. f_1 has a critical point at x_1. The second derivative test is for f_1 however, inconclusive. f_2 has a critical point at x_2 as well, for f_2 the second derivative test shows us that f_2 has a local maximum at x_1. The second derivative test for f_2 at x_2 is inconclusive at this point. *

The critical points of multivariate functions are the values on the domain in which the partial derivative is 0 for all parameters. Whether a critical point is a local maximum or minimum is tested with the second derivative test for n parameters. A multivariate function is monotone in a specific parameter if the partial derivative in that parameter is 0 independent of the values of the other parameters.

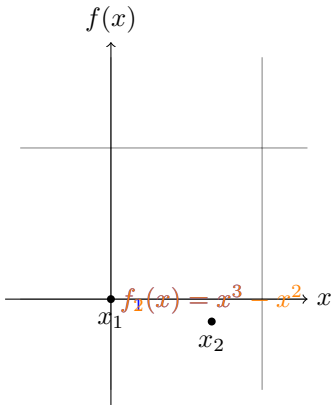


Figure 2.6: Critical points of univariate functions

Definition 2.4.6 (Critical points multivariate function)
Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuous.

$$\vec{v} \in \mathbb{R}^n \text{ is a } \textit{critical point} \text{ for } f \text{ in } v_i \iff {f_{v_i}}'(\vec{v}) = 0$$
$$\vec{v} \in \mathbb{R}^n \text{ is a } \textit{critical point} \text{ for } f \iff \forall i \in [1 \dots n].{f_{v_i}}'(\vec{v}) = 0 \qquad *$$

Definition 2.4.7 (Second derivative test multivariate)
Suppose that the second partial derivatives of $f : \mathbb{R}^n \to \mathbb{R}$ are continuous on a ball with center $\vec{x}$, with $\vec{x}$ a critical point of f.

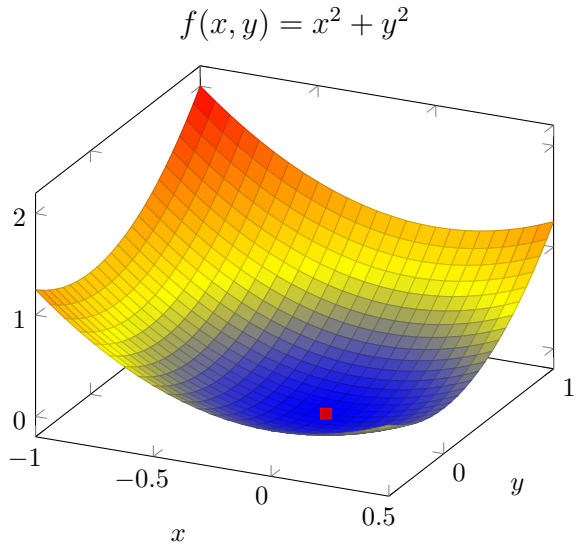
The *Hessian matrix* is the square matrix of the second-order partial derivatives. For Hessian matrix h index (i, j) is given by $h_{i,j} = \dfrac{\partial^2 f}{\partial x_i \partial x_j}$.

Let h denote the Hessian matrix of second partial derivatives and for each $k = 1, 2, \dots, n$ let D_k denote the determinant of the Hessian in the parameters $x_1, x_2, \dots, x_k$. Assume that $|H(\vec{x})| \neq 0$.

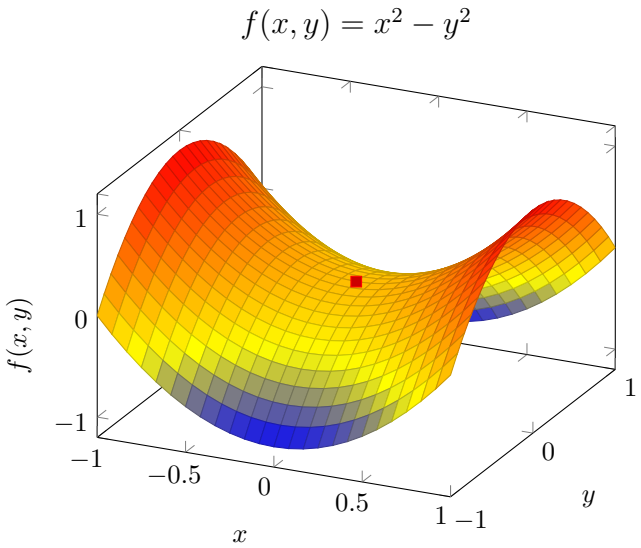
- If $D_k(c) > 0$ for all $k = 1, 2, \dots, n$ then f has a local minimum at $c\vec{x}$
- if $(-1)D_k(c) > 0$ for all $k = 1, 2, \dots, n$ then f has a local maximum at $\vec{x}$
- otherwise f has a saddle point at $\vec{x}$ *

A saddle point is a critical point of a multivariate function which is not an extremum.

***Example* 2.9**
Figure 2.7 shows two different multivariate functions. Figure 2.7a contains the plot of $f(x, y) = x^2 + y^2$, this function has a local minimum at $(0, 0)$. Figure 2.7b shows function $f(x, y) = x^2 - y^2$, this function has a saddle point at $(0, 0)$. *



(a) A multivariate function with a turning point



(b) A multivariate function with a saddle point

Figure 2.7: Multivariate functions

Monotonicity and turning points

For a univariate function f we know that given v is a local maximum, $f\uparrow$ closely to v and $f\downarrow$ from v until another turning point or boundaries of the interval. When v is a local minimum, it holds vice versa.

Definition 2.4.8 (Monotonicity around a local maximum of an univariate function)
Let $f : \mathbb{R} \to \mathbb{R}$ with turning point $v \in \mathbb{R}$ and points $v_1, v_2 \in \mathbb{R}$ such that $v_1 < v < v_2$ and f is monotone on $(v_1, v]$ and $[v, v_2)$.

$$v \text{ is a local maximum } \leftrightarrow f\uparrow \text{ on } (v_1, v] \text{ and } f\downarrow \text{ on } [v, v_2) \qquad *$$

For a multivariate function f we know that if $\vec{v}$ is a local maximum, $f\uparrow$ to $\vec{v}$ and $f\downarrow$ from $\vec{v}$ until another turning point or boundary of the interval. When $\vec{v}$ is a local minimum, it holds vice versa.

Definition 2.4.9 (Monotonicity around a local maximum of a multivariate function)
Let $f : \mathbb{R}^n \to \mathbb{R}$ with turning point $\vec{v} \in \mathbb{R}^n$ and points $\vec{v_1}, \vec{v_2} \in \mathbb{R}^n$ such that $\vec{v_1} < \vec{v} < \vec{v_2}$ for all variables in $\vec{v}$, $\vec{v_1}$ and $\vec{v_2}$ and f is monotone on $(\vec{v_1}, \vec{v}]$ and $[\vec{v}, \vec{v_2})$.

$$\vec{v} \text{ is a local maximum } \leftrightarrow f\uparrow \text{ on } (\vec{v_1}, \vec{v}] \text{ and } f\downarrow \text{ on } [\vec{v}, \vec{v_2}) \qquad *$$

Remark. When a multivariate function f has a critical point for parameter v_i, and this critical point is independent of the values of the other parameters, the theory on the univariate functions is applicable.

2.5 While Language

The While language is a simple programming language, used in the theoretical analysis of imperative programming language semantics. The While language consists of arithmetic expressions, Boolean expressions and program statements. Listing 2.1 gives the abstract syntax of the While language. In this listing, a is an arithmetic expression, n is a natural number, b is a Boolean expression, and S is the program statement.

```
a ::= n | x | a1 + a2 | a1 * a2 | a1 - a2
b ::= true | false | a1 = a2 | a1 ≤ a2 | ¬b | b1 ∧ b2
S ::= x := a | skip | S1;S2 | if b then S1 else S2 |
        while b do S
```

Listing 2.1: The While language

To describe the case studies, we use a probabilistic version of the While language (`pWhile`) as defined in Definition 2.5.1.

Definition 2.5.1 (Program in `pWhile`)
The syntax of a program `Prog` written in `pWhile` is denoted by:

---

a ::= n | x | a_1 + a_2 | a_1 * a_2 | a_1 - a_2
b ::= **true** | **false** | a_1 = a_2 | $a_1 \leq a_2$ | ¬b | b_1 ∧ b_2
S ::= x := a | skip | S_1;S_2 | **if** b **then** S_1 **else** S_2 |
while b **do** S | S_1 [f] S_2
Prog ::= S; **return;**

---

where

- a is an *arithmetic expression*
- n is defined on a bounded integer interval ($\mathbb{Z}_{\text{bound}}$)
- b is a *Boolean expression*
- S is a *program statement*
- x $\in Var_{\texttt{Prog}}$ with $Var_{\texttt{Prog}}$ the set of program variables of `Prog`
- $f \in \mathbb{Q}_V$ is a rational functions over the set of parameters V
- `Prog` is the program. ∗

***Example* 2.10**
Listing 2.2 shows a program `Prog` in which a biased coin is flipped. With probability p, `state` will be set to `0` and the program will terminate. With probability $1-p$, the program will set `state` to `1` and the while loop is entered. The while loop is executed at most three times. ∗

```
state := 0 [p] state := 1;
count := 1;
while count <= 3 ∧ state = 1 do
    state := 0 [p] state := 1;
    count := count + 1;
return;
```

Listing 2.2: Example of a program `Prog`

Boolean variable assignment We write `x := b` instead of

```
if b then
    x := 1;
else
    x := 0;
```

for readability of the programs.

Booleans In `pWhile` only {=, <=, ¬ and ∧} are defined. Clearly, {<, >=, >, ≠ and ∨} are syntactic sugar for combinations of {=, <=, ¬ and ∧}. In the remainder of this thesis we use in `pWhile` programs both {=, <=, ¬ and ∧} and {<, >=, >, ≠ and ∨}.

If then else For readability of the programs, we omit **`else`** `skip` in **`if`** `b` **`then`** `S` **`else`** `skip`.

Chapter 3

Related Work

In this chapter we discuss the work related to parameter synthesis and different tools developed for probabilistic model checking.

3.1 Parameter Synthesis

Daws [9] proposes a language-theoretical approach to determine the parametric reachability probability of DTMCs. First of all, Daws converts the parametric DTMC to a FSA. Secondly, a regular expression is computed with the help of state elimination. This expression is then evaluated into a closed-form function representing the reachability property. All parameters must be strictly positive since a transition between two states is only present in the derived FSA when it corresponds to a strictly positive probability.

Hahn et al. [10] investigate how Daws's idea can be turned into an efficient procedure. The bottleneck in Daws's idea is the growth of the regular expression relative to the number of states. To overcome this problem Hahn et al. intertwine the regular expression computation with its evaluation. In this way an efficient method is created which avoids a blow up in most practical cases. Although in the worst case the size of the final rational function is still $n^{\mathcal{O}(logn)}$, with n the number of states.

Quatmann et al. [11] introduce parameter lifting, which is described in Section 2.3.

Barnat et al. [12] apply LTL model checking procedures directly on a graph representing the dynamics of all possible valuations of parameters. Barnat et al. make use of parameterized Kripke structures to describe the models. The most significant difference between Barnat et al. [12] and the work described before is that the number of valuations of parameters in a parametric Kripke structure

is finite, whereas with the pMCs, the parameters can have any well-defined valuation in $\mathbb{R}$.

3.2 Tools for probabilistic model checking

There are several tools developed for probabilistic model checking. IscasMc [13] allows model checking on MCs and MDPs with LTL, PCTL and PCTL* specifications. MRMC [14] supports PCTL and CSL model checking. LTSmin [15] has developed into a model checker with multi-core algorithms for on-the-fly LTL checking with partial-order reduction, and multi-core symbolic checking for the modal μ-calculus, based on the multi-core decision diagram package Sylvan. Through SCOOP [16], probabilistic models can be checked with LTSmin. Storm [17] can model check both discrete- and continuous-time MCs and MDPs. Problem with these tools, and several other model checking tools, is that they don't allow for parametric model checking. HyTech analyses linear hybrid automaton [18] which is a mathematical model for dynamical systems whose behaviour exhibit both discrete and continuous changes. The tool can perform parameter analysis for linear hybrid automaton with temporal logic requirements. Since pMCs also contain non-linear functions, HyTech isn't useful to further investigate.

PRISM [19], PARAM [20], and PROPhESY [21] are all tools for analyzing parametric MCs. Both PRISM and PARAM work with state elimination and are based on the ideas of Daws [9] and Hahn et al. [10]. PROPhESY uses PRISMs language as input language for the pMCs. Together with the use of advanced gcd-computation on rational functions described by Jansen et al. [22], PROPhESY searches for safe and unsafe regions. This is done either with incremental parameter synthesis in a CEGAR manner or with parameter lifting.

To the best of our knowledge PRISM, PARAM and PROPhESY are the only tools used for analyzing parametric MCs. All of these tools assume that the structure of the pMCs won't change, i.e. the probability of taking a transitions won be zero and all valuations are well-defined.

Chapter 4

Problem Description and Approach

In this chapter, we describe a set of problems which can be solved by formal methods. Then, we elaborate on our approach in this research.

4.1 Problem Description

We consider *reachability probabilities* on MCs. Let $\Pr_s^{\mathcal{D}}(\Diamond T)$ denote the probability of eventually reaching a state $t \in T \subseteq S$ starting from a state $s \in S$, given DTMC $\mathcal{D}$. $\Pr^{\mathcal{D}}(\Diamond T)$ refers to this probability starting from initial state s_0.

Let $\varphi_{reach} = \mathbb{P}_{\geq\lambda}(\Diamond T)$ denote the *reachability property* asserting that eventually a state in T is reached with at least probability $\lambda \in (0, 1)$. We denote $\mathcal{D} \models \varphi_{reach}$ if and only if $\Pr^{\mathcal{D}}(\Diamond T) \geq \lambda$.

4.1.1 *Formalisation of the problems*

Given a pMC $\mathcal{D} = (S, s_0, \mathcal{P}, AP, L, V)$, set of target states $T \subseteq S$ and reachability property φ_{reach}. Let SAT be $\{u | u \text{ is graph-preserving and } \mathcal{D}_u \models \varphi_{reach}\}$. In Problems 3 and 4 we assume $SAT \neq \emptyset$. We define the problems as follows:

1. SAT $= \emptyset$
2. Check if region $r \subseteq$ SAT
3. Find any $u \in$ SAT
4. Find $u \in$ SAT such that $\Pr^{\mathcal{D}_u}(\Diamond T) = max_{u' \in \text{SAT}} \Pr^{\mathcal{D}_{u'}}(\Diamond T)$

4.1.2 *Observations*

Given a pMC $\mathcal{D} = (S, s_0, \mathcal{P}, AP, L, V)$ with reachability probability $\Pr^{\mathcal{D}}(\Diamond T)$ we first of all observe that when $\Pr^{\mathcal{D}}(\Diamond T)$ is monotone in a $v_i \in V$, solving the problems might become easier.

Assume $V = \{p, q\}$ with $p, q \in [0, 1]$. Given that $\Pr^{\mathcal{D}}(\Diamond T)$ is monotone increasing in p. Then the problems described above can be simplified by:

1. $\text{SAT} = \emptyset$
   To show $\text{SAT} = \emptyset$, it is sufficient to show that $\forall q \in [0, 1]$ and $p = 1$ $\Pr^{\mathcal{D}}(\Diamond T) = 0$.

2. Check if region $r \subseteq \text{SAT}$
   Let region r consist of intervals $I(p) = [0.3, 0.7]$ and $I(q) = [0.2, 0.4]$, this region is shown in Figure 4.1. Given the monotonicity in p, it is sufficient to show that all points on the line from $(0.3, 0.2)$ to $(0.3, 1.6)$, described by $p = 0.3$ and $q \in [0.2, 0.4]$, are in SAT. These are all the points on the thick line in Figure 4.1.

3. Find any $u \in \text{SAT}$
   When it is known that $\Pr^{\mathcal{D}}(\Diamond T)$ is monotone increasing in p, we take p as high as possible in u. When we know $\text{SAT} \neq \emptyset$, there will be an $u \in \text{SAT}$, with this highest value of p. So we are left finding a value for q in u, such that $u \in \text{SAT}$.

4. Find $u \in \text{SAT}$ such that $\Pr^{\mathcal{D}_u}(\Diamond T) = max_{u' \in \text{SAT}}$
   When it is known that $\Pr^{\mathcal{D}}(\Diamond T)$ is monotone increasing in p, we take p as high as possible in u. So we are left with finding the value of q in u, such that $\Pr^{\mathcal{D}_u}(\Diamond T) = max_{u' \in \text{SAT}}$.

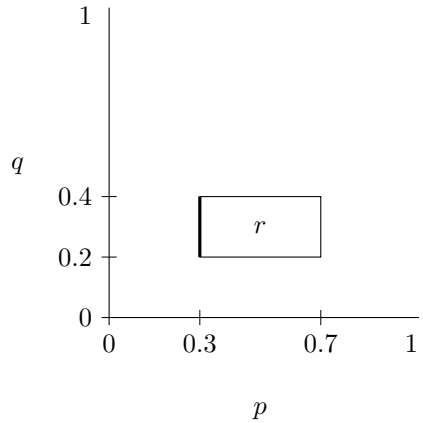


Figure 4.1: Region r of intervals $I(p) = [0.3, 0.7]$ and $I(q) = [0.2, 0.4]$

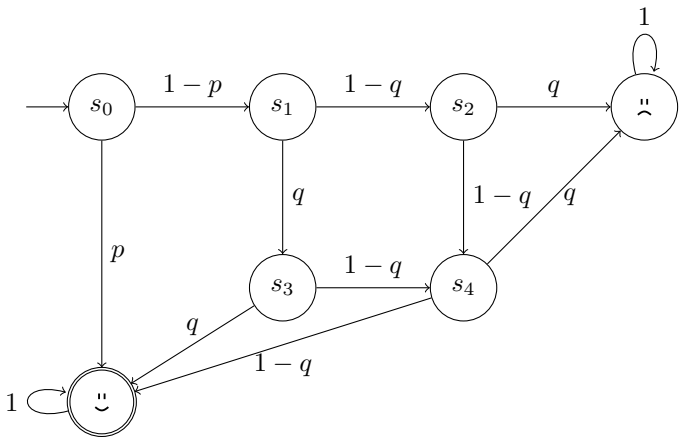


Figure 4.2: pMC $\mathcal{D}$ monotone in p

***Example* 4.1**
Figure 4.2 shows a pMC, with $T = \{$ ☺ $\}$. $\Pr^{\mathcal{D}}(\Diamond T)$ is monotone increasing in p. Consider Problem 2, a possible region to consider while using parameter lifting is build of $I(p) = [0.3, 0.7]$ and $I(q) = [0.2, 0.4]$. Exploiting the monotonicity in p, we only need to look at the lower bound on p, since increasing p will increase the reachability probability. Therefore, we don't need to split in p with parameter lifting. ∗

Secondly, we observe for small pMCs it is easy to determine if $\Pr_{\geq\lambda}(\Diamond T)$ is monotone in parameters of V (the clearly monotone pMCs in Figure 4.3). We can obtain the rational function and check, with basic function theory, if the rational function is monotone. However, this is not possible for larger pMCs, since the rational functions get too large.

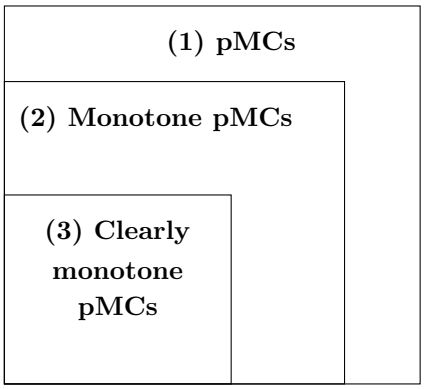


Figure 4.3: Subsets on pMCs

Our hypothesis is that pMCs which are known to be monotone in parameters of V can be constructed from simple, clearly monotone, pMCs. So we can find a subset of (2) in Figure 4.3, that includes (3). However, it may be hard to find this construction. The risk is that it can be complicated in either time or space.

4.1.3 *Problem statement*

We want to find monotonicity in pMCs without analyzing the whole rational function and apply this on structures in high level descriptions.

4.2 Approach

We provide a framework to deduce monotonicity of MCs. We describe in a compositional way, a subset of pMCs which are guaranteed to be monotone in one or more parameters. We provide proofs of the monotonicity of this subset of pMCs. As it might be hard to find a way to compose larger monotone pMCs from the small pMCs. We describe structures in `pWhile`, which map to a subset of pMCs which are monotone. We then show how we transform case studies which are monotone in one or more parameters into the monotone structures of `pWhile`.

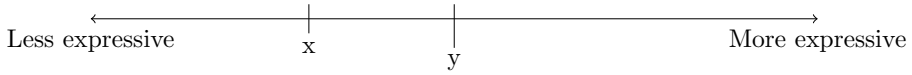


Figure 4.4: Structures of the While language

***Example* 4.2**
Figure 4.4 shows the difference in expressiveness of different structures of the probabilistic While language. Everything on the left of point y is monotone in one or more parameters. We try to find a point x, which is expressive enough to use in our case studies and is guaranteed to be monotone in one or more parameters. ∗

For this research the following questions must be answered.

1. Which subset of pMCs is monotone in one or more parameters?
2. Which structures in `pWhile` have an underlying pMC which is monotone in one or more parameters?
3. Can improvement in the run time of parameter lifting be achieved by exploiting the monotonicity?

We answer the first question in Part II. The second question is answered in Part III. As answering the first and second question took more time than expected, Question 3 is left as future work (Section 10.2).

Part II

Framework: Foundation

Chapter 5

Mapping process algebra to pMC

We use processes in PA to describe both pMCs and `pWhile` programs. We make use of processes, as this is a concise way to describe both pMCs and `pWhile` programs. Theorems on monotonicity in pMCs (Chapter 6), are defined on the process algebras of the pMCs. By mapping `pWhile` programs to process algebras (Chapter 7), we obtain theorems on the monotonicity in structures in `pWhile` (Chapter 8).

Process algebras are used for modelling and reasoning about the functional aspects of concurrent processes. Jonsson et al. [23] introduce a probabilistic process algebra (PCCS) for Markov decision processes. Our process algebra (PA) is an adaptation to PCCS and yields probabilistic deterministic processes, in which also Boolean expression may occur.

This chapter provides the introduction of the process algebra PA (Section 5.1) and the mapping from PA to pMCs (Section 5.2). Furthermore, it provides the PA notation of different building blocks of pMCs (Section 5.3), these blocks are used in Chapter 6.

Notation. For any function $f(p_1, \ldots, p_n)$, we write f instead. We use this notation throughout this thesis.

5.1 Process algebra

We give a formal definition of the syntax of a process term in PA (Definition 5.1.1). This allows us to define processes in PA as the pair of an initial process term (Definition 5.1.2) and the set of process terms.

Definition 5.1.1 (Syntax process term)
The syntax of a process term in PA is given by:
$\texttt{Proc} ::= ⏡ \mid ⏠ \mid \sum_{i=1}^{n} f_i.\ \texttt{Proc}_i \mid \texttt{Proc}_1\ \texttt{?}\ \texttt{Proc}_2\ \texttt{:}\ \texttt{Proc}_3$
where:

- ⏡ is a ***goal*** process term,
- ⏠ is a ***zero*** process term,
- $\sum_{i=1}^{n} f_i\,.\,\texttt{Proc}_i$ is a probabilistic sum over process terms with $n \geq 1$, $f_i \in \mathbb{Q}_V$ and $\sum_{i=1}^{n} f_i = 1$, and
- $\texttt{Proc}_1\ \texttt{?}\ \texttt{Proc}_2\ \texttt{:}\ \texttt{Proc}_3$ is the composition of three process terms. ∗

Definition 5.1.2 (Process)
A process in PA is a pair $\texttt{P} = (\texttt{Proc}_0, \{\texttt{Proc}_0, \ldots, \texttt{Proc}_n\})$ where

- $\texttt{Proc}_0$ is the initial process term of P, and
- $\texttt{Proc}_i$ is a process term for $0 \leq i \leq n$. ∗

A process term can either be a goal process term (⏡), a zero process term (⏠), a probabilistic sum of processes ($\sum_{i=1}^{n} f_i.\ \texttt{Proc}_i$) or the composition of different processes through an **`if then else`** statement ($\texttt{Proc}_1\texttt{?}\texttt{Proc}_2\texttt{:}\texttt{Proc}_3$). In the **`if then else`** statement, all occurrences of process term ⏡ in $\texttt{Proc}_1$ are replaced by $\texttt{Proc}_2$ and all occurrences of process term ⏠ in $\texttt{Proc}_1$ are replaced by $\texttt{Proc}_3$.

When the probabilistic sum consists of two elements, the binary sum, $\oplus_f$ is used. So $\texttt{Proc}_1\ \oplus_f\ \texttt{Proc}_2$ is shorthand for $f\,.\,\texttt{Proc}_1\ +\ (1-f)\,.\,\texttt{Proc}_2$. It is easy to see that every n-ary sum can be written as the composition of process terms with the binary sum.

***Example* 5.1**
An example process P is denoted in Listing 5.1. The initial process term of P is $\texttt{Proc}_0$. The set of process terms of P consists of 5 process terms, of which two goal process terms and one zero process term. From the initial process term, term $\texttt{Proc}_1$ is taken with probability p, and $\texttt{Proc}_2$ is taken with probability $1-p$. ∗

Initial process term: $\texttt{Proc}_0$

$\texttt{Proc}_0 = \texttt{Proc}_1 \oplus_p \texttt{Proc}_2$
$\texttt{Proc}_1 = \overset{\cdot\cdot}{\smile}$
$\texttt{Proc}_2 = \texttt{Proc}_3 \oplus_q \texttt{Proc}_4$
$\texttt{Proc}_3 = \overset{\cdot\cdot}{\frown}$
$\texttt{Proc}_4 = \overset{\cdot\cdot}{\smile}$

Listing 5.1: Example of a process P

5.2 From process to pMC

To describe the mapping from a process in PA to a pMC, we observe that every process term of a process describes a state. This state can be mapped directly to a state in the associated pMC. The mapping between a process in PA and a pMC is formalized as follows.

Definition 5.2.1 (Mapping from a process in PA to a pMC)
Let process $\texttt{P} = (\texttt{Proc}_0, \{\texttt{Proc}_0, \ldots, \texttt{Proc}_n\})$. The associated pMC $\mathcal{M}$ of P is given by $(S, s_0, \mathcal{P}, AP, L, V)$ where:

- the set of states S is $\{\texttt{Proc}_0, \ldots, \texttt{Proc}_n\}$,
- the initial state s_0 is $\texttt{Proc}_0$,
- the probability matrix $\mathcal{P}$ is obtained by the structural operational semantics in Table 5.1,
- the set of atomic propositions $AP = \{\overset{\cdot\cdot}{\smile}, \overset{\cdot\cdot}{\frown}\}$,
- the labelling function $L(s) = \begin{cases} \overset{\cdot\cdot}{\smile} & \text{if } s = \overset{\cdot\cdot}{\smile} \\ \overset{\cdot\cdot}{\frown} & \text{if } s = \overset{\cdot\cdot}{\frown} \end{cases}$, and
- the set of parameters V is the set of all parameters occurring in the process terms. ∗

Remark. Rule 3 of Table 5.1 could be replaced by $\dfrac{\texttt{Proc}_1 \xrightarrow{g} \texttt{X}}{\texttt{Proc}_1 \oplus_f \texttt{Proc}_2 \xrightarrow{f \cdot g} \texttt{X}}$.
A similar replacement for Rule 4 could be made. However, we choose not to replace these rules, as the replacement makes that a directly mapping between process terms and states is ommitted.

***Example* 5.2**
Figure 5.1 shows the pMC $\mathcal{M}$ associated with process P as described in Listing 5.1. Each process term refers to a single state in $\mathcal{M}$. ∗

$$\frac{}{\text{"}\smile\text{"} \xrightarrow{1} \text{"}\smile\text{"}}\ (1) \qquad \frac{}{\text{"}\frown\text{"} \xrightarrow{1} \text{"}\frown\text{"}}\ (2)$$

$$\frac{}{\texttt{Proc}_1 \oplus_f \texttt{Proc}_2 \xrightarrow{f} \texttt{Proc}_1}\ (3) \qquad \frac{}{\texttt{Proc}_1 \oplus_f \texttt{Proc}_2 \xrightarrow{1-f} \texttt{Proc}_2}\ (4)$$

$$\frac{\texttt{Proc}_1 \notin \{\text{"}\smile\text{"}, \text{"}\frown\text{"}\} \qquad \texttt{Proc}_1 \xrightarrow{f} \texttt{X}}{\texttt{Proc}_1 \ ?\ \texttt{Proc}_2 : \texttt{Proc}_3 \xrightarrow{f} \texttt{X}\ ?\ \texttt{Proc}_2 : \texttt{Proc}_3}\ (5)$$

$$\frac{\texttt{Proc}_1 = \text{"}\smile\text{"} \qquad \texttt{Proc}_2 \xrightarrow{f} \texttt{X}}{\texttt{Proc}_1 \ ?\ \texttt{Proc}_2 : \texttt{Proc}_3 \xrightarrow{f} \texttt{X}}\ (6) \qquad \frac{\texttt{Proc}_1 = \text{"}\frown\text{"} \qquad \texttt{Proc}_3 \xrightarrow{f} \texttt{X}}{\texttt{Proc}_1 \ ?\ \texttt{Proc}_2 : \texttt{Proc}_3 \xrightarrow{f} \texttt{X}}\ (7)$$

Table 5.1: Structural operational semantics of PA

Remark. In Figure 5.1 we differentiate between the two "⌣" states for readability. However, these two states are equal, just like process terms $\texttt{Proc}_1$ and $\texttt{Proc}_4$ in Listing 5.1.

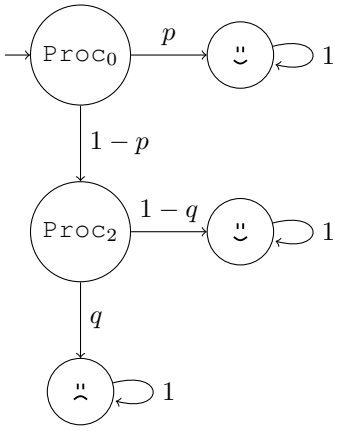


Figure 5.1: pMC $\mathcal{M}$ for process P in Listing 5.1

Above we define the translation from a process in PA to a pMC, with this definition we can define equivalence of processes. This definition is used in the proofs in Chapter 8.

Definition 5.2.2 (Equivalence of processes)
Let $\mathcal{M}_1$ be the pMC of process P, and $\mathcal{M}_2$ be the pMC of process Q.

Processes P and Q are equivalent $\iff$

$$\Pr^{\mathcal{M}_1}(\Diamond \dddot{\smile}) = \Pr^{\mathcal{M}_2}(\Diamond \dddot{\smile}) \text{ and } \Pr^{\mathcal{M}_1}(\Diamond \dddot{\frown}) = \Pr^{\mathcal{M}_2}(\Diamond \dddot{\frown})$$

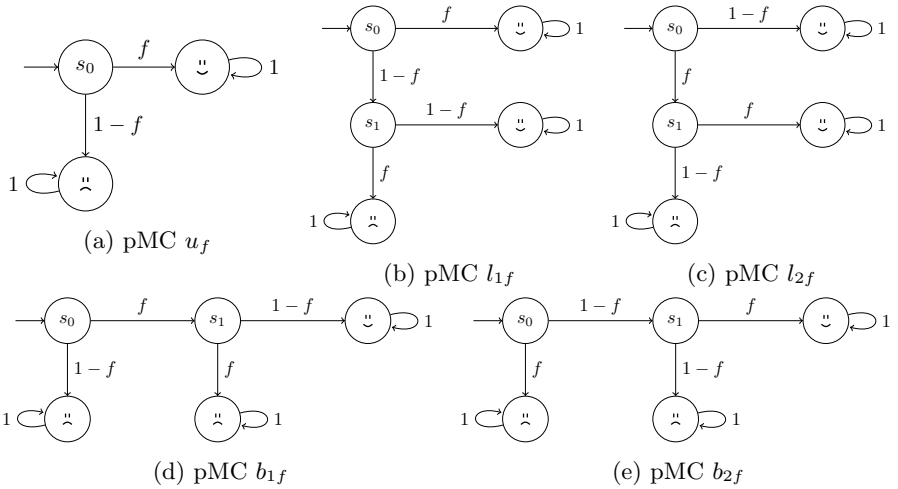
*

Remark. If every finite path in pMCs $\mathcal{M}_1$ and $\mathcal{M}_2$ can be extended such that it visits finitely many states, and almost surely reaches $\dddot{\smile}$ or $\dddot{\frown}$, then $\Pr(\Diamond \dddot{\smile}) + \Pr(\Diamond \dddot{\frown}) = 1$. To show equivalence between P and Q it is sufficient to show $\Pr^{\mathcal{M}_1}(\Diamond \dddot{\smile}) = \Pr^{\mathcal{M}_2}(\Diamond \dddot{\smile})$.

5.3 Building Blocks

In this section, we describe the different building blocks we use to describe monotonicity in pMCs in Chapter 6 on page 33. Figure 5.2 shows the pMCs of these building blocks. For readability, we place the $\dddot{\smile}$ states at the right side, and the $\dddot{\frown}$ states at the bottom of the figures. The subscript f denotes a function with $f \in \mathbb{Q}_V$ (see also Section 2.1). Table 5.2 on the next page shows the process of the building blocks.

Remark. In Table 5.2 we use an inline notation of processes. Clearly, it can be rewritten into process terms.



(a) pMC u_f (b) pMC l_{1f} (c) pMC l_{2f}

(d) pMC b_{1f} (e) pMC b_{2f}

Figure 5.2: The building blocks for constructing pMCs

Building block	**Process term**
u_f	$\ddot{\smile} \oplus_f \ddot{\frown}$
l_{1f}	$\ddot{\smile} \oplus_f (\ddot{\frown} \oplus_f \ddot{\smile})$
l_{2f}	$(\ddot{\smile} \oplus_f \ddot{\frown}) \oplus_f \ddot{\smile}$
b_{1f}	$(\ddot{\frown} \oplus_f \ddot{\smile}) \oplus_f \ddot{\frown}$
b_{2f}	$\ddot{\frown} \oplus_f (\ddot{\smile} \oplus_f \ddot{\frown})$

Table 5.2: Process terms for the building blocks of Figure 5.2

***Example* 5.3**
Figure 5.3 shows pMC $\mathcal{M}$ composed from u_f, l_{1f}, and b_{1f} through $\texttt{Proc}_1$ `?` $\texttt{Proc}_2$ `:` $\texttt{Proc}_3$.
Let $\texttt{Proc}_1 = u_f$, $\texttt{Proc}_2 = l_{1f}$, and $\texttt{Proc}_3 = b_{1f}$. Through the composition, the $\ddot{\frown}$ states of $\texttt{Proc}_1$ are replaced by $\texttt{Proc}_3$, and the $\ddot{\smile}$ states of $\texttt{Proc}_1$ are replaced by $\texttt{Proc}_2$. ∗

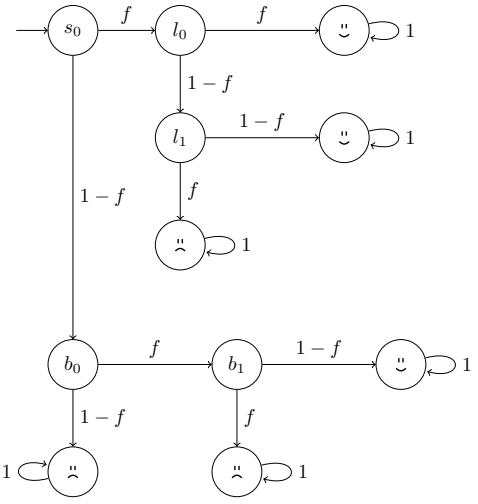


Figure 5.3: pMC $\mathcal{M}$ for the process term u_f ? l_{1f} : b_{1f}

5.3.1 *Cycles*

The building blocks only consider acyclic pMCs. However, cycles may occur in a pMC. In the case studies we consider, all cyclic transitions loop to the same state. Therefore, we introduce the notation of $\ddot{\smile}_g$ and $\ddot{\frown}_g$.

- $\overset{\frown}{\smile}_g$ P: Replace the $\smile$ states of process P with a process in which we go with probability g to $\smile$ and with probability $1-g$ to the initial process term of P.
- $\overset{\frown}{\frown}_g$ P: Replace the $\frown$ states of process P with a process in which we go with probability g to $\frown$ and with probability $1-g$ to the initial process term of P.

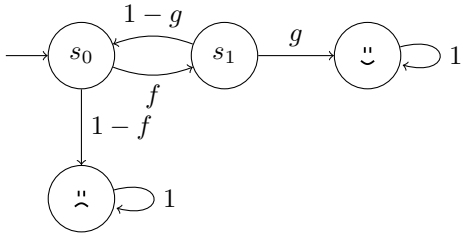
```
Proc0 = Proc1 ⊕f Proc2
Proc1 = Proc3 ⊕g Proc0
Proc2 = ⌢
Proc3 = ⌣
```

Listing 5.2: Process terms for $\overset{\frown}{\smile}_g\ u_f$

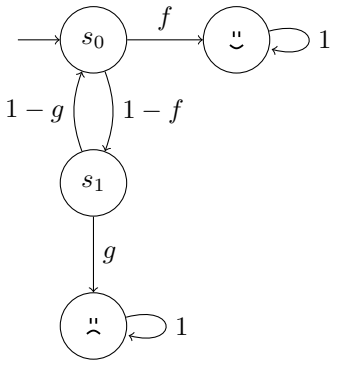
```
Proc0 = Proc3 ⊕f Proc1
Proc1 = Proc3 ⊕g Proc0
Proc2 = ⌢
Proc3 = ⌣
```

Listing 5.3: Process terms for $\overset{\frown}{\frown}_g\ u_f$

***Example* 5.4**
Figure 5.4a shows the pMC $\mathcal{M}_1$ of $\overset{\frown}{\smile}_g\ u_f$ and Figure 5.4b shows the pMC $\mathcal{M}_2$ of $\overset{\frown}{\frown}_g\ u_f$. Listing 5.2 shows the process terms for $\overset{\frown}{\smile}_g\ u_f$ and Listing 5.3 shows the process terms for $\overset{\frown}{\frown}_g\ u_f$. The processes of $\overset{\frown}{\smile}_g\ u_f$ and $\overset{\frown}{\frown}_g\ u_f$ both have initial process term $\texttt{Proc}_0$. ∗



(a) pMC $\mathcal{M}_1$ for process term $\overset{\frown}{\smile}_g\ u_f$



(b) pMC $\mathcal{M}_2$ for process term $\overset{\frown}{\frown}_g\ u_f$

Figure 5.4: Cycles in pMCs

Chapter 6

Monotonicity in pMCs

The previous chapter defines a process algebra (PA) for expressing pMCs. In this chapter, we consider monotonicity in the parameters of the probability functions of pMCs. A possible way to find monotonicity in the probability function of eventually reaching a ☺ state in a pMC, is to obtain a rational function symbolically expressing the reachability probability and check if this function is monotone for a given parameter. We discuss how monotonicity in pMCs can be obtained from the *structure* of the pMC without analyzing the rational function. This chapter presents the following results:

- For acyclic composition through $u_f ? u_g : u_h$, Theorems 6.3, 6.5, 6.6, 6.8 and 6.11 provide cases in which either monotonicity occurs, or the pMC is not monotone.
- For cyclic composition through $\circlearrowleft_{f_2} u_{f_1}$, Theorems 6.13 and 6.14 provide cases in which monotonicity occurs.

We start with the acyclic composition of pMCs through $u_f ? u_g : u_h$ (Section 6.1) and continue with cyclic composition trough $\circlearrowleft_{f_2} u_{f_1}$ (Section 6.2). In most theorems we assume $f{\uparrow_p}$ for any graph-preserving valuation and prove that $sol_{\mathcal{M}}{\uparrow_p}$. In a similar manner, theorems for $f{\downarrow_p}$ and claims on $sol_{\mathcal{M}}$ can be obtained. In Section 6.3, we show that for some sub-domains of graph-preserving valuations $f{\uparrow_p}$ might hold, and show that the theorems defined in Sections 6.1 and 6.2 can also be applied on these sub-domains.

Notation. Let $sol_{\mathcal{M}}$ be the function describing the probability of reaching state s in pMC $\mathcal{M}$ with $L(s) = ☺$. Let $\uparrow_p$ and $\downarrow_p$ be as defined before, $\boldsymbol{\times}_p$ denote that $sol_{\mathcal{M}}$ is not monotone in p and $\boldsymbol{?}_p$ denote that we do not know whether $sol_{\mathcal{M}}$ is monotone or not in p. Furthermore, we replace subscript 1 and 2 with i in building blocks b_1 and b_2 and building blocks l_1 and l_2 (Section 5.3). We

write f continuous for p_i, instead of for: any graph-preserving valuation u with $u = \{p_0, \ldots, p_n\}$, in which p_j is fixed for $i \neq j$, f is continuous.

Assumptions. In the theorems we only consider functions f for which:

- $\frac{\partial}{\partial p} f$ exists.
- Any total valuation $u \in Dom(f)$ is well-defined (Definition 2.1.5).

When we either consider composition with cyclic pMCs, or composition with not monotone structures, we additionally assume all total valuations $u \in Dom(f)$ are graph-preserving (Definition 2.1.6).

We observe that when a parameter p does not occur in function f, f is both monotonically increasing and decreasing in p; Lemma 6.1 formalizes this observation.

Lemma 6.1
If f is constant in parameter p, then $f{\uparrow_p}$ and $f{\downarrow_p}$. $*$

Proof of Lemma 6.1
$\frac{\partial}{\partial p} f = 0$ as p does not occur in f. Therefore, $\frac{\partial}{\partial p} f \geq 0$ and $\frac{\partial}{\partial p} f \leq 0$. $*$

Lemma 6.2
Let pMC $\mathcal{M} = u_f$, then: $f{\uparrow_p} \iff sol_{\mathcal{M}}{\uparrow_p}$ $*$

Proof of Lemma 6.2
We obtain the following equalities:

$$sol_{\mathcal{M}} = f$$
$$\frac{\partial}{\partial p} sol_{\mathcal{M}} = \frac{\partial}{\partial p} f$$

As the derivatives are equal, we obtain $\frac{\partial}{\partial p} sol_{\mathcal{M}} \geq 0$ if and only if $\frac{\partial}{\partial p} f \geq 0$. $*$

6.1 Acyclic composition

In this section we look at monotonicity of $sol_{\mathcal{M}}$ in parameter p through the composition of pMCs with $\mathcal{M} = u_f ? u_g : u_h$.

6.1.1 *General Composition*

Our first goal is to find monotonic structures while composing different pMCs. Theorem 6.3 provides two cases in which $sol_{\mathcal{M}}$ is monotone. Corollaries 6.4.a and 6.4.b follow from Theorem 6.3. Theorem 6.5 on page 36 provides cases in which $sol_{\mathcal{M}}$ is not monotone.

Theorem 6.3 (General composition)
Let pMC $\mathcal{M}$ be described by $u_f ? u_g : u_h$. Then:

1. $(f\uparrow_p$ and $g \geq h$ and $g\uparrow_p$ and $h\uparrow_p) \implies sol_{\mathcal{M}}\uparrow_p$
2. $(f\uparrow_p$ and $g \leq h$ and $g\downarrow_p$ and $h\downarrow_p) \implies sol_{\mathcal{M}}\downarrow_p$ *

Proof of Theorem 6.3
We provide the proof for Case 1; the proof of Case 2 is obtained in a similar manner. Let $sol_{\mathcal{M}}$ be the function describing the probability of eventually reaching a state s with $L(s) = \text{☺}$ in pMC $\mathcal{M} = u_f ? u_g : u_h$. So:

$$sol_{\mathcal{M}} = f \cdot g + (1 - f) \cdot h$$

By the definition of monotonicity (Definition 2.4.3 on page 13), to obtain $sol_{\mathcal{M}}\uparrow_p$ we need to show: $\frac{\partial}{\partial p} sol_{\mathcal{M}} \geq 0$. With the product rule we obtain:

$$\frac{\partial}{\partial p} sol_{\mathcal{M}} = \frac{\partial}{\partial p} f \cdot g - \frac{\partial}{\partial p} f \cdot h + \frac{\partial}{\partial p} g \cdot f + \frac{\partial}{\partial p} h \cdot (1 - f)$$

By assumption, $\frac{\partial}{\partial p} f$, $\frac{\partial}{\partial p} g$ and $\frac{\partial}{\partial p} h$ exist. Furthermore, $f\uparrow_p$, so $\frac{\partial}{\partial p} f \geq 0$.

To prove $\frac{\partial}{\partial p} sol_{\mathcal{M}} \geq 0$, it is sufficient to show that all of the following holds:

$$\frac{\partial}{\partial p} f \cdot g - \frac{\partial}{\partial p} f \cdot h \geq 0 \tag{6.1}$$

$$\frac{\partial}{\partial p} g \cdot f \geq 0 \tag{6.2}$$

$$\frac{\partial}{\partial p} h \cdot (1 - f) \geq 0 \tag{6.3}$$

We observe the following:

- By assumption $\frac{\partial}{\partial p} f \geq 0$, so $g \geq h$ implies (6.1) holds.
- $g\uparrow_p$ implies $\frac{\partial}{\partial p} g \geq 0$. Therefore, (6.2) holds.
- $h\uparrow_p$ implies $\frac{\partial}{\partial p} h \geq 0$. Therefore, (6.3) holds.

Thus we obtain $\frac{\partial}{\partial p} sol_{\mathcal{M}} \geq 0$. *

***Example* 6.1**
Let $\mathcal{M} = u_f?u_g : u_h$ as in Figure 6.1. Assume $f = p$, $g = p + (1-p) \cdot p$ and $h = \frac{1}{9} \cdot p$. Take e.g. $p \in [0,1]$ then $g \geq h$. Furthermore, $f{\uparrow_p}$, $g{\uparrow_p}$ and $h{\uparrow_p}$. Therefore, by Theorem 6.3 Case 1 we obtain $sol_{\mathcal{M}}{\uparrow_p}$. $*$

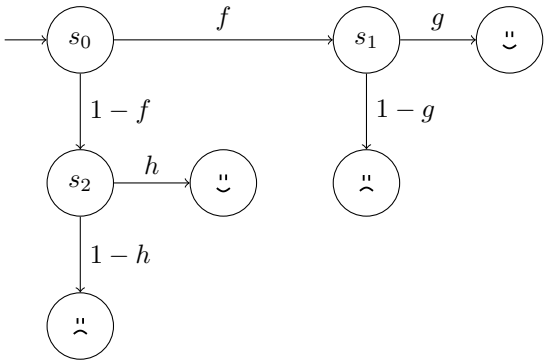


Figure 6.1: pMC $\mathcal{M} = u_f?u_g : u_h$

As $sol_{\smile} = 1$ and $sol_{\frown} = 0$, we obtain Corollaries 6.4.a and 6.4.b by combining Lemma 6.1 and Theorem 6.3.

Corollary 6.4

a. Let pMC $\mathcal{M} = u_f?u_g : \frown$. Then:

$$f{\uparrow_p} \text{ and } g{\uparrow_p} \Longrightarrow sol_{\mathcal{M}}{\uparrow_p}$$

b. Let pMC $\mathcal{M} = u_f?\smile : u_h$. Then:

$$f{\uparrow_p} \text{ and } h{\uparrow_p} \Longrightarrow sol_{\mathcal{M}}{\uparrow_p}$$ $*$

Theorem 6.5 (Not monotone)
Let pMC $\mathcal{M}$ be described by $u_f?u_g : u_h$. Then:

1. $\frac{\partial}{\partial p} f = 0$ and $\frac{\partial}{\partial p} g = 0$ and $h\text{✗}_p \Longrightarrow sol_{\mathcal{M}}\text{✗}_p$
2. $\frac{\partial}{\partial p} f = 0$ and $\frac{\partial}{\partial p} h = 0$ and $g\text{✗}_p \Longrightarrow sol_{\mathcal{M}}\text{✗}_p$
3. $g \neq h$ and $\frac{\partial}{\partial p} g = 0$ and $\frac{\partial}{\partial p} h = 0$ and $f\text{✗}_p \Longrightarrow sol_{\mathcal{M}}\text{✗}_p$ $*$

Proof of Theorem 6.5
We provide the proof for Case 1, the proofs of Cases 2 and 3 are obtained in a similar manner. For Case 1, we have the following functions:

$$\begin{aligned} sol_{\mathcal{M}} &= f \cdot g + (1-f) \cdot h \\ \frac{\partial}{\partial p} sol_{\mathcal{M}} &= \frac{\partial}{\partial p} f \cdot (g-h) + \frac{\partial}{\partial p} g \cdot f + \frac{\partial}{\partial p} h \cdot (1-f) \end{aligned}$$

As $\frac{\partial}{\partial p} f = 0$ and $\frac{\partial}{\partial p} g = 0$, we obtain:

$$\begin{aligned} \frac{\partial}{\partial p} sol_{\mathcal{M}} &= 0 \cdot (g-h) + 0 \cdot f + \frac{\partial}{\partial p} h \cdot (1-f) \\ &= \frac{\partial}{\partial p} h \cdot (1-f) \end{aligned}$$

As we only consider graph-preserving valuations $(1-f) > 0$. So, from $h \not\downarrow_p$, $sol_{\mathcal{M}} \not\downarrow_p$ follows. ∗

6.1.2 *Composition of building blocks with the same function*

In this section, we treat composition through $u_f ? u_g : u_h$, in which f, g and h might not be monotone. Let u_f, u_g, and u_h be one of the building blocks (see Section 5.3 on page 30), and assume that these building blocks are all build with the same function f. Theorem 6.6 provides the results for the following pMCs:

- $u_f ? u_g : u_h$ with $u_g, u_h \in \{\cup, \not\cup, u_f, l_{if}, b_{if}\}$,
- $l_{if} ? u_g : u_h$ with $u_g, u_h \in \{\cup, \not\cup, u_f\}$, and
- $b_{if} ? u_g : u_h$ with $u_g, u_h \in \{\cup, \not\cup, u_f\}$.

Remark. Tables 6.1b and 6.1c show symmetry, this is caused by the symmetry of l_{if} and b_{if}.

Theorem 6.6 (Composition of building blocks)
For any function f, with $f \uparrow_p$: $sol_{\mathcal{M}}$ in Table 6.1 is either 0, 1, $\uparrow_p$, $\downarrow_p$ or $?_p$. ∗

Remark. Theorem 6.6 is an extension to Theorem 6.3 on page 35. In Theorem 6.6 we do not require u_f, u_g and u_h to be monotone in all cases, where in Theorem 6.3 we require this monotonicity.

Proof of Theorem 6.6
We provide the proof for $u_f ? \cup : l_{if}$ in Table 6.1a. All other proofs are obtained in a similar manner.

$\mathbf{u_g}$ \ $\mathbf{u_h}$	$\Subset$	$\Supset$	u_f	l_{if}	b_{if}
$\Subset$	1	$\uparrow_p$	$\uparrow_p$	$?_p$	$\uparrow_p$
$\Supset$	$\downarrow_p$	0	$?_p$	$\downarrow_p$	$?_p$
u_f	$?_p$	$\uparrow_p$	$\uparrow_p$	$?_p$	$\uparrow_p$
l_{if}	$?_p$	$\uparrow_p$	$\uparrow_p$	$?_p$	$\uparrow_p$
b_{if}	$\downarrow_p$	$?_p$	$?_p$	$\downarrow_p$	$?_p$

(a) $\mathcal{M} = u_f ? u_g : u_h$

$\mathbf{u_g}$ \ $\mathbf{u_h}$	$\Subset$	$\Supset$	u_f
$\Subset$	1	$?_p$	$?_p$
$\Supset$	$?_p$	0	$?_p$
u_f	$\uparrow_p$	$\uparrow_p$	$\uparrow_p$

(b) $\mathcal{M} = l_{if} ? u_g : u_h$

$\mathbf{u_g}$ \ $\mathbf{u_h}$	$\Subset$	$\Supset$	u_f
$\Subset$	1	$?_p$	$\uparrow_p$
$\Supset$	$?_p$	0	$\uparrow_p$
u_f	$?_p$	$?_p$	$\uparrow_p$

(c) $\mathcal{M} = b_{if} ? u_g : u_h$

Table 6.1: Monotonicity of $sol_{\mathcal{M}}$ given $f{\uparrow_p}$

Note that:

- If for all states $s \in S$: $L(s) \neq \frown$, then $sol_{\mathcal{M}} = 1$.
- If for all states $s \in S$: $L(s) \neq \smile$, then $sol_{\mathcal{M}} = 0$.

For $\mathcal{M} = u_f ? \smile : l_{if}$, we obtain the following functions:

$$\begin{aligned} sol_{\mathcal{M}} &= f + (1-f) \cdot (f + (1-f)^2) \\ &= -f^3 + 2 \cdot f^2 - f + 1 \\ \frac{\partial}{\partial p} sol_{\mathcal{M}} &= -3 \cdot \frac{\partial}{\partial p} f \cdot f^2 + 2 \cdot \frac{\partial}{\partial p} f \cdot 2 \cdot f - \frac{\partial}{\partial p} f \\ &= \frac{\partial}{\partial p} f \cdot (-3 \cdot f^2 + 4 \cdot f - 1) \end{aligned}$$

As $\frac{\partial}{\partial p} f \geq 0$, we need to show either $(-3 \cdot f^2 + 4 \cdot f - 1) \geq 0$ or $(-3 \cdot f^2 + 4 \cdot f - 1) \leq 0$. Therefore, we need to know when:

$$-3 \cdot f^2 + 4 \cdot f - 1 = 0 \tag{6.4}$$

We obtain that Equation (6.4) holds when $f = \frac{-4 \pm \sqrt{4}}{-6}$. As $-3 \cdot f^2 + 4 \cdot f - 1$ is concave we have:

$$-3 \cdot f^2 + 4 \cdot f - 1 \begin{cases} > 0 & \text{if } \frac{1}{3} < f < 1 \\ = 0 & \text{if } f = \frac{1}{3} \text{ or } f = 1 \\ < 0 & \text{otherwise} \end{cases} \tag{6.5}$$

By assumption, $f \in [0, 1]$, however, it might be that $f \in [a, b]$ with $0 \leq a \leq b \leq 1$. Therefore, we can't claim anything about $\frac{\partial}{\partial p} sol_{\mathcal{M}}$, so, $sol_{\mathcal{M}} ?_p$. ∗

Although in general, $sol_{\mathcal{M}} ?_p$, there are special cases for which $sol_{\mathcal{M}} ✗_p$ can be derived. Recall the definition of turning points (Definitions 2.4.8 and 2.4.9 on page 16). We observe that for an entry in Table 6.1: $sol_{\mathcal{M}} ?_p$, then there might be a turning point x, in $sol_{\mathcal{M}}$, which is in the domain of f. If this is the case, then $sol_{\mathcal{M}} ✗_p$.

***Example* 6.2**
Let $\mathcal{M} = u_f ? \smile : l_{if}$, with $f = p$, we obtain the following functions:

$$\begin{aligned} sol_{\mathcal{M}} &= -p^3 + 2 \cdot p^2 - p + 1 \\ \frac{\partial}{\partial p} sol_{\mathcal{M}} &= -3 \cdot p^2 + 4 \cdot p - 1 \end{aligned}$$

By replacing f with p in Equation (6.5), we obtain that $p = \frac{1}{3}$ is a turning point of $sol_{\mathcal{M}}$. Therefore, $sol_{\mathcal{M}} ✗_p$. ∗

Theorem 6.7
Let f_{min} be the minimal value of f and f_{max} the maximal value of f. If $f_{min} \in [0, \frac{1}{3})$ and $f_{max} \in (\frac{2}{3}, 1]$, then all occurrences of $\mathbf{?}_p$ in Table 6.1 are replaced by $\mathbf{✗}_p$. ∗

Proof of Theorem 6.7
As f is a probability function, $f \in [0, 1]$. For all $sol_{\mathcal{M}}\mathbf{?}_p$ in Table 6.1, we obtain that $sol_{\mathcal{M}}$ has a turning point within $[\frac{1}{3}, \frac{2}{3}]$. This is proven in a similar way as for Theorem 6.6. So, for any function f with $f_{min} \in [0, \frac{1}{3})$ and $f_{max} \in (\frac{2}{3}, 1]$ we obtain $sol_{\mathcal{M}}\mathbf{✗}_p$. ∗

***Example* 6.3**
Recall pMC $\mathcal{M}$ in Figure 5.3 on page 31, where u_g and u_h are denoted by l_{if} and b_{if}, respectively. Although both $sol_{l_{if}}\mathbf{✗}_p$ and $sol_{b_{if}}\mathbf{✗}_p$, we obtain from Theorem 6.6 (Table 6.1a) that for $\mathcal{M} = u_f?l_{if} : b_{if}$:

$$sol_{\mathcal{M}}{\uparrow_p}$$ ∗

Notation. Let $(l_{if}?\smile)^m$ be shorthand for $\underbrace{(l_{if}?\smile : (l_{if}?\smile : \ldots))}_{m \text{ times } l_{if}}$ in which we nest l_{if} m times, and let $(b_{if} : \frown)^n$ be shorthand for $\underbrace{(b_{if}?(b_{if}? \ldots : \frown) : \frown)}_{n \text{ times } b_{if}}$ in which we nest b_{if} n times. If $m = 0$, then $(l_{if}?\smile)^m = \smile$, and if $n = 0$, then $(b_{if} : \frown)^n = \frown$.

Theorem 6.8
Let $\mathcal{M} = u_f?(l_{if}?\smile)^m : (b_{if} : \frown)^n$. For any $m, n \in \mathbb{N}$, it holds that:

$$f{\uparrow_p} \implies sol_{\mathcal{M}}{\uparrow_p}$$ ∗

***Example* 6.4**
Figure 6.2 shows the pMC $\mathcal{M} = u_f?(l_{if}?\smile)^m : (b_{if} : \frown)^n$. Assume $f = p$, from Theorem 6.8 it follows that $sol_{\mathcal{M}}{\uparrow_p}$. ∗

Lemmas 6.9 and 6.10 will be used to prove Theorem 6.8. Lemma 6.9 provides the case for $m \in \mathbb{N}$ and $n = 0$, where Lemma 6.10 provides the case for $n \in \mathbb{N}$ and $m = 0$.

Lemma 6.9
Let $\mathcal{M} = u_f?(l_{if}?\smile)^m : \frown$. For any $m \in \mathbb{N}$:

$$f{\uparrow_p} \implies sol_{\mathcal{M}}{\uparrow_p}$$ ∗

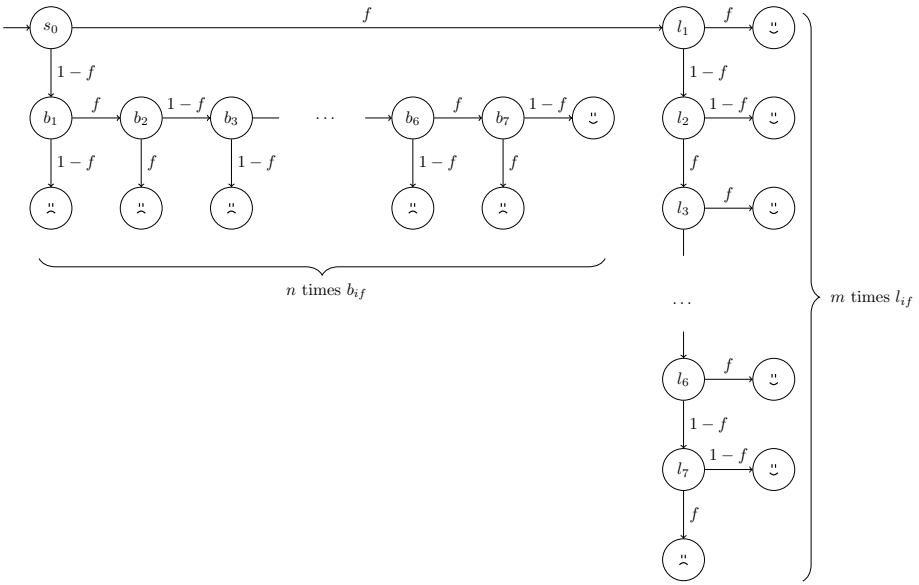


Figure 6.2: pMC $\mathcal{M} = u_f?(l_{if}?\frown\!\!\!\!\smile)^m : (b_{if} : \frown\!\!\!\!\frown)^n$

Lemma 6.10
Let $\mathcal{M} = u_f?\frown\!\!\!\!\smile : (b_{if} : \frown\!\!\!\!\frown)^n$. For any $n \in \mathbb{N}$:

$$f\uparrow_p \implies sol_{\mathcal{M}}\uparrow_p \qquad *$$

Proofsketch of Lemma 6.9
We make the following observations:

- $sol_{\mathcal{M}} = 1 - (1-f) - f \cdot (f \cdot (1-f))^m = f \cdot (1 - (f \cdot (1-f))^m)$
- $\dfrac{\partial}{\partial p} sol_{\mathcal{M}} \geq 0$ if and only if
$$(f \cdot (1-f))^{m-1} \cdot (f \cdot (1-f) + (1-2f) \cdot m \cdot f) \leq 1$$
- As $-1 \leq (1-2f) \leq 1$, there are two cases: $f \in [0, \frac{1}{2}]$ and $f \in (\frac{1}{2}, 1]$.

1. If $f \in (\frac{1}{2}, 1]$ then Lemma 6.9 holds. This proof is similar to the proof Theorem 6.3.
2. If $f \in [0, \frac{1}{2}]$ then Lemma 6.9 holds. This is proven by induction on m.

By Cases 1 and 2 we obtain that Lemma 6.9 holds for any $m \in \mathbb{N}$. A technical proof can be found in the Appendix. *

Proof of Lemma 6.10
The proof is obtained in a similar manner as the proof of Lemma 6.9. ∗

These results now bring us in a position to prove Theorem 6.8.

Proof of Theorem 6.8
The reachability probability in $\mathcal{M}$ is given by:

$$sol_{\mathcal{M}} = f \cdot (1 - (f \cdot (1-f))^m) + (1-f) \cdot (f \cdot (1-f))^n$$

We need to show $sol_{\mathcal{M}}{\uparrow_p}$, given $f{\uparrow_p}$ for any $m, n \in \mathbb{N}$. This is done by simultaneous induction on m and n.

- **Base case: $\mathbf{m = 0}$, $\mathbf{n = 0}$**
  This case maps to $u_f ?\ddot{\smile} : \ddot{\frown}$, $sol_{\mathcal{M}}{\uparrow_p}$ follows from Theorem 6.3.
- **Induction**
  Assume $sol_{\mathcal{M}}{\uparrow_p}$ for $m \leq k$, $n \leq l$. We need to show that for both $m = k+1$, $n = l$ and $m = k$, $n = l+1$, $sol_{\mathcal{M}}{\uparrow_p}$.
  - $m = k+1, n = l$
    Similar to the proof of Lemma 6.9 with induction hypothesis $sol_{\mathcal{M}}{\uparrow_p}$ until $m \in [0,k]$, $n \in [0,l]$.
  - $m = k, n = l+1$
    Similar to the proof of Lemma 6.10 with induction hypothesis $sol_{\mathcal{M}}{\uparrow_p}$ until $m \in [0,k]$, $n \in [0,l]$.

By induction on m and n we obtain: $f{\uparrow_p} \Longrightarrow sol_{\mathcal{M}}{\uparrow_p}$ for any $m, n \in \mathbb{N}$. ∗

6.1.3 *Composition of building blocks with different functions*

In the previous section, we introduced the composition of building blocks with the same function f. In this section we extend this to the composition of building blocks with different functions; we obtain Proposition 6.11 and Corollaries 6.12.a and 6.12.b.

Remark. The comparison with $\frac{1}{2}$ is introduced by the derivatives of $sol_{l_{ig}}$ and $sol_{b_{ih}}$, which need to be ≥ 0 to have monotonicity in $sol_{\mathcal{M}}$.

Theorem 6.11
Let pMC $\mathcal{M} = u_f ? l_{ig} : b_{ih}$, and let $f{\uparrow_p}$, $g{\uparrow_p}$ and $h{\uparrow_p}$. If

1. $((g + (1-g)^2) \geq h \cdot (1-h)$, and
2. $\frac{\partial}{\partial p} g = 0$ or $g \geq \frac{1}{2}$, and
3. $\frac{\partial}{\partial p} h = 0$ or $h \leq \frac{1}{2}$

then $sol_{\mathcal{M}}{\uparrow_p}$ ∗

Proof of Proposition 6.11
Let $sol_{\mathcal{M}}$ be the function describing the probability of eventually reaching a $\ddot{\smile}$ state in pMC $\mathcal{M} = u_f?l_{ig} : b_{ih}$. This function is given by:

$$sol_{\mathcal{M}} = f \cdot (g + (1-g)^2) + (1-f) \cdot h \cdot (1-h)$$

Its derivative is given by:

$$\begin{aligned}\frac{\partial}{\partial p} sol_{\mathcal{M}} = &\frac{\partial}{\partial p} f \cdot ((g + (1-g)^2) - h \cdot (1-h)) \\ &+ \frac{\partial}{\partial p} g \cdot f \cdot (2g-1)) \\ &+ \frac{\partial}{\partial p} h \cdot (1-f) \cdot (1-2h)\end{aligned}$$

We make the following observations:

- If $((g + (1-g)^2) \geq h \cdot (1-h)$ then

$$\frac{\partial}{\partial p} f \cdot ((g + (1-g)^2) - h \cdot (1-h)) \geq 0$$

- If either $\frac{\partial}{\partial p} g = 0$ or $g \geq \frac{1}{2}$ then

$$\frac{\partial}{\partial p} g \cdot f \cdot (2g-1)) \geq 0$$

- If either $\frac{\partial}{\partial p} h = 0$ or $h \leq \frac{1}{2}$ then

$$\frac{\partial}{\partial p} h \cdot (1-f) \cdot (1-2h) \geq 0$$

By these observations, we obtain $\frac{\partial}{\partial p} sol_{\mathcal{M}} \geq 0$. ∗

***Example* 6.5**
Figure 6.3 shows the pMC for $\mathcal{M} = u_f?l_{1g} : b_{1h}$. Assume $f = p$, $g = \frac{1}{2} \cdot p^2 + \frac{1}{2}$ and $h = \frac{1}{4} \cdot q$. By Proposition 6.11, it follows that $sol_{\mathcal{M}}\uparrow_p$. Furthermore, from Theorem 6.5 we obtain $sol_{\mathcal{M}}$ ✗$_q$, as $\frac{\partial}{\partial q} f = 0$, $\frac{\partial}{\partial q} sol_{l_{1g}} = 0$ and $sol_{b_{1h}}$ ✗$_q$. ∗

As $sol_{\ddot{\smile}} = 1$ and $sol_{\ddot{\frown}} = 0$, we obtain Corollaries 6.12.a and 6.12.b by combining Lemma 6.1 and Proposition 6.11.

Corollary 6.12
a. Let pMC $\mathcal{M} = u_f?l_{ig} : \frown, f\uparrow_p$ and $g\uparrow_p$:

$$\frac{\partial}{\partial p}g = 0 \text{ or } g \geq \frac{1}{2} \Longrightarrow sol_{\mathcal{M}}\uparrow_p$$

b. Let pMC $\mathcal{M}$ be given by $u_f?\smile : b_{ih}$, $f\uparrow_p$ and $g\uparrow_p$:

$$\frac{\partial}{\partial p}h = 0 \text{ or } h \leq \frac{1}{2} \Longrightarrow sol_{\mathcal{M}}\uparrow_p \qquad *$$

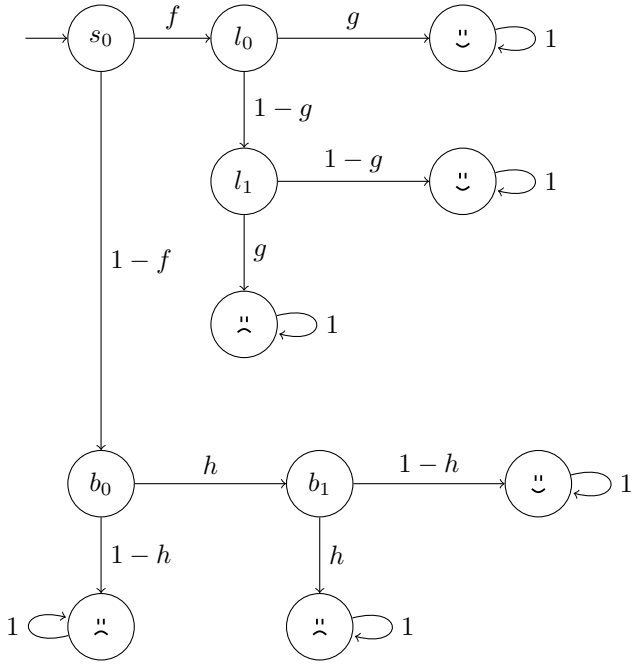


Figure 6.3: pMC $\mathcal{M} = u_f \ ? \ l_{1g} : b_{1h}$

6.2 Cyclic composition

In this section, we consider monotonicity in pMCs through the composition with $\overset{\curvearrowleft}{\smile}_g \, u_f$. Theorems 6.13 and 6.13 provide claims on the monotonicity for cyclic uni-variate pMCs. Note that $\overset{\curvearrowleft}{\frown}_g \, u_f = (\overset{\curvearrowleft}{\frown}_{1-g} \, u_{1-f}))?\frown : \smile$. Theorems for $\overset{\curvearrowleft}{\frown}_g \, u_f$ can be deduced by combining Theorems 6.6, 6.13 and 6.14.

***Example* 6.6**
Recall pMC $\mathcal{M}_1$ of $\overset{\curvearrowleft}{\smile}_g \, u_f$ in Figure 5.4a. We observe that if $f\uparrow_p$ and $g\uparrow_p$, then $sol_{\mathcal{M}_1}\uparrow_p$. Also if $f = 1 - g$ and $f\uparrow_p$, then $sol_{\mathcal{M}_1}\uparrow_p$. From this observation we obtain Theorem 6.13. *

Theorem 6.13
Let $\mathcal{M} = \circlearrowleft_{g}\ u_f$, then:

1. if $f\uparrow_p$ and $g\uparrow_p$ then $sol_{\mathcal{M}}\uparrow_p$
2. if $g = 1 - c \cdot f$, $c \in (0, 1]$ and $f \uparrow_p$ then $sol_{\mathcal{M}}\uparrow_p$ ∗

***Example* 6.7**
Recall $\mathcal{M} = \circlearrowleft_{g}\ u_f$ from Figure 5.4a. Assume $f = p$ and $g = 1 - p$, from Theorem 6.13 Case 2, we obtain $sol_{\mathcal{M}}\uparrow_p$. ∗

Theorem 6.14
Let $\mathcal{M} = \circlearrowleft_{1-g}\ (\circlearrowleft_{g}\ u_f)$, then:

1. if $f\uparrow_p$ and $g\uparrow_p$ and $g \leq \frac{1}{2}$ then $sol_{\mathcal{M}}\uparrow_p$
2. if $f\uparrow_p$ and $g\downarrow_p$ and $g \geq \frac{1}{2}$ then $sol_{\mathcal{M}}\uparrow_p$ ∗

Proof of Theorems 6.13 and 6.14
We provide the proof for Theorem 6.13 Case 1, the proofs of Theorem 6.13 Case 2 and Theorem 6.14 are obtained in a similar manner. Let $sol_{\mathcal{M}}$ denote the probability of reaching $\circlearrowleft$ in $\circlearrowleft_{g}\ u_f$, $f\uparrow_p$ and $g\uparrow_p$). We need to show $sol_{\mathcal{M}}\uparrow_p$.

$$\begin{aligned} sol_{\mathcal{M}} &= \sum_{i=0}^{\infty} f \cdot g \cdot (f \cdot (1-g))^i \\ &= f \cdot g \cdot \frac{1}{1 - f \cdot (1-g)} \\ \frac{\partial}{\partial p} sol_{\mathcal{M}} &= \frac{\frac{\partial}{\partial p} f \cdot g + \frac{\partial}{\partial p} g \cdot (1-f) \cdot f}{(1 - f(1-g))^2} \end{aligned}$$

We make the following observations:

- $(1 - f(1-g))^2 > 0$, and
- if both $f\uparrow_p$ and $g\uparrow_p$, then $\frac{\partial}{\partial p} f \cdot g + \frac{\partial}{\partial p} g \cdot (1-f) \cdot f \geq 0$

By these observations we obtain $\frac{\partial}{\partial p} sol_{\mathcal{M}} \geq 0$. ∗

6.3 Monotonicity and turning points

In Sections 6.1 and 6.2, we considered monotonicity on the whole function domain. Recall, that by calculating the critical points of a function f (Definition 2.4.4 and 2.4.6 on pages 13–14) we can obtain the turning points (Definitions 2.4.5 and 2.4.7 on pages 13–14). When we look at the turning points

of a function, we obtain sub-domains at which a function is monotone (Definitions 2.4.8 and 2.4.9 on page 16). Theorem 6.15 states that for any sub-domain of p, we can apply the Theorems introduced in this Chapter.

Theorem 6.15
For any subdomain $dom_{sub}(f) \subseteq dom(f)$, we can apply Lemmas 6.1 and 6.2 and Theorems 6.3-6.14. ∗

***Example* 6.8**
Let pMC $\mathcal{M} = u_f ? u_g : u_h$ with $f = p$ and $g = 1$ and $h = 1 - p + p^2$.

$$sol_{\mathcal{M}} = 1 - p + 2p^2 - p^3$$

A turning point of h is $p = \frac{1}{2}$. So if $p \geq \frac{1}{2}$ then $h{\uparrow_p}$. From Theorem 6.3 Case 1 and Theorem 6.15, we obtain:

$$\text{if } p \in [\frac{1}{2}, 1], \text{ then } sol_{\mathcal{M}}{\uparrow_p}$$ ∗

Proof of Theorem 6.15
The proofs are similar as those of the associated theorems, however, we now only consider a subdomain for f ∗

Part III

Framework: Application

Chapter 7

Mapping `pWhile` to process algebra

In Chapter 5, we introduced the process algebra PA and used this notion to describe pMCs. Recall that we make use of processes, as this is a concise way to describe both pMCs and `pWhile` programs. The subset of pMCs which are guaranteed to be monotone in some of their parameters is described in Chapter 6. As it might be hard to find a way to compose larger monotone pMCs from the simple pMCs, the aim of this chapter is to find substructures in `pWhile`, which map to the subset of pMCs which are monotone. To this end, we map `pWhile` to process algebras, and obtain theorems on the monotonicity in structures of `pWhile` (Chapter 8).

This chapter presents the following results:

- Restrictions on the `pWhile` programs, which are needed for the mapping from `pWhile` programs to a process in PA.
- The mapping from a `pWhile` program to a process in PA.
- Equivalence of statements `S1 [f] S2` and **`if`** `b` **`then`** `S1` **`else`** `S2`.

In this chapter, we describe how the process algebra PA is used to describe certain programs in `pWhile`. First of all, we introduce two restrictions on `pWhile` (Section 5.2). Secondly, we provide a mapping from `pWhile` programs to processes in PA (Section 7.2). For this mapping, we assume the `pWhile` programs to satisfy the restrictions. The mapping is used, together with the Theorems of Chapter 6, to prove the theorems of Chapter 8. Finally, we show equivalence of the probabilistic choice (`S1 [f] S2`) and the if statement (**`if`** `b` **`then`** `S1` **`else`** `S2`.) in Section 7.3. For this equivalence, we

require both the probabilistic choice, and the probability that b holds before executing the if statement, to have the same probability function f.

7.1 Restrictions on **pWhile**

To define the mapping from a pWhile program to a PA process, we define two restrictions on the programs:

1. The keyword state is reserved for a program variable to track whether the program is in a good (state = 1) or bad (state = 0) state. In the associated process the good and bad state are denoted by $\ddot\smile$ and $\ddot\frown$, respectively.
2. At the end of a program, state should either contain 0 or 1.

We observe that when a program Prog does not satisfy these restrictions, we can modify Prog to meet them. We need a Boolean expression b over the program variables, which states whether or not the program is in a $\ddot\smile$ state. The value of state will be 1 if b holds, and 0 otherwise. Let Prog ::= S; **return;**, the modified program Prog′ is denoted by: S; state := b; **return;**.

7.2 Probabilistic While language and PA

Recall the syntax of a program written in pWhile, as denoted in Definition 2.5.1 on page 17. In this section, we provide the mapping from pWhile programs and to processes in PA. We observe that every valuation of program variables together with the program statements (which still need to be executed) maps to a process term. This mapping is formalized as follows:

Definition 7.2.1 (Mapping from pWhile to PA)
Given a pWhile program Prog and the variable valuation η. Let the function

$$\texttt{Proc}\colon (Var_{\texttt{Prog}} \to \mathbb{Z}_{\text{bound}}) \times \texttt{pWhile} \to \text{PA}$$

for program Prog be defined as follows:

- $\texttt{Proc}(\eta, \textbf{return;}) = \begin{cases} \ddot\smile & \text{if } \eta \models [\![\text{state=1}]\!] \\ \ddot\frown & \text{if } \eta \models [\![\text{state=0}]\!] \end{cases}$
- $\texttt{Proc}(\eta, \texttt{x := } a\texttt{; P}) = 1.\texttt{Proc}(\eta\{x \leftarrow a\}, \texttt{P})$
- $\texttt{Proc}(\eta, \texttt{skip; P}) = 1.\texttt{Proc}(\eta, \texttt{P})$
- $\texttt{Proc}(\eta, \texttt{(}\textbf{if}\texttt{ b }\textbf{then}\texttt{ S}_1\texttt{ }\textbf{else}\texttt{ S}_2\texttt{); P}) = \begin{cases} 1.\texttt{Proc}(\eta, \texttt{ S}_1\texttt{; P}) & \text{if } \eta \models [\![\text{b}]\!] \\ 1.\texttt{Proc}(\eta, \texttt{ S}_2\texttt{; P}) & \text{otherwise} \end{cases}$

- $\texttt{Proc}(\eta,\ (\textbf{while}\ \texttt{b}\ \textbf{do}\ \texttt{S}_1);\ \texttt{P}) = \begin{cases} \texttt{1.Proc}(\eta,\ (\texttt{S}_1;\ \texttt{while b do S}_1);\ \texttt{P}) & \text{if } \eta \models [\![\texttt{b}]\!] \\ \texttt{1.Proc}(\eta,\ \texttt{P}) & \text{otherwise} \end{cases}$
- $\texttt{Proc}(\eta,\ (\texttt{S}_1\ \texttt{[f]}\ \texttt{S}_2);\ \texttt{P}) = \texttt{Proc}(\eta,\ \texttt{S}_1;\ \texttt{P})\ \oplus_f\ \texttt{Proc}(\eta,\ \texttt{S}_2;\ \texttt{P})$

With:

- η: $Var_{\texttt{Prog}} \to \mathbb{Z}_{\text{bound}} \cup \{\bot\}$ the valuation function of $Var_{\texttt{Prog}}$,
- $Var_{\texttt{Prog}}$ the set of program variables occurring in `Prog`, and
- $\bot$ the initial value of any variable in $Var_{\texttt{Prog}}$. ∗

Remark. For valuation of the program variables η: $Var_{\texttt{Prog}}$, let $\eta_{\text{pre}}\{S\}$ be the valuation of the program variables after executing program statement `S`, given current valuation η_{pre}. Furthermore, let for arithmetic expression a (Definition 2.5.1), $\eta(a)$ be the value of a after replacing all program variables by their value in η.

In the definition above, we inductively define the mapping from a `pWhile` program to a process term. The process of a `pWhile` program is formalized as follows:

Definition 7.2.2 (Process of a program)
The process `P` in PA of program `Prog` in `pWhile` is defined by:

$$\texttt{P(Prog)} = \texttt{Proc}\,(Var_{\texttt{Prog}} \to \bot\ ,\ \texttt{Prog}) \qquad *$$

***Example* 7.1**
Consider the `pWhile` program `Prog` in Listing 7.1. In each iteration of the while loop, a coin is flipped. This coin determines if we enter the while loop again, or we continue the rest of the program. When the while loop is executed an even number of times, the program will terminate with `state = 1`. Otherwise, it terminates with `state = 0`. The associated process `P` can be found in Listing 7.2 on the next page, in which `Pi-j` denotes the program statements contained by line-numbers i to j.

Remark. Some process terms are not defined for all variable valuations, as it might not be possible to enter a part of the code for this valuation. E.g. in Listing 7.1, the term `Proc({1,1}, P4-5; P3-6)` is not defined. With variable valuation {`state = 1, c = 1`}, it is not possible to enter the while loop. ∗

For processes we defined the equivalence in Definition 5.2.2. In a similar manner, we can define the equivalence of programs.

```
1 state := 0;
2 c := 0 [p] c := 1;
3 while c = 0 do
4     c := 0 [p] c := 1;
5     state := ¬ state;
6 return;
```

Listing 7.1: Example of a pWhile program Prog

```
Initial process term: Proc({⊥,⊥}, P1-9)

Proc({⊥,⊥}, P1-6) = 1.Proc({0,⊥}, P2-6)

Proc({0,⊥}, P2-9)
    = Proc({0,⊥}, c := 0; P3-6)
      ⊕p Proc({0,⊥}, c := 1; P3-6)

Proc({0,⊥}, c := 0; P3-6) = 1.Proc({0,0}, P3-6)
Proc({0,⊥}, c := 1; P3-6) = 1.Proc({0,1}, P3-6)

Proc({0,0}, P3-6) = 1.Proc({0,0}, P4-5; P3-6)
Proc({0,1}, P3-6) = 1.Proc({0,1}, P6)
Proc({1,0}, P3-6) = 1.Proc({1,0}, P4-5; P3-6)
Proc({1,1}, P3-6) = 1.Proc({1,1}, P6)

Proc({0,0}, P4-5; P3-6)
    = Proc({0,0}, c := 0; P5; P3-6)
      ⊕p Proc({0,0}, c := 1; P5; P3-6)
Proc({1,0}, P4-5; P3-6)
    = Proc({1,0}, c := 0; P5; P3-6)
      ⊕p Proc({1,0}, c := 1; P5; P3-6)

Proc({0,0}, c := 0; P5; P3-6) = 1.Proc({0,0}, P5; P3-)
Proc({0,0}, c := 1; P5; P3-6) = 1.Proc({0,1}, P5; P3-6)
Proc({1,0}, c := 0; P5; P3-6) = 1.Proc({1,0}, P5; P3-6)
Proc({1,0}, c := 1; P5; P3-6) = 1.Proc({1,1}, P5; P3-6)

Proc({0,0}, P5; P3-6) = 1.Proc({1,0}, P3-9)
Proc({0,1}, P5; P3-6) = 1.Proc({1,1}, P3-9)
Proc({1,0}, P5; P3-6) = 1.Proc({0,0}, P3-9)
Proc({1,1}, P5; P3-6) = 1.Proc({0,1}, P3-9)

Proc({0,1}, P9) = ⏜
Proc({1,1}, P9) = ⏝
```

Listing 7.2: Process P of program Prog in Listing 7.1

Definition 7.2.3 (Equivalence of programs)
Programs $\texttt{Prog}_1$ and $\texttt{Prog}_2$ are equivalent if and only if their processes $\texttt{P(Prog}_1\texttt{)}$ and $\texttt{P(Prog}_2\texttt{)}$ are equivalent (Definition 5.2.2). *

So far, we have shown how to obtain the process of a program. When proving theorems for conditions for monotonicity of `pWhile` programs, we need to know the process of a program statement `S`. Therefore, we need to know the goal, denoted by Boolean expression `b`, we want to achieve after executing `S`. When `b` holds after executing `S` we set `state` to 1. Otherwise, we set `state` to 0. We formalize this in Definition 7.2.4.

Definition 7.2.4 (Process of a program statement)
The process of program statement `S`, with Boolean expression `b` over `S`, is given by the process of the program `Prog` in Listing 7.3.

```
S;
state := ¬b;
return;
```

Listing 7.3: Program `Prog` obtained from program statement `S` and Boolean expression `b`.

*

7.2.1 *Reducing the process of a program*

Process `P` of a `pWhile` program can often be rewritten to obtain less process terms. A process `Q` that is equivalent to `P`, and `Q` has less process terms than `P` is called a *reduced process* of `P`. In this section, we provide an example in which we obtain a reduced process $\texttt{P}_{\texttt{red}}$.

***Example* 7.2**
Process `P` of Listing 7.2 can be reduced to process $\texttt{P}_{\texttt{red}}$ in Listing 7.5. This can be done in the following manner:

1. *Replace* all occurrences of $\texttt{Proc}_1 \texttt{ = 1.Proc}_2$ and $\texttt{Proc}_2 \texttt{ = Proc}_3$ by $\texttt{Proc}_1 \texttt{ = Proc}_3$.
2. *Remove* all superfluous process terms.
3. *Merge* similar process terms.

Remark. The reducement does not necessarily lead to an unique minimal program, as this is not the goal of the reducement. Furthermore, we observe that one could also reduce the pMC instead of the process. However, we choose to reduce the process, as we use the processes to proof Theorems in Chapter 8.

```
Proc({⊥,⊥}, P1-9)
    = Proc({1,⊥}, c := 0; P3-9)
      ⊕p Proc({1,⊥}, c := 1; P3-9)

Proc({1,⊥}, c := 0; P3-9)
    = Proc({1,0}, c := 0; P5-8; P3-9)
      ⊕p Proc({1,0}, c := 1; P5-8; P3-9)

Proc({1,⊥}, c := 1; P3-9) = ⁀
Proc({0,0}, P3-9)
        = Proc({0,0}, c := 0; P5-8; P3-9)
      ⊕p Proc({0,0}, c := 1; P5-8; P3-9)

Proc({1,0}, P3-9)
        = Proc({1,0}, c := 0; P5-8; P3-9)
      ⊕p Proc({1,0}, c := 1; P5-8; P3-9

Proc({0,0}, c := 0; P5-8; P3-9)
        = Proc({1,0}, c := 0; P5-8; P3-9)
      ⊕p Proc({1,0}, c := 1; P5-8; P3-9

Proc({0,0}, c := 1; P5-8; P3-9) = ⁀
Proc({1,0}, c := 0; P5-8; P3-9)
        = Proc({0,0}, c := 0; P5-8; P3-9)
      ⊕p Proc({0,0}, c := 1; P5-8; P3-9)

Proc({1,0}, c := 1; P5-8; P3-9) =  ‿
```

Listing 7.4: Process terms of P in Listing 7.2 after steps replace and remove

Initial process term: $\mathtt{Proc}_0$

$\mathtt{Proc}_0 = \mathtt{Proc}_1 \oplus_p \mathtt{Proc}_2$
$\mathtt{Proc}_1 = \mathtt{Proc}_0 \oplus_p \mathtt{Proc}_4$
$\mathtt{Proc}_2 = \frown$
$\mathtt{Proc}_4 = \smile$

Listing 7.5: Process $\mathtt{P}_{\mathrm{red}}$ of process P in Listing 7.2

In the first step we can replace in P:

```
Proc(\{⊥,⊥\}, P1-9) = 1.Proc(\{1,⊥\}, P2-9)
```

by:

```
Proc({⊥,⊥}, P1-9)
    = Proc({1,⊥}, c := 0; P3-9)
      ⊕p Proc({1,⊥}, c := 1; P3-9)
```

By the replacement, `Proc({1,⊥}, P2-9)` has become superfluous and can be removed from P.

After applying the first two steps on all process terms in P, we obtain the process in Listing 7.4. In this Listing,

`Proc({1, ⊥ }, c := 1; P3-9)` and
`Proc({0,0}, c := 1; P5-8; P3-9)`

refer to the same process $\frown$. Therefore, they can be merged. By merging all similar processes terms, and renaming them, we obtain $\mathtt{P}_{\mathrm{red}}$ as in Listing 7.5. This reduced process consists of four different process terms. ∗

7.2.2 *Examples on loops*

In this part we provide the pWhile programs associated with processes $\circlearrowleft_g$ P and $\circlearrowright_g$ P. Let S be a program statement of which the process in PA is equivalent to process P.

***Example* 7.3**
Let pWhile program Prog be the program in Listing 7.6. Assume S is given by: `state := 1 [f] state := 0`.

The process P(Prog), is denoted in Listing 7.7. In the same manner as in Section 7.2.1, P can be reduced. We observe that the reduced process of P(Prog) is equivalent to the process in Listing 5.2 on page 32.

```
1 S;
2 c := 0 [g] c := 1;
3 while state = 1 ∧ c = 1 do
4     S;
5     c := 0 [g] c := 1;
6 return;
```

Listing 7.6: Program of which the process is equivalent to process $\circlearrowleft_g$ P(S).

Remark. In Listing 7.7 we use $*$ as a wild-card for either 0, 1, or $\perp$.

In a similar manner the process for the pWhile program in Listing 7.8, with S given by state := 1 [f] state := 0, can be obtained, this process is equivalent to the one of Listing 5.3 on page 32.

Remark. Note that the program in Listing 7.6 with Prog as defined above, is equivalent to Zeroconf [24] with MAX=1. $*$

```
Initial process term: Proc({⊥,⊥}, P1-6)

Proc({⊥,⊥}, P1-6)
    = Proc({⊥,⊥}, state := 1; P2-6)
          ⊕_f Proc({⊥,⊥}, state := 0; P2-6)

Proc({⊥,⊥}, state := 0; P2-6) = 1.Proc({0,⊥}, P2-6)
Proc({⊥,⊥}, state := 1; P2-6) = 1.Proc({1,⊥}, P2-6)

Proc({*,⊥}, P2-6)
        = Proc({*,⊥}, c := 0; P3-6)
      ⊕_g Proc({*,⊥}, c := 1; P3-6)

Proc({*,⊥}, c := 0; P3-6) = 1.Proc({*,0}, P3-6)
Proc({*,⊥}, c := 1; P3-6) = 1.Proc({*,1}, P3-6)

Proc({0,0}, P3-6) = 1.Proc({0,0}, P6)
Proc({0,1}, P3-6) = 1.Proc({0,1}, P6)
Proc({1,0}, P3-6) = 1.Proc({1,0}, P6)
Proc({1,1}, P3-6) = 1.Proc({1,1}, P4-5; P3-6)

Proc({1,1}, P4-5; P3-6)
    = Proc({1,1}, state := 0; P5; P3-6)
      ⊕_f Proc({1,1}, state := 1; P5; P3-6)

Proc({1,1}, state := 0; P5; P3-6) = 1.Proc({0,1}, P5; P3-6)
Proc({1,1}, state := 1; P5; P3-6) = 1.Proc({1,1}, P5; P3-6)

Proc({*,1}, P5; P3-6)
    = Proc({*,1}, c := 0; P3-6)
      ⊕_g Proc({*,1}, c := 1; P3-6)

Proc({*,1}, c := 0; P3-6) = 1.Proc({*,0}, P3-6)
Proc({*,1}, c := 1; P3-6) = 1.Proc({*,1}, P3-6)

Proc({0,*}, P6) = ⌢
Proc({1,*}, P6) = ⌣
```

Listing 7.7: Process of the pWhile program of $\overset{\curvearrowleft}{\smile}_g\ u_f$

```
1 S;
2 c := 0 [g] c := 1;
3 while state = 0 ∧ c = 0 do
4     S;
5     c := 0 [g] c := 1;
6 return;
```

Listing 7.8: Program of which the process is equivalent to process $\overset{\curvearrowleft}{\frown}_g$ P(S).

7.3 Equivalence probabilistic choice and if statement

We observe that program statements

- S_1 [f] S_2, and
- **if** b **then** S_1 **else** S_2

are equivalent given that $\Pr_{\mathtt{b}} = f$. We use this equivalence in Chapter 8.

***Example* 7.4**
Let S_1 and S_2 be the program statements in Listing 7.9 and 7.10, respectively. Let Boolean expression c given by state = 1. For S_1, it is clear that:

$$\Pr\left(\eta^{S_1}_{\text{post}} \models [\![\mathtt{c}]\!]\right) = f$$

Let Boolean expression d be given by a = 0. For S_2, we observe that:

$$\Pr\left(\eta^{S_2}_{\text{post}} \models [\![\mathtt{d}]\!]\right) = f$$

So, $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{c}]\!]\right) = \Pr\left(\eta^{S_2}_{\text{post}} \models [\![\mathtt{d}]\!]\right)$.
Furthermore, $\Pr\left(\eta^{S_2}_{\text{post}} \models [\![\mathtt{c}]\!]\right) = \Pr\left(\eta^{S_2}_{\text{post}} \models [\![\mathtt{d}]\!]\right)$. Therefore, we obtain that for goal c given by state = 1 these program statements are equivalent (Definition 7.2.3).

Remark. $\Pr\left(\eta^{S_i}_{\text{post}} \models [\![\mathtt{b}]\!]\right)$ is the probability that after executing S_i, Boolean statement b holds. More information on the notation $\Pr\left(\eta^{i}_{\text{post}} \models [\![\mathtt{b}]\!]\right)$ can be found in Chapter 8. ∗

```
state := 1 [f] state := 0;
```

Listing 7.9: Probabilistic choice with $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{state=1}]\!]\right) = f$

```
a := 0 [f] a := 1;
state := a = 0;
```

Listing 7.10: If then else with $\Pr\left(\eta_{\text{post}} \models [\![\texttt{state=1}]\!]\right) = f$

Lemma 7.1
Let $\texttt{S}_3$ and $\texttt{S}_4$ be program statements, and let:

- program statement $\texttt{S}_1$ be given by $\texttt{S}_3$ `[f]` $\texttt{S}_4$,
- program statement $\texttt{S}_2$ be given by **`if`** `b` **`then`** $\texttt{S}_3$ **`else`** $\texttt{S}_4$, and
- $f = \Pr_{\texttt{b}}^{\texttt{S}_2}(\eta)$ for variable valuation η.

Then $\texttt{S}_1$ and $\texttt{S}_2$ are equivalent. ∗

Proof of Lemma 7.1
We obtain that by Definition 7.2.4 with goal `b` both the process of $\texttt{S}_1$ and $\texttt{S}_2$ are equivalent to u_f. By Definition 7.2.3 we obtain that $\texttt{S}_1$ and $\texttt{S}_2$ are equivalent. ∗

Chapter 8

Monotonic `pWhile` programs

In Chapter 6, we discusses how monotonicity in pMCs can be concluded from the structure of the pMC. As it may be hard to find a way to compose larger monotone pMCs from simple pMCs, our aim is to be able to identify monotonic pMC structures at a higher level of abstraction. To that end, this chapter presents results enabling to decide (non-) monotonicity of pWhile programs by a purely syntactic analysis. In our `pWhile` programs, the goal is to reach `state = 1` at the end of the program. The probability of reaching this state might be monotone in one or more parameters. By the theorems of Chapter 6, and the mapping from `pWhile` programs to processes in Chapter 7, we deduce conditions for monotonicity in `pWhile` programs.

This chapter presents conditions for monotonicity in the probability of reaching a specific variable valuation in:

- the effect of the assignment of program variables (`x := a`),
- Boolean expressions, and
- program statements:
  - `skip`,
  - `x := a`,
  - S_1 `[f]` S_2,
  - **`if`** `b` **`then`** S_1 **`else`** S_2,
  - **`while`** `b` **`do`** `S`, and
  - S_1`;`S_2.

First of all, we make some general observations on the influence of a program statement on the value of a program variable (Section 8.1). More precisely, we consider the probability that x increases or decreases. We show under which conditions this influence on x is monotone. Secondly, we show in Section 8.2 under which conditions the probability that a Boolean expression holds is monotone. Finally, we provide theorems on the conditions for monotonicity in the program statements of pWhile (Section 8.3).

Notation. We use the following notation:

- Let $\Pr(\eta \models \mathtt{b})$ be the probability that Boolean expression b holds given valuation η: $Var \to \mathbb{Z}_{\text{bound}}$.
- For program Prog, let η_{pre} be the valuation of the program variables before executing Prog, and $\eta_{\text{post}} = \eta_{\text{pre}}\{\mathtt{Prog}\}$ be the valuation of the program variables after executing Prog.
- For program statement S, let η_{pre} be the valuation of the program variables before executing S, and $\eta_{\text{post}} = \eta_{\text{pre}}\{\mathtt{S}\}$ be the valuation of the program variables after executing S.
- Let $\mathtt{S}_{i-j}$ denote the statements on Lines i-j of a program or program statement.
  Let $\eta_{\text{post}}^{S_{i-j}} = \begin{cases} \eta_{\text{pre}}\{\mathtt{S}_{i-j}\} & \text{if } i = 1 \\ \eta_{\text{post}}^{1-(i-1)}\{\mathtt{S}_{i-j}\} & \text{otherwise} \end{cases}$
- Let n be the number of lines in program Prog. Let program statement $\mathtt{S}_1$ end at line $i < n$ of Prog and program statement $\mathtt{S}_2$ start at line $i+1$ the following holds:
  $$\eta_{\text{post}}^{i} = \eta_{\text{pre}}^{i+1}$$
  Furthermore, $\eta_{\text{post}}^{n} = \eta_{\text{post}}$.
- As in Chapter 7, let ⍨ denote `state = 1`, and ⍩ denote `state = 0`.

Assumptions. We make the following assumptions:

- All program runs of program Prog visit *finitely* many states, and almost surely reach ⍨ or ⍩. Therefore, for the pMC of process P(Prog): $\Pr(\Diamond ⍨) + \Pr(\Diamond ⍩) = 1$.
- Before executing program Prog, all program variables x have valuation $\eta(\mathtt{x}) = 0$. Without this assumption, e.g. $\mathtt{x} \leq 1$ is not defined when $\eta(\mathtt{x}) = \bot$.
- Program statement S is executed given current valuation η_{pre}
- We only consider functions f for which:
  - $\frac{\partial}{\partial p} f$ exists, and
  - every total valuation $u \in Dom(f)$ is well-defined (Definition 2.1.5).

If **while** b **do** S is part of the program statement (e.g. in Section 8.3.3), then we assume all total valuations $u \in Dom(f)$ are graph-preserving (Definition 2.1.6).

We need these assumptions, to be able to apply the theorems of Chapter 6.

Remark. In most theorems we show $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\uparrow_p$. As $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\downarrow_p$ is an analogous statement, we omit this.

8.1 General Considerations

In this section, we consider the influence of a program statement on the value of program variable x. Recall from Definition 2.5.1 that any program variable is defined on a bounded integer interval $\mathbb{Z}_{\text{bound}}$. We use $\mathbb{N}_{\text{bound}}$ as the set of non-negative integers in $\mathbb{Z}_{\text{bound}}$.

We consider two different effects of program statement S on the value of program variable x. The effects are as follows:

1. $\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})$: that is x increases or remains the same ($\mathtt{x}_{\nearrow}$).
2. $\eta_{\text{pre}}(\mathtt{x}) > \eta_{\text{post}}(\mathtt{x})$: that is x decreases ($\mathtt{x}_{\swarrow}$).

Similar to Definition 7.2.4, we obtain the process of a program statement S, of which the goal is to either obtain $\mathtt{x}_{\nearrow}$ or $\mathtt{x}_{\swarrow}$.

Definition 8.1.1 (Process of a program statement with goal $\mathbf{x}_{\swarrow}$ or $\mathbf{x}_{\nearrow}$) Let S be a program statement. The process of S, with goal $\mathtt{x}_{\nearrow}$, is given by the process of the following program:

```
x_old := x;
S;
state := x >= x_old;
return;
```

The process of S, with goal $\mathtt{x}_{\swarrow}$, is given by the process of the following program:

```
x_old := x;
S;
state := x < x_old;
return;
```

∗

***Example* 8.1**
Let program statement S be `x := x + 1`. S increments x. For any valuation η_{pre}, $\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})$, so $\mathtt{x}_{\nearrow}$ follows from S. ∗

As $\neg(\eta_{\text{pre}}(\mathtt{x}) > \eta_{\text{post}}(\mathtt{x}))$ is equivalent to: $\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})$. We observe that if $\mathtt{x}_{\nearrow}$ follows from S, then $\neg(\mathtt{x}_{\swarrow})$ follows from S.

Corollary 8.1
For any program variable x, it holds that:

- $\neg(\mathtt{x}_{\nearrow}) = \mathtt{x}_{\swarrow}$, and
- $\neg(\mathtt{x}_{\swarrow}) = \mathtt{x}_{\nearrow}$ *

We consider the influence of a program statement S, when program variable x does *not occur* in S. Clearly, x remains the same during execution of S. From this observation we obtain Proposition 8.2.

Proposition 8.2
For every program statement S: $\mathtt{x} \notin Var_{\mathtt{S}} \Longrightarrow \Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big) = 1$ *

As $\mathtt{x}_{\swarrow}$ is the complement of $\mathtt{x}_{\nearrow}$, it immediately follows:

$$\mathtt{x} \notin Var_{\mathtt{S}} \Longrightarrow \Pr\big(\eta_{\text{pre}}(\mathtt{x}) > \eta_{\text{post}}(\mathtt{x})\big) = 0$$

Proof of Proposition 8.2
If $\mathtt{x} \notin Var_{\mathtt{S}}$, then S does not change x. So, for goal $\mathtt{x}_{\nearrow}$, we obtain for S that the associated process is equivalent to $\lightning$. Therefore, we have:

$$\Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big) = 1$$ *

We now look at the influence of the executing program statement S.

***Example* 8.2**
Let S be the following program statement:

```
x := 3 [f] x := -3
```

Suppose in the current valuation η_{pre}, $\eta_{\text{pre}}(\mathtt{x}) = 0$. So, if `x := 3` is executed, then $\mathtt{x}_{\nearrow}$, otherwise, $\mathtt{x}_{\swarrow}$. Let `b := x = 3`, we observe for S, that $\Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big) = f$. Furthermore, $\Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)$ directly influences $\Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big)$ and $\Pr\big(\eta_{\text{pre}}(\mathtt{x}) > \eta_{\text{post}}(\mathtt{x})\big)$.

Remark. In Section 8.2, we provide theorems on the monotonicity in the probability that a given Boolean expression holds.

*

Theorem 8.3
Let `b := x = a` be a Boolean expression. For every program statement `S`, it holds that:

- $\Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)\uparrow_p$ and $\eta\,(\mathtt{x}) \leq \eta\,(\mathtt{a}) \implies \Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big) \models \mathtt{b})\uparrow_p$
- $\Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)\uparrow_p$ and $\eta\,(\mathtt{x}) > \eta\,(\mathtt{a}) \implies \Pr\big(\eta_{\text{pre}}(\mathtt{x}) > \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p$ $*$

Proof of Theorem 8.3
We provide the proof of Case 1; the proof of Case 2 is obtained in a similar manner.

Let $\Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big) = f$. Assume $f\uparrow_p$. If $\eta(\mathtt{x}) \leq \eta(\mathtt{a})$, with probability f, `S` either increases `x`, or leaves `x` unchanged. By Definition 7.2.4 and reducing the obtained process, we deduce that the process of `S` with the goal of reaching $\mathtt{x}_{\nearrow}$, is equivalent to u_f. From Lemma 6.2, $\Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p$ follows. $*$

Since $\mathtt{x} \in \mathbb{Z}_{\text{bound}}$ (Definition 2.5.1), there is always an implicit upper and lower bound on `x`. We omit this for the sake of readability. Furthermore, the program may induce other upper and lower bounds on `x`. We consider programs in which an if statement induces a bound on the value of program variable. Let `min(x, MAX)` and `max(x, MIN)` be syntactic sugar for these bounds, with $\mathtt{MIN}, \mathtt{MAX} \in \mathbb{Z}_{\text{bound}}$ fixed constants. We observe that, also in a while loop a bound on the value of a program variable can be induced. Propositions for this are obtained in a similar manner.

***Example* 8.3**
Let program statement `S` be

```
x := x + 1;
x := min(x, 10);
```

`S` increases `x` with `1`, at the same time, it induces an upper bound on `x` of `10`. $*$

We observe that if we execute program statement `S` with current valuation η_{pre}, and $\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})$, then $\mathtt{x}_{\nearrow}$ follows from `S`. Otherwise, $\mathtt{x}_{\swarrow}$ follows from `S`. This observation leads to Proposition 8.4.

Proposition 8.4
Let $\mathtt{MAX} \in \mathbb{Z}_{\text{bound}}$, and let program statement `S` be:

```
1  S1;
2  x := min(x, MAX);
```

We distinguish two cases:

1. If $\eta(\mathtt{x}) \leq \mathtt{MAX}$, then:

$$\Pr\big(\eta_{\text{pre}}(1) \leq \eta_{\text{post}}(1)\big)\uparrow_p \iff \Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p$$

2. If $\eta(\mathtt{x}) > \mathtt{MAX}$, then:

$$\Pr\big(\eta_{\text{pre}}(\mathtt{x}) > \eta_{\text{post}}(\mathtt{x})\big) = 1 \qquad *$$

Proof of Proposition 8.4
First consider Case 1; let $\eta(\mathtt{x}) \leq \mathtt{MAX}$, and let $f = \Pr\big(\eta_{\text{pre}}(1) \leq \eta_{\text{post}}(1)\big)$.
We want to show: $\Pr\big(\eta_{\text{pre}}(1) \leq \eta_{\text{post}}(1)\big)\uparrow_p \iff \Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p$. For goal $\mathtt{x}_{\nearrow}$, we obtain that the process of S is equivalent to u_f. By Lemma 6.2, we obtain:

$$\Pr\big(\eta_{\text{pre}}(1) \leq \eta_{\text{post}}(1)\big)\uparrow_p \iff \Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p$$

Secondly, consider Case 2; let $\eta(\mathtt{x}) > \mathtt{MAX}$.
We want to show: $\Pr\big(\eta_{\text{pre}}(\mathtt{x}) > \eta_{\text{post}}(\mathtt{x})\big) = 1$. For goal, $\mathtt{x}_{\swarrow}$, we obtain for S that the associated process is equivalent to $\circlearrowleft$. Therefore, we have:

$$\Pr\big(\eta_{\text{pre}}(\mathtt{x}) > \eta_{\text{post}}(\mathtt{x})\big) = 1 \qquad *$$

Above we provide a proposition for variable assignment with an upper bound. In a similar manner we obtain a proposition for variable assignment with a lower bound.

Proposition 8.5
Let $\mathtt{MIN} \in \mathbb{Z}_{\text{bound}}$, and let program statement S be:

```
1  S1;
2  x := max(x, MIN);
```

We distinguish two cases:

1. If $\eta(\mathtt{x}) > \mathtt{MIN}$, then:

$$\Pr\big(\eta^1_{\text{pre}}(\mathtt{x}) > \eta^1_{\text{post}}(\mathtt{x})\big)\uparrow_p \iff \Pr\big(\eta_{\text{pre}}(\mathtt{x}) > \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p$$

2. If $\eta(\mathtt{x}) \leq \mathtt{MIN}$, then:

$$\Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big) = 1 \qquad *$$

The proof of Proposition 8.5 is similar to the proof of Proposition 8.4.

8.2 Boolean expressions

In Section 8.1, we introduced monotonicity obtained from the assignment of program variables. We considered two different cases:

1. x increases or remains the same ($\mathtt{x}_{\nearrow}$).
2. x decreases ($\mathtt{x}_{\swarrow}$).

In this section, we show how monotonicity in the probability that a Boolean expression b holds is obtained from the assignment of program variables. We provide propositions for $\{=,\ <=,\ \neg,\ \text{and}\ \wedge\}$. Propositions for $\{<,\ >=,\ >,\ \neq,\ \text{and}\ \vee\}$ are obtained in a similar manner.

First of all, we consider the assignment of a program variable and Boolean expressions x = a and x <= a. For both $\mathtt{x}_{\nearrow}$ and $\mathtt{x}_{\swarrow}$ we provide two propositions:

Proposition 8.6
Consider statement S.

a. Let b be a Boolean expression denoted by x = MAX, and let $\eta_{\text{pre}}(\mathtt{x}) \leq \mathtt{MAX}$. Then:

$$\Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p \Longrightarrow \Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)\uparrow_p$$

b. Let b be a Boolean expression denoted by x <= a. Then:

$$\Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p \Longrightarrow \Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)\downarrow_p \qquad *$$

Proof of Proposition 8.6
We provide the proof for Proposition 8.6.a. The other proof is obtained in a similar manner.

Let $f = \Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big)$. If $\eta_{\text{post}} \models [\![\mathtt{b}]\!]$ then the associated process is equivalent to u_f, thus $\Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p \Longrightarrow \Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)\uparrow_p$ follows. If $\eta_{\text{post}} \not\models [\![\mathtt{b}]\!]$ then $\Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big) = 0$, the associated process is equivalent to $\ddot{\frown}$, by Lemma 6.1 $\Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)\uparrow_p$. *

***Example* 8.4**
Let program Prog be the program in which S is executed n times. Assume we need to execute S n times to obtain:

$$\eta^n_{\text{post}} \models [\![\mathtt{b}]\!]$$

The associated process is equivalent to:

$$u_f{}^n = u_f?(u_f? \ldots : \ddot{\frown}) : \ddot{\frown}$$

By Corollary 6.4.a we obtain:

$$\Pr\big(\eta_{\text{pre}}(\mathtt{x}) \leq \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p \Longrightarrow \Pr\big(\eta^n_{\text{post}} \models [\![\mathtt{b}]\!]\big)\uparrow_p$$

Note that this also follows by Proposition 8.6 and Theorem 8.19. ∗

Proposition 8.7
Consider statement `S`.

a. Let `b` be a Boolean expression denoted by `x = MIN`, and let $\eta_{\text{post}}(\mathtt{x}) > \mathtt{MIN}$. Then:

$$\Pr\big(\eta_{\text{pre}}(\mathtt{x}) > \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p \Longrightarrow \Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)\uparrow_p$$

b. Let `b` be a Boolean expression denoted by `x <= a`. Then:

$$\Pr\big(\eta_{\text{pre}}(\mathtt{x}) > \eta_{\text{post}}(\mathtt{x})\big)\uparrow_p \Longrightarrow \Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)\uparrow_p \quad *$$

The proof of Proposition 8.7 is obtained in a similar manner as the proof of Proposition 8.6.

We have now defined propositions for `x = a` and `x <= a`. Next, we define a proposition for the negation of a Boolean expression. We observe that if `b` holds with probability f, $\neg$`b` holds with probability $1 - f$.

Proposition 8.8
Consider program statement `S`. Let `b` be a Boolean expression on the program variables in η. Then:

$$\Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)\uparrow_p \Longleftrightarrow \Pr\big(\eta_{\text{post}} \models [\![\neg\mathtt{b}]\!]\big)(\eta)\downarrow_p \quad *$$

Proof of Proposition 8.8
We proof the $\Longrightarrow$ direction; in a similar manner $\Longleftarrow$ is obtained.

The associated process of `S`, with the goal that `b` holds after executing `S`, is equivalent to u_f with $f = \Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)$. As by assumption every process almost surely terminates, the probability of reaching $\lightning$, is given by $1 - f$. At $\lightning$, $\neg$`b` holds, so when the goal is to reach $\lightning$, this is equivalent to the process `P` denoted by $u_f ?\lightning : \lightning$. If $f\uparrow_p$, then we obtain from Theorem 6.6 for the underlying pMC $\mathcal{M}$ of `P`: $sol_{\mathcal{M}}\downarrow_p$. So:

$$\Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\big)\uparrow_p \Longrightarrow \Pr\big(\eta_{\text{post}} \models [\![\neg\mathtt{b}]\!]\big)(\eta)\downarrow_p \quad *$$

Furthermore, we consider the conjunction of two Boolean expressions.

Proposition 8.9
Consider program statement `S`. Let $\mathtt{b}_1$ and $\mathtt{b}_2$ be Boolean expressions. Then:

$$\Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}_1]\!]\big)\uparrow_p \text{ and } \Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}_2]\!]\big)\uparrow_p \Longrightarrow \Pr\big(\eta_{\text{post}} \models [\![\mathtt{b}_1 \wedge \mathtt{b}_2]\!]\big)\uparrow_p \quad *$$

Proof of Proposition 8.9
Let $f = \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}_1]\!]\right)$ and $g = \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}_2]\!]\right)$. As $\mathtt{b}_1$ and $\mathtt{b}_2$ are expressions, f and g are independent. So, the process of S with goal b, is equivalent to $u_f ?u_g : \ddot{\smile}$, then from Corollary 6.4.a it follows that $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}_1 \wedge \mathtt{b}_2]\!]\right)\uparrow_p$. So:

$$\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}_1]\!]\right)\uparrow_p \text{ and } \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}_2]\!]\right)\uparrow_p \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}_1 \wedge \mathtt{b}_2]\!]\right)\uparrow_p \quad *$$

Above we defined the influence of program statement S on the probability that a Boolean expression holds, it might also be possible that this probability is constant in parameter p. Similar to Lemma 6.1, we obtain Proposition 8.10. Let $Var_{\mathtt{b}}$ denote the program variables occurring in Boolean expression b.

Proposition 8.10
Consider program statement S. Let b be a Boolean expression. Then:

$$\begin{aligned}&\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right) \text{ is constant in } p\\ &\qquad\qquad \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\uparrow_p \text{ and } \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\downarrow_p \qquad *\end{aligned}$$

The proof of Proposition 8.10 follows directly from Lemma 6.1.

At the end of Section 8.1 we consider the influence on $\mathtt{x}_{\nearrow}$ and $\mathtt{x}_{\swarrow}$, given that program variable x does not occur. This results in Proposition 8.2. Similarly, variables in a Boolean expression might not occur in a program statement, we obtain in a similar manner. Proposition 8.11.

Proposition 8.11
Consider program statement S. Let b be a Boolean expression. Then:

$$(\forall \mathtt{x} \in Var_{\mathtt{b}}.\mathtt{x} \notin Var_{\mathtt{S}}) \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right) = \Pr\left(\eta_{\text{pre}} \models [\![\mathtt{b}]\!]\right) \qquad *$$

Proof of Proposition 8.11
If $\forall \mathtt{x} \in Var_{\mathtt{b}}.\mathtt{x} \notin Var_{\mathtt{S}}$, then $\forall \mathtt{x} \in Var_{\mathtt{b}}.\eta_{\text{pre}}(\mathtt{x}) = \eta_{\text{post}}(\mathtt{x})$.
So, $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right) = \Pr\left(\eta_{\text{pre}} \models [\![\mathtt{b}]\!]\right)$. *

From Propositions 8.10 and 8.2 follows:

Corollary 8.12
Consider program statement S. Let b be a Boolean expression. Then:

$$\begin{aligned}&\forall(\mathtt{x} \in Var_{\mathtt{b}}.\mathtt{x} \notin Var_{\mathtt{S}}) \text{ and } \Pr\left(\eta_{\text{pre}} \models [\![\mathtt{b}]\!]\right) = c, c \in [0,1]\\ &\qquad\qquad \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\uparrow_p \text{ and } \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\downarrow_p \quad *\end{aligned}$$

8.3 Program statements

In this section, we provide conditions for monotonicity of the program statements:

- `skip`,
- `x := a`,
- $\mathtt{S}_1$ `[f]` $\mathtt{S}_2$,
- **`if`** `b` **`then`** $\mathtt{S}_1$ **`else`** $\mathtt{S}_2$,
- **`while`** `b` **`do`** `S`, and
- $\mathtt{S}_1$`;`$\mathtt{S}_2$.

8.3.1 *Empty statement and variable assignment*

In this section, we consider the empty statement (`skip`) and the variable assignment (`x := a`). First of all, we consider statement `skip`. Clearly, `skip` does not influence the values of the program variables. Proposition 8.11 yields Corollary 8.13.

Corollary 8.13 (Monotonicity of the empty statement)
Consider program statement `S = skip`. Let `b` be a Boolean expression. Then:

$$\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\uparrow_p \text{ and } \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\downarrow_p \qquad *$$

Secondly, we consider the direct assignment of program variables. Recall that in Section 8.1, we consider the influence of this assignment.

Proposition 8.14 (Monotonicity of variable assignment)
Consider program statement `S` given by `x := a`. Let Boolean expression `b` be denoted by `x := a`. Then:

$$\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right) = 1 \qquad *$$

Proof of Proposition 8.14
Let program statement `S` be `x := a`. The process of `S`, with as goal `x = a` is equivalent to $\Cup$. Clearly, in $\Cup$, $\Pr(\Diamond\Cup) = 1$. So,

$$\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right) = 1 \qquad *$$

8.3.2 *Probabilistic choice and if statement*

In this section we consider the probabilistic choice and the if statement.

***Example* 8.5**
Consider the following program statement S:

```
x := x + 2 [f] x := x - 3;
```

In S, with probability f, `x := x + 2` is executed, and with probability $1-f$, `x := x - 3` is executed. Let b be given by `x >= 10`. We observe the following:

- If $\eta_{\mathtt{pre}}(\mathtt{x}) \geq 10$, then $\eta_{\text{pre}} \models [\![\mathtt{b}]\!]$. If `x := x + 2` is executed, x does not decrease, $\eta_{\text{post}} \models [\![\mathtt{b}]\!]$. However, when `x := x - 3` is executed, x might become smaller than 10, so it may be that $\eta_{\text{post}} \not\models [\![\mathtt{b}]\!]$.
- If $\eta(\mathtt{x}) < 10$, then $\eta \not\models [\![\mathtt{b}]\!]$ hold. If `x := x + 2` is executed, x might become larger than 10, so it may be that $\eta_{\text{pre}} \models [\![\mathtt{b}]\!]$. When `x := x - 3` is executed, x does not increase, so $\eta_{\text{pre}} \not\models [\![\mathtt{b}]\!]$.

$\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)$ is directly influenced by f. Therefore,

$$\text{if } f{\uparrow_p}, \text{ then } \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \qquad *$$

Theorem 8.15 (Monotonicity of probabilistic choice)
Consider program statement $\mathtt{S} = \mathtt{S}_1$ `[f]` $\mathtt{S}_2$, with $f{\uparrow_p}$. Let b be a Boolean expression. Then:

$$\begin{aligned} &\Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \text{ and } \Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \\ &\qquad \text{and } \Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right) \geq \Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right) \\ &\qquad\qquad \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \end{aligned} \qquad *$$

Proof of Theorem 8.15
Let $g = \Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right)$ and $h = \Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right)$. Then, the process of $\mathtt{S}_1$ `[f]` $\mathtt{S}_2$ is equivalent to $u_f ? u_g : u_h$, the result follows directly from Theorem 6.3. *

Corollary 8.16 follows directly from Lemma 7.1 and Theorem 8.15.

Corollary 8.16 (Monotonicity of if statements)
Consider program statement $\mathtt{S} =$ **`if`** $\mathtt{b}_1$ **`then`** $\mathtt{S}_1$ **`else`** $\mathtt{S}_2$.

Let $f = \Pr\left(\eta_{\text{pre}} \models [\![\mathtt{b}_1]\!]\right)$, let $f{\uparrow_p}$, and let $\mathtt{b}$ a Boolean expression. Then:

$$\begin{aligned} \Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \text{ and } \Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \\ \text{and } \Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right) \geq \Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right) \\ \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \end{aligned}$$ *

8.3.3 *While loops*

In this subsection, we consider the while loop with the following structure:

```
while counter < MAX ∧ b do
    S1;
```

Listing 8.1: While loop

Recall that by assumption all valuations of functions occurring in a while loop are graph-preserving valuations (Definition 2.1.6).

***Example* 8.6**
Consider program statement `S` in Listing 8.2. In this program a coin is flipped, and either the state is set to `0`, or `1`. When the `state` equals `0`, or the counter reached a maximum, we exit the while loop. The goal is, to eventually reach process term $\lightning$ in the process of a program. Therefore, we prefer to exit the while loop with `state = 1`. We make the following observations:

- Influence of `counter`:
  - `counter` is increased in every loop iteration. This increase is independent of any parameter.
  - As we want $\eta_{\text{post}} \models [\![\texttt{state = 1}]\!]$, we want to exit the while loop with `counter >= MAX`.
- Relation `x = 0` and `state = 1`:
  - The probability that `x := 0` is executed equals f.
  - When we look at the body of the while loop, we observe that `x = 0` must hold, to obtain `state = 1`. So, probability f should be as high as possible.

As `counter` is not influenced by any parameter, we obtain from the observations: if $f{\uparrow_p}$, then $\Pr\left(\eta \models [\![\texttt{state=1}]\!]\right){\uparrow_p}$. *

```
while counter < MAX ∧ state = 1 do
    counter := counter + 1;
    x := 0 [f] x := 1;
    state := x = 0;
```

Listing 8.2: Example of a while loop

In Example 8.6, we observe that if in the i^{th} loop iteration:

$$\Pr\left(\eta_{\text{pre}_i}(\texttt{counter}) \leq \eta_{\text{post}_i}(\texttt{counter})\right)\uparrow_p \text{ and } \Pr\left(\eta_{\text{post}_i} \models [\![\texttt{state=1}]\!]\right)\uparrow_p$$

then

$$\Pr\left(\eta_{\text{post}} \models [\![\texttt{state=1}]\!]\right)\uparrow_p$$

We generalize this observation in Theorems 8.17 and 8.18.

Theorem 8.17 (Monotonicity While loop)
Consider program statement S:

```
while counter < MAX ∧ b do
    S1;
```

Let:

- $\eta_{\text{pre}}(\texttt{counter}) = 0$,
- i denote the i^{th} loop iteration,
- $f_i = \Pr\left(\eta_{\text{post}_i} \models [\![\texttt{b}]\!]\right)$,
- $g_i = \Pr\left(\eta_{\text{pre}_i}(\texttt{counter}) + 1 = \eta_{\text{post}_i}(\texttt{counter})\right)$, and
- $\Pr\left(\eta_{\text{pre}_i}(\texttt{counter}) + 1 = \eta_{\text{post}_i}(\texttt{counter})\right)$
  $+ \Pr\left(\eta_{\text{post}_i} \models [\![\texttt{counter=0}]\!]\right) = 1$

Then:

1. $\forall i. f_i\uparrow_p$ and $g_i\uparrow_p \implies \Pr\left(\eta_{\text{post}} \models [\![\texttt{b}]\!]\right)\uparrow_p$
2. $\forall i. f_i\uparrow_p$ and $g_i = 1 - c \cdot f_i$, $c \in (0, 1] \implies \Pr\left(\eta_{\text{post}} \models [\![\texttt{b}]\!]\right)\uparrow_p$
3. $\forall i. f_i\downarrow_p$ and $g_i\downarrow_p \implies \Pr\left(\eta_{\text{post}} \models [\![\texttt{b}]\!]\right)\downarrow_p$
4. $\forall i. f_i\downarrow_p$ and $g_i = 1 - c \cdot f_i$, $c \in (0, 1] \implies \Pr\left(\eta_{\text{post}} \models [\![\texttt{b}]\!]\right)\uparrow_p$ *

Proof of Theorem 8.17
We provide the proof of Case 1 for statement `S` in Listing 8.1; the proof of Case 2 is obtained in a similar manner.

Let m be the number of times the while loop is executed.

As $\eta_{\text{pre}}(\texttt{counter}) = 0$, we can write the process of `S` as follows:

$$\begin{aligned}
\texttt{P(S)} &= \texttt{Proc}(m) \\
\texttt{Proc}(0) &= \begin{cases} \overset{\cup}{\cdot\cdot} & \text{if } \eta_{\text{pre}} \models [\![\mathtt{b}]\!] \\ \overset{\cap}{\cdot\cdot} & \text{if } \eta_{\text{pre}} \not\models [\![\mathtt{b}]\!] \end{cases} \\
\texttt{Proc}(m) &= \begin{cases} \overset{\curvearrowleft}{\cdot\cdot}_g \; u_{f_1} & \text{if } m = 1 \\ \overset{\curvearrowleft}{\cdot\cdot}_g \; (\texttt{Proc}(m-1) ? u_{f_m} : \overset{\cap}{\cdot\cdot}) & \text{otherwise} \end{cases}
\end{aligned}$$

Let pMC $\mathcal{M}$ be the pMC of `P(S)`. We show for each value of m, $sol_{\mathcal{M}}\!\uparrow_p$. The proof is by induction on m.

- $m = 0$:
  `Proc(0)` is either $\overset{\cup}{\cdot\cdot}$ or $\overset{\cap}{\cdot\cdot}$. Since, $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right) = \begin{cases} 1 & \text{if } \eta \models [\![\mathtt{b}]\!] \\ 0 & \text{otherwise} \end{cases}$
  By Lemma 6.1, $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\!\uparrow_p$, $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\!\downarrow_p$, $\Pr\left(\eta_{\text{post}} \models [\![\neg\mathtt{b}]\!]\right)\!\uparrow_p$, and $\Pr\left(\eta_{\text{post}} \models [\![\neg\mathtt{b}]\!]\right)\!\downarrow_p$ hold. So, $sol_{\mathcal{M}}\!\uparrow_p$ for $m = 0$.

- $m = 1$:
  `Proc(1)` is $\overset{\curvearrowleft}{\cdot\cdot}_g \; u_{f_1}$, since $f_1\!\uparrow_p$ and $g_1\!\uparrow_p$, From Theorem 6.13, $sol_{\mathcal{M}}\!\uparrow_p$ follows for $m = 1$.

- $m > 1$:
  We assume $sol_{\mathcal{M}}\!\uparrow_p$ for some fixed $m = k \geq 1$.

  $$\texttt{Proc(k)} = \overset{\curvearrowleft}{\cdot\cdot}_g \; (\texttt{Proc}(k-1) ? u_{f_k} : \overset{\cap}{\cdot\cdot})\!\uparrow_p \tag{IH}$$

  We need to show for $k+1$: $sol_{\mathcal{M}}\!\uparrow_p$. We observe that:

  $$\begin{aligned}
  \texttt{Proc(k + 1)} &= \overset{\curvearrowleft}{\cdot\cdot}_g \; (\texttt{Proc}(k) ? u_{f_{k+1}} : \overset{\cap}{\cdot\cdot}) \\
  &= \overset{\curvearrowleft}{\cdot\cdot}_g \; ((\overset{\curvearrowleft}{\cdot\cdot}_g \; (\texttt{Proc}(k-1) ? u_{f_k} : \overset{\cap}{\cdot\cdot})) ? u_{f_{k+1}} : \overset{\cap}{\cdot\cdot})
  \end{aligned}$$

  By (IH) we obtain: $\overset{\curvearrowleft}{\cdot\cdot}_g \; (\texttt{Proc}(k-1) ? u_{f_k} : \overset{\cap}{\cdot\cdot})\!\uparrow_p$. Therefore, we can replace $\overset{\curvearrowleft}{\cdot\cdot}_g \; (\texttt{Proc}(k-1) ? u_{f_k} : \overset{\cap}{\cdot\cdot})$ by u_h with $h\!\uparrow_p$.

  $$\texttt{Proc(k + 1)} = \overset{\curvearrowleft}{\cdot\cdot}_g \; (u_h ? u_{f_{k+1}} : \overset{\cap}{\cdot\cdot})$$

  As $f_{k+1}\!\uparrow_p$ and $h\!\uparrow_p$ we deduce from Corollary 6.4.a, $(u_h ? u_{f_{k+1}} : \overset{\cap}{\cdot\cdot})\!\uparrow_p$. By Theorem 6.13, we now obtain $\texttt{Proc}(k+1)\!\uparrow_p$.

Clearly, if $sol_{\mathcal{M}}\uparrow_p$, then $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\uparrow_p$. As we have proven by induction on m the first hold, $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\uparrow_p$ holds as well. $*$

In Theorem 8.17, we look at the probability that `counter` is incremented by 1, and the probability that `counter` is reset to 0. In some case studies, `counter` is not reset to 0, but remains the same. Therefore, we obtain Theorem 8.18.

Theorem 8.18
Let `S`, i, f_i and g_i as in Theorem 8.17, and let:

- $\eta_{\text{pre}}(\mathtt{counter}) = 0$,
- $\Pr\left(\eta_{\text{pre}_i}(\mathtt{counter}) + 1 = \eta_{\text{post}_i}(\mathtt{counter})\right)$
  $+\Pr\left(\eta_{\text{pre}_i}(\mathtt{counter}) = \eta_{\text{post}_i}(\mathtt{counter})\right) = 1$

Then:

1. $\forall i.f_i\uparrow_p$ and $g_i\uparrow_p \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\uparrow_p$
2. $\forall i.f_i\uparrow_p$ and $g_i = 1 - c \cdot f_i$, $c \in (0,1] \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\uparrow_p$
3. $\forall i.f_i\downarrow_p$ and $g_i\downarrow_p \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\downarrow_p$
4. $\forall i.f_i\downarrow_p$ and $g_i = 1 - c \cdot f_i$, $c \in (0,1] \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right)\uparrow_p$ $*$

Proof of Theorem 8.18
The proof is obtained in a similar manner as the proof of Theorem 8.17. Note that the process of `S` for Case 1 is given as follows:

$$\mathtt{P(S)} = \mathtt{Proc}(m)$$

$$\mathtt{Proc}(m) = \begin{cases} \text{"}\smile\text{"} & \text{if } m = 0 \text{ and } \eta \models [\![\mathtt{b}]\!] \\ \text{"}\frown\text{"} & \text{if } m = 0 \text{ and } \eta \models [\![\neg\mathtt{b}]\!] \\ \text{"}\smile\text{"}_g\ u_{f_1} & \text{if } m = 1 \\ \mathtt{Proc}(m-1)\ ?\ \text{"}\smile\text{"}_g\ u_{f_m} : \text{"}\frown\text{"} & \text{otherwise} \end{cases}$$

$*$

8.3.4 *Sequential composition*

In the previous subsections, we provide theorems for conditions for monotonicity in all program statements of Definition 2.5.1 on page 17 except composition of program statements (`S1;S2`). In this subsection, we provide conditions for monotonicity in the probability that `b` holds after executing `S1;S2`.

***Example* 8.7**
Consider the following program statements:

```
S1: x := x + 2 [f] x := x - 3;
S2: x := x + 1 [g] x := x - 2;
```

Let S be the composition of $\mathtt{S}_1$ and $\mathtt{S}_2$ ($\mathtt{S}_1\mathtt{;}\,\mathtt{S}_2$), and let b be given by `x >= 10`. By assumption, $\eta(\mathtt{x}) = 0$. So, from Theorem 8.15 it follows that:

- $f{\uparrow_p} \Longrightarrow \Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p}$
- $g{\uparrow_p} \Longrightarrow \Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p}$

Now for S, we observe that if both $f{\uparrow_p}$ and $g{\uparrow_p}$, then $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p}$. ∗

Theorem 8.19 (Monotonicity of sequential composition)
Consider program statement $\mathtt{S} = \mathtt{S}_1\mathtt{;}\,\mathtt{S}_2$. Let b be a Boolean expression. Then:

$$\Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \text{ and } \Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \qquad *$$

Proof of Theorem 8.19
We distinguish two cases:

1. $\eta_{\text{pre}} \models [\![\mathtt{b}]\!]$, and
2. $\eta_{\text{pre}} \not\models [\![\mathtt{b}]\!]$.

We make the following observations:

1. If $\eta_{\text{pre}} \models [\![\mathtt{b}]\!]$ and $\Pr\left(\eta^1_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p}$, then $\eta^1_{\text{post}} \models [\![\mathtt{b}]\!]$
2. If $\eta^1_{\text{post}} \models [\![\mathtt{b}]\!]$ and $\Pr\left(\eta^2_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p}$, then $\eta^2_{\text{post}} \models [\![\mathtt{b}]\!]$

Let $\Pr\left(\eta^1_{\text{post}} \models [\![\mathtt{b}]\!]\right) = f$ and $\Pr\left(\eta^2_{\text{post}} \models [\![\mathtt{b}]\!]\right) = g$. By the observations we obtain:

- Case 1: the associated process is equivalent to $\cup$
  If $\eta_{\text{pre}} \models [\![\mathtt{b}]\!]$, and $f{\uparrow_p}$, and $g{\uparrow_p}$, then by Observations 1 and 2 we obtain: $\eta^2_{\text{pre}} \models [\![\mathtt{b}]\!]$. Therefore, $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right) = 1$, and the associated process is equivalent to $\cup$. We obtain from Lemma 6.1: $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p}$.
- Case 2: the associated process is equivalent to $u_f ? \cup : u_g$
  We obtain from Corollary 6.4.b: $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p}$. Therefore,

$$\Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \text{ and } \Pr^{\mathtt{S}_2}_{\mathtt{b}}(\eta^1_{\text{pre}}){\uparrow_p} \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right){\uparrow_p} \qquad *$$

Theorem 8.20 (Not monotone)
Consider program statement $\mathtt{S} = \mathtt{S}_1;\mathtt{S}_2$. Let $\mathtt{b}$ be a Boolean expression, with $\eta_{\text{pre}} \not\models [\![\mathtt{b}]\!]$. The following holds:

1. $$\Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right) \not\checkmark_p \text{ and } \frac{\partial}{\partial p}\Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right) = 0 \text{ and } \Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right) < 1 \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right) \not\checkmark_p \text{ for } \mathtt{S}_1;\mathtt{S}_2$$

2. $$\Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right) \not\checkmark_p \text{ and } \frac{\partial}{\partial p}\Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right) = 0 \text{ and } \Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right) < 1 \Longrightarrow \Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right) \not\checkmark_p \text{ for } \mathtt{S}_1;\mathtt{S}_2$$ ∗

***Example* 8.8**
Let program statement $\mathtt{S} = \mathtt{S}_1;\mathtt{S}_2$ with:

```
S1: x := 0 [p·(1-p)] x := 1;
S2: y := 4;
```

Let b be x = 0. We observe that $\Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right) = p \cdot (1-p)$, which is not monotone. Furthermore, $\mathtt{S}_2$ does not change x, we observe the following:

- $\frac{\partial}{\partial p}\Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right) = 0$.
- As the probability that $\eta\{\mathtt{S}_1\} \models [\![\mathtt{b}]\!]$ is less than 1: $\Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right) < 1$.

By Theorem 8.20 we obtain: $\Pr\left(\eta_{\text{post}} \models [\![\mathtt{b}]\!]\right) \not\checkmark_p$. ∗

Proof of Theorem 8.20
Let $\Pr\left(\eta^{\mathtt{S}_1}_{\text{post}} \models [\![\mathtt{b}]\!]\right) = f$ and $\Pr\left(\eta^{\mathtt{S}_2}_{\text{post}} \models [\![\mathtt{b}]\!]\right) = h$. As $\eta \models [\![\neg\mathtt{b}]\!]$, the associated process is equivalent to $u_f ?\!\!\cup : u_h$. Clearly, $\cup$ is equivalent to u_g, with $g = 1$, and $\frac{\partial}{\partial p} g = 0$.

The results now follow directly from Theorem 6.5 Cases 3 and 1. ∗

Chapter 9

Case Studies

In this chapter, we apply the results from Chapter 8 on several existing benchmarks for parameter synthesis from the literature.

An overview of the results for the benchmarks is provided in Table 9.1. The column *Case Study* denotes which case study we consider. The column *Type* indicates whether the underlying pMC of the case study is acyclic or cyclic. The column *Parameters* denotes how many parameters occur in the case study, and the column *Monotone parameters* denotes how many of the parameters are monotone. The column *Successfully found* indicates how many of the monotone parameters we found by applying the results of Chapter 8.

For each of the case studies we elaborate on the results in separate sections below. In each section, we first describe the case study and provide their modeling by a `pWhile` program. Subsequently, we show how monotone substructures occur and how we apply the theorems of Chapter 8.

Notation. We let $\texttt{S}_{i-j}$ denote the statements of a program on Lines i-j.

9.1 BRP

The bounded retransmission protocol [4] (BRP) aims to send a file in a reliable manner. The sender and receiver communicate over two lossy unidirectional channels. Parameters p and q denote their reliability. The first channel (reliability p) is used for communication from the sender to the receiver and the second for communication from the receiver to the sender. The file is divided into N chunks. Iteratively, the sender sends a chunk and waits for an acknowledgement. If the acknowledgement is not received, then the sender re-transmits the chunk. For each chunk the number of retransmissions of the chunk and acknowledgements at most `MAX` times. So, for each chunk the following routine

Case study	**Type**	**Parameters**	**Monotone parameters**	**Successfully found**	**Remarks**
BRP	acyclic	2	2	2	Theorems Chapter 8
Zeroconf	cyclic	2	2	2	Theorems Chapter 8
Load-unload	acyclic	2	2	2	Theorems Chapter 8
Grids 9.4.1	acyclic	2	1	1	Theorems Chapter 8, given assumptions as in Section 9.4.1
Grids 9.4.2	cyclic	2	1	1	Theorems Chapter 8, given assumptions as in Section 9.4.2
Crowds [3]	cyclic	2	2	0	Theorems Chapter 8
				2	Theorems Chapter 6
Crowds [2]	cyclic	2	2	0	Theorems Chapter 8
				2	Theorems Chapter 6
NAND Multiplexing	acyclic	2	2	0	Associated `pWhile` program is too large to analyze

Table 9.1: Results for the case studies

is executed, as long as the retransmission bound has not been reached:

1. Send chunk
   - With probability p, it is received correctly, we move to step 2.
   - With probability $1 - p$, sending failed, and we need to repeat step 1.
2. Send acknowledge message
   - With probability q, the acknowledgment is received correctly, we move to the next chunk.
   - With probability $1-q$, the acknowledgment failed, and we need to repeat step 1.

The goal is to eventually reach the state in which all chunks are sent and acknowledged successfully.

The `pWhile` program of BRP ($\texttt{Prog}_{\text{BRP}}$) is given in Listing 9.1. At Lines 1 and 2, the program variables are assigned their initial value. Then a while loop (Lines 3-13) is entered. In this while loop, first the chunk is sent, and then, if the chunk is received properly (`first = 1`), an acknowledgement message is sent. When this acknowledgement is received by the original sender (`second = 1`), the number of sent chunks is increased, and the counter is reset. We observe that the while loop induces an upper bound of N on `sent` (Observation 1). Furthermore, we observe that parametric changes occur at Lines 4 and 6 (Observation 2). Also, we observe that in the body of the while loop, `count` is either increased, or reset to 0 (Observation 3).

Finally, at Line 14, if `sent` $= N$, then `state` is set to 1. `sent` $= N$ implies that all chunks have been sent successfully. By Observation 1, we obtain that `sent` $\leq N$. Therefore, at Line 14 `sent` is at most N, thus `sent` $= N$ is equivalent to `sent` $\geq N$ (Observation 4).

Figure 9.1 on the facing page shows the pMC of BRP for $N = 2$ and $\texttt{MAX} = 2$.

Remark. The states with only one outgoing edge with probability 1 can be eliminated. However, to make the model more understandable we keep these states.

We first present the results and formally deduce them afterwards.

Results. Let η be the variable valuation before executing the given program statement. We obtain the following results for the statements of $\texttt{Prog}_{\text{BRP}}$ in Listing 9.1:

1. Statements on Lines 1 and 2:

$$\Pr\left(\eta^{1-2}_{\text{post}} \models [\![\texttt{sent>=}N]\!]\right)\uparrow \text{ and } \Pr\left(\eta^{1-2}_{\text{post}} \models [\![\texttt{sent>=}N]\!]\right)\downarrow \qquad (1)$$

```
1  sent := 0;
2  count := 0;
3  while count < MAX ∧ sent < N do
4      first := 1 [p] first := 0;
5      if first = 1 then
6          second := 1 [q] second := 0;
7          if second = 1 then
8              sent := sent + 1;
9              count := 0;
10         else
11             count := count + 1;
12     else
13         count := count + 1;
14 state := sent = N;
15 return;
```

Listing 9.1: $\texttt{Prog}_{\text{BRP}}$, BRP in pWhile

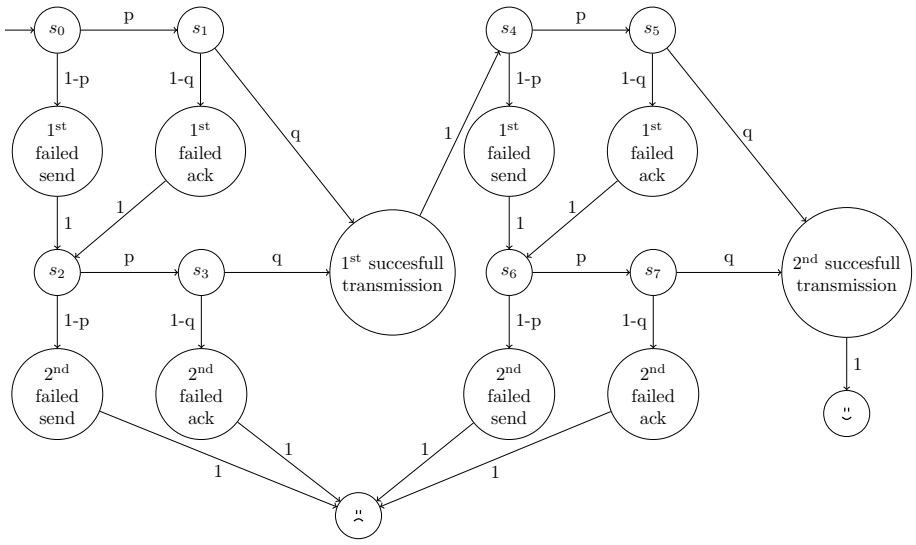


Figure 9.1: pMC of brp for $N = 2$ and $\texttt{MAX} = 2$

2. Statements on Lines 3-13:
   - We obtain for the body of the while loop for `count`:
   $$\Pr\left(\eta_{\text{pre}}^{4-13}(\texttt{count}) + 1 = \eta_{\text{post}}^{4-13}(\texttt{count})\right)\downarrow$$
   .
   - We obtain for the body of the while loop for `sent`:
   $$\Pr\left(\eta_{\text{post}}^{4-13} \models [\![\texttt{sent<}N]\!]\right)$$
   - We obtain for the while loop:
   $$\Pr\left(\eta_{\text{post}}^{3-13} \models [\![\texttt{sent<}N]\!]\right)\downarrow \tag{2}$$
3. Statement on Line 14
   We obtain:
   $$\Pr\left(\eta_{\text{post}}^{14} \models [\![\neg(\texttt{sent<}N)]\!]\right) = sol_{\text{BRP}}\uparrow \tag{3}$$

Combining (1) and (2), we obtain through Theorem 8.19:

$$\Pr\left(\eta_{\text{post}}^{1-13} \models [\![\texttt{sent<}N]\!]\right)\downarrow$$

.

By Proposition 8.8, we obtain:

$$\Pr\left(\eta_{\text{post}}^{1-13} \models [\![\texttt{sent>=}N]\!]\right)(\eta)\uparrow$$

.

From this result combined with (3) we obtain from Theorem 8.19: $sol_{\text{BRP}}\uparrow$.

Deduction. The results are formally obtained in the following manner:

Notation. Let sol_{BRP} denote the probability of eventually reaching $\lightning$ in the associated pMC of BRP, and let:

- $\mathtt{b}$ be given by $\texttt{sent} < N$.
- $\mathtt{b}_1$ be given by $\texttt{first} = 1$.
- $\mathtt{b}_2$ be given by $\texttt{second} = 1$.
- $\mathtt{b}_3$ be given by $\texttt{sent}_{\nearrow}$.
- $\mathtt{b}_4$ be given by $\texttt{count} = 0$.

We obtain the following results for the statements of $\texttt{Prog}_{\text{BRP}}$ in Listing 9.1:

1. Statements on Lines 1 and 2:
   By Proposition 8.14 we obtain:

$$\Pr\left(\eta^1_{\text{post}} \models [\![\texttt{sent=0}]\!]\right) = 1 \text{ and } \Pr\left(\eta^2_{\text{post}} \models [\![\texttt{count=0}]\!]\right) = 1$$

   By Observation 2 we obtain through Corollary 8.12 and Proposition 8.10 and Theorem 8.19, for Lines 1 and 2 that for all Boolean expressions $\texttt{b}$ defined above:

$$\Pr\left(\eta^{1-2}_{\text{post}} \models [\![\texttt{b}]\!]\right)\uparrow \text{ and } \Pr\left(\eta^{1-2}_{\text{post}} \models [\![\texttt{b}]\!]\right)\downarrow \tag{1}$$

2. Statements on Lines 3-13:
   First of all, we obtain the following:

   - For all Lines $i \in [3, 13]$ except Lines 4 and 6, we obtain from Corollary 8.12 and Proposition 8.10 $\forall * \in \{\texttt{b}, \texttt{b}_1, \texttt{b}_2, \texttt{b}_3, \texttt{b}_4\}$:

$$\begin{aligned} &\diamond \Pr\left(\eta^i_{\text{post}} \models [\![*]\!]\right)\uparrow \\ &\quad \Pr\left(\eta^i_{\text{post}} \models [\![*]\!]\right)\downarrow \\ &\diamond \Pr\left(\eta^i_{\text{pre}}(\texttt{count}) + 1 = \eta^i_{\text{post}}(\texttt{count})\right)\uparrow \\ &\quad \Pr\left(\eta^i_{\text{pre}}(\texttt{count}) + 1 = \eta^i_{\text{post}}(\texttt{count})\right)\downarrow \end{aligned} \tag{9.1}$$

   - Line 4, Corrolary 8.12, Proposition 8.10, Proposition 8.14, and Theorem 8.15:

$$\begin{aligned} &\diamond \Pr\left(\eta^4_{\text{post}} \models [\![\texttt{b}_1]\!]\right)\uparrow_p \\ &\diamond \Pr\left(\eta^4_{\text{post}} \models [\![\texttt{b}_1]\!]\right)\uparrow_q \\ &\quad \Pr\left(\eta^4_{\text{post}} \models [\![\texttt{b}_1]\!]\right)\downarrow_q \\ &\diamond \Pr\left(\eta^4_{\text{pre}}(\texttt{count}) + 1 = \eta^4_{\text{post}}(\texttt{count})\right)\uparrow \\ &\quad \Pr\left(\eta^4_{\text{pre}}(\texttt{count}) + 1 = \eta^4_{\text{post}}(\texttt{count})\right)\downarrow \\ &\diamond \forall * \in \{\texttt{b}, \texttt{b}_2, \texttt{b}_3, \texttt{b}_4\}\colon \Pr\left(\eta^4_{\text{post}} \models [\![*]\!]\right)\uparrow \text{ and } \Pr\left(\eta^4_{\text{post}} \models [\![*]\!]\right)\downarrow \end{aligned} \tag{9.2}$$

   - Line 6, Corollary 8.12, Proposition 8.10, Proposition 8.14, and Theorem 8.15:

$$\begin{aligned} &\diamond \Pr\left(\eta^6_{\text{post}} \models [\![\texttt{b}_2]\!]\right)\uparrow_q \\ &\diamond \Pr\left(\eta^6_{\text{post}} \models [\![\texttt{b}_2]\!]\right)\uparrow_p \\ &\quad \Pr\left(\eta^6_{\text{post}} \models [\![\texttt{b}_2]\!]\right)\downarrow_p \\ &\diamond \Pr\left(\eta^6_{\text{pre}}(\texttt{count}) + 1 = \eta^6_{\text{post}}(\texttt{count})\right)\uparrow \\ &\quad \Pr\left(\eta^6_{\text{pre}}(\texttt{count}) + 1 = \eta^6_{\text{post}}(\texttt{count})\right)\downarrow \\ &\diamond \forall * \in \{\texttt{b}, \texttt{b}_1, \texttt{b}_3, \texttt{b}_4\}\colon \Pr\left(\eta^6_{\text{post}} \models [\![*]\!]\right)\uparrow \text{ and } \Pr\left(\eta^6_{\text{post}} \models [\![*]\!]\right)\downarrow \end{aligned} \tag{9.3}$$

   From these results we obtain:

- Corollary 8.16, Theorem 8.19, (9.1), and (9.3):

$$\begin{aligned} &\diamond \text{if } \Pr\left(\eta^{6-11}_{\text{post}} \models [\![\mathtt{b}_2]\!]\right)\uparrow_q, \text{ then } \Pr\left(\eta^{6-11}_{\text{post}} \models [\![\mathtt{b}_3]\!]\right)\uparrow_q \\ &\diamond \text{if } \Pr\left(\eta^{6-11}_{\text{post}} \models [\![\mathtt{b}_2]\!]\right)\uparrow_p, \text{ then } \Pr\left(\eta^{6-11}_{\text{post}} \models [\![\mathtt{b}_3]\!]\right)\uparrow_p \\ &\quad \text{if } \Pr\left(\eta^{6-11}_{\text{post}} \models [\![\mathtt{b}_2]\!]\right)\downarrow_p, \text{ then } \Pr\left(\eta^{6-11}_{\text{post}} \models [\![\mathtt{b}_3]\!]\right)\downarrow_p \end{aligned} \tag{9.4}$$

- Corollary 8.16, Theorem 8.19, (9.1), (9.2), and (9.4):

$$\begin{aligned} &\diamond \text{if } \Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}_1]\!]\right)\uparrow_p, \text{ then } \Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}_3]\!]\right)\uparrow_p \\ &\diamond \text{if } \Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}_1]\!]\right)\uparrow_q, \text{ then } \Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}_3]\!]\right)\uparrow_q \end{aligned} \tag{9.5}$$

- By Observation 4, Proposition 8.6 and Proposition 8.8 we obtain

$$\text{if } \Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}_3]\!]\right)\uparrow, \text{ then } \Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}]\!]\right)\downarrow \tag{9.6}$$

By combining (9.1)-(9.6) we obtain through Theorem 8.19 for the body of the while loop: $\Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}]\!]\right)\downarrow$.

- Similar to (9.4) and (9.5), we obtain:

$$\begin{aligned} \text{if } \Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}_1]\!]\right)\uparrow \text{ and } \Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}_2]\!]\right)\uparrow \\ \text{then } \Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}_4]\!]\right)(\eta)\uparrow \end{aligned} \tag{9.7}$$

- By Observation 3 and Proposition 8.8, we obtain:

$$\begin{aligned} &\Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}_4]\!]\right)(\eta)\uparrow \\ &\qquad \Longleftrightarrow \Pr\left(\eta^{4-13}_{\text{pre}}(\mathtt{count}) + 1 = \eta^{4-13}_{\text{post}}(\mathtt{count})\right)\downarrow \end{aligned} \tag{9.8}$$

By combining (9.1)-(9.3), and (9.7) and (9.8) we obtain through Theorem 8.19 for the body of the while loop:

$$\Pr\left(\eta^{4-13}_{\text{pre}}(\mathtt{count}) + 1 = \eta^{4-13}_{\text{post}}(\mathtt{count})\right)(\eta)\downarrow$$

Let $\Pr\left(\eta^{4-13}_{\text{post}} \models [\![\mathtt{b}]\!]\right) = f$ and $\Pr\left(\eta^{4-13}_{\text{pre}}(\mathtt{count}) + 1 = \eta^{4-13}_{\text{post}}(\mathtt{count})\right) = g$. By Theorem 8.17 Case 3, it follows:

$$\Pr\left(\eta^{3-13}_{\text{post}} \models [\![\mathtt{b}]\!]\right)\downarrow \tag{2}$$

3. Statement on Line 14
   We obtain from Observation 4 and Corollary 8.16:

$$\Pr\left(\eta^{14}_{\text{post}} \models [\![\neg\mathtt{b}]\!]\right) = sol_{\text{BRP}}(\eta)\uparrow \tag{3}$$

Combining (1) and (2), we obtain through Theorem 8.19: $\Pr\left(\eta^{1-13}_{\text{post}} \models [\![\mathtt{b}]\!]\right)\downarrow$.

By Proposition 8.8, we obtain for Lines 1-13:

$$\Pr\left(\eta^{1-13}_{\text{post}} \models [\![\neg\mathtt{b}]\!]\right)\uparrow \tag{9.9}$$

By Theorem 8.19 and (3) and (9.9) we obtain: $sol_{\text{BRP}}(\eta)\uparrow$.

```
1  free := 1 [q] free := 0;
2  count := 0;
3  while count < MAX ∧ free = 0 do
4      answerReceived := 0 [p] answerReceived := 1;
5      if answerReceived = 1 then
6          free := 1 [q] free := 0;
7          count := 0;
8      else
9          count := count + 1;
10 state := free = 1;
11 return;
```

Listing 9.2: Zeroconf in `pWhile`

9.2 Zeroconf

Zeroconf [24] allows the installation and operation of a network. When a new host joins the network, it randomly selects an address. There are K possible addresses, so the possibility of a collision in a network in which m hosts already joined the network is $q(= \frac{m}{K})$. To make sure the new host uses a free address the following protocol is used to detect a collision:

1. The host picks randomly an address and asks the other host if this address is in use:
   - With probability q this is a free address
   - With probability $1 - q$ a collision occurs
2. If a collision occurs the host tries to detect this by waiting for an answer:
   - With probability p the host get no answer, in which he repeats his question and goes to step 2.
   - With probability $1 - p$ the host get the answer that the address is in use and restarts from step 1.

If after `MAX` tries the host got no answer, then the host will erroneously consider the chosen address as free. Figure 9.2 on the next page shows the associated pMC for `MAX` $= 2$.

The `pWhile` program of Zeroconf is denoted in Listing 9.2.

Observations. As for BRP, we make some observations:

1. The only parametric changes occur at Lines 1, 4 and 6.
2. `free` is either `0` or `1`.

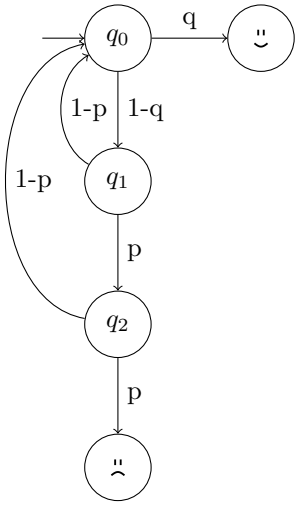


Figure 9.2: pMC of Zeroconf with `MAX` = 2

Notation. Let sol_{Zeroconf} denote the probability of eventually reaching ☺ in the associated pMC Zeroconf, and we let:

- `b` be given by `free = 0`.

Results. We obtain the following for Line 1 by Propositions 8.4, and 8.10, Theorem 8.15:

- $\Pr\left(\eta^1_{\text{post}} \models [\![\mathtt{b}]\!]\right)\uparrow_p$, $\Pr\left(\eta^1_{\text{post}} \models [\![\mathtt{b}]\!]\right)\downarrow_p$ and $\Pr\left(\eta^1_{\text{post}} \models [\![\mathtt{b}]\!]\right)\uparrow_q$

All other results are deduced in a similar manner as for BRP. However, for parameter q, we use Theorem 8.17 Case 1 instead of Theorem 8.17 Case 3. Finally, we obtain $sol_{\text{Zeroconf}}\downarrow_p$ and $sol_{\text{Zeroconf}}\uparrow_q$.

9.3 Load-unload

In the Load-unload case study, the first goal is to fully load a program, the second goal is to totally unload it after it was fully loaded. In the first step, the probability to load is given by p and to unload by $1-p$, in the second step this is denoted by q and $1-q$, respectively. The `pWhile` program of Load-unload is denoted in Listing 9.3 on the next page we observe that there are two while loops, in which the parametric state changes occur.

Notation. Let $sol_{\text{Load-unload}}$ denote the probability of eventually reaching ☺ in the associated pMC of Load-unload.

We obtain $sol_{\text{Load-unload}}\uparrow_p$ and $sol_{\text{Load-unload}}\downarrow_q$ is a similar manner as for BRP and Zeroconf.

```
1  loaded:=0;
2  counter:=1;
3  while counter < k ∧ loaded < N do
4      counter := counter + 1;
5      loaded := loaded + 1 [p] loaded := loaded - 1;
6      if loaded < 0 then
7          loaded := 0;
8      else
9          skip;
10 if  loaded < N then
11     full = 0;
12 else
13     full = 1;
14 while counter < k ∧ loaded > 0 do
15     counter := counter + 1;
16     loaded := loaded + 1 [q] loaded := loaded - 1;
17     if loaded > N then
18         loaded := N;
19     else
20         skip;
21 state := loaded = 0 ∧ full = 1;
22 return;
```

Listing 9.3: Load and unload in at most k steps

9.4 Grids

In the Grids case study there is a $N \times M$ grid. Let $(0, 0)$ be the downleft corner of the grid, and let p_u, p_r, p_d and p_l be the probabilities to move upwards, right, downwards and left, respectively. Clearly, the sum of these probabilities must be 1. In `pWhile` programs on Grids the initial position is given by (x, y).

Assumptions. We assume the grid is bounded by walls, therefore falling off the grid is not possible. When we are at the border of the grid, and make a movement out of the grid, we remain in our original position. Furthermore, we assume:

- the initial value of x is `X`,
- the initial value of y is `Y`,
- $p_r + p_u = q$,
- $p_l + p_d = 1 - q$, and
- $\frac{p_r}{p_r+p_u} = \frac{p_l}{p_l+p_d} = r$.

We define two possible goals for a `pWhile` program on Grids.

1. Reach (a, b) in at most k steps
2. Probability of reaching (a, b) before reaching (c, d)

For both goals we show how monotonicity is obtained from the structure of the `pWhile` program.

9.4.1 *Reach a goal in at most k steps*

The goal of the program is to reach a given goal state (a, b) in at most k steps. Listing 9.4 shows the associated `pWhile` program.

Results. We observe that, `x` is bounded on $[0, N]$ and `y` is bounded on $[0, M]$. Analogous to BRP and Zeroconf we obtain:

- If (a, b) is given by (N, M), then $sol{\uparrow_q}$.
- If (a, b) is given by $(0, 0)$, then $sol{\downarrow_q}$.

If we need to move in both the `x` and the `y` direction to reach a goal state, then we obtain by analyzing the rational function for Lines 4-24: $\Pr\left(\eta_{\text{post}}^{4-24} \models [\![\mathtt{b}]\!]\right)\text{✗}_r$. Furthermore,

- $\Pr\left(\eta_{\text{post}}^{1-3} \models [\![\mathtt{b}]\!]\right) = \begin{cases} 0 & \text{if } \eta_{\text{post}}^{1-2} \not\models [\![\mathtt{b}]\!] \\ 1 & \text{otherwise} \end{cases}$, and

```
1  x := X; // initial x
2  y := Y; // initial y
3  counter := 0;
4  while counter <= k ∧ x = x_goal ∧ y = y_goal do
5      counter := counter + 1;
6      change_x := 1 [r] change_x := 0;
7      if change_x = 1 then
8          x := x + 1 [q] x := x - 1;
9          if x < 0 then
10             x := 0;
11         else
12             if x > N then
13                 x := N;
14             else
15                 skip;
16     else
17         y := y + 1 [q] y := y - 1;
18         if y < 0 then
19             y := 0;
20         else
21             if y > M then
22                 y := M;
23             else
24                 skip;
25 state := x = x_goal ∧ y = y_goal;
26 return;
```

Listing 9.4: Grid in which the goal is to reach a given state in at most k steps.

- $\Pr\left(\eta_{\text{post}}^{25-26} \models [\![\mathtt{b}]\!]\right) = \begin{cases} 0 & \text{if } \eta_{\text{post}}^{1-2} \not\models [\![\mathtt{b}]\!] \\ 1 & \text{otherwise} \end{cases}$

By Theorem 8.20, we obtain $sol\text{✗}_r$.

Remark. When the goal state is not on either (N, M) or $(0, 0)$, we cannot obtain results from the theorems of Chapter 8. If the goal is either $(0, M)$ or $(N, 0)$, then we obtain $sol\text{✗}_q$.

9.4.2 *Probability of reaching good before reaching bad*

The goal of the program is to reach a good state (a, b), before reaching a bad state (c, d). We will only consider the case in which the good state is (N, M). Listing 9.5 shows the associated `pWhile` program.

Results. By analyzing the rational function we obtain that if either $c < \mathtt{X} \leq a$ or $d < \mathtt{Y} \leq b$ does not hold, then the probability of eventually reaching (a, b) ✗_q. Note that there is no counter in the while loop. This is equivalent to a while loop in which $\eta(counter) < \mathtt{MAX}$ and counter remains the same in all iterations of the while loop ($\eta_i(counter) = \eta_{i-1}(counter)$).

In the same manner as for BRP and Zeroconf, we obtain that, if the good state (a, b) is given by (N, M), and $c < \mathtt{X} \leq a$, and $d < \mathtt{Y} \leq b$, then $sol{\uparrow}_q$. Note that we use Theorem 8.18 instead of Theorem 8.17.

In a similar manner as in Section 9.4.1, we obtain that if we need to move in both the `x` and the `y` direction to reach a goal state, then $sol\text{✗}_r$.

```
1  x := X; // initial x
2  y := Y; // initial y
3
4  if x = a ∧ y = b then
5      finish := 1;
6  if x = c ∧ y = d then
7      finish := 1;
8  else
9      finish := 0;
10
11 while finish = 0 do
12     change_x := 1 [r] change_x := 0;
13     if change_x = 1 then
14         x := x + 1 [q] x := x - 1;
15         if x < 0 then
16             x := 0;
17         else
18             if x > N then
19                 x := N;
20             else
21                 skip;
22     else
23         y := y + 1 [q] y := y - 1;
24         if y < 0 then
25             y := 0;
26         else
27             if y > M then
28                 y := M;
29             else
30                 skip;
31     if x = a ∧ y = b then
32         finish := 1;
33
34     if x = c ∧ y = d then
35         finish := 1;
36     else
37         finish := 0;
38
39 state := x = c ∧ y = d;
40 return;
```

Listing 9.5: Grid in which the goal is to reach a good state before a bad state.

9.5 Crowds

The goal of the Crowds protocol [3] is to protect the anonymity of users. There is an initial sender i, and a final receiver r. Furthermore, there are N honest members and M bad members. When i is seen `MAX` times by the same bad member, i is not anonymous anymore. To protect the anonymity of sender i the Crowds protocol works in the following manner:

1. i chooses a random Crowds member c (possibly itself) and sends the message to c
2. c flips a biased coin and handles in the following manner:
   - With probability p, c executes the Crowds protocol.
   - With probability $1 - p$, c directly delivers the message to r.

The protocol is run each time the initial sender wants to establish a connection to a webserver. We let R be the number of times initial sender i wants to connect to the webserver.

The probability of sending the message to a bad member is $q = \frac{M}{N+M}$ which depends on both N and M. We consider q as a free variable. The goal of this protocol is to get R sessions with the webserver without a bad member identifying you.

The `pWhile` program of Crowds is denoted in Listing 9.6, in the program we assume that bad members share their information on identified senders. Therefore, `bad` keeps track of the number of times any `badMember` sees the `initial` sender. Furthermore, we stay in the while loop, as long as both the maximal number of identifications, and the total number of connections is not yet reached.

Notation. Let sol_{Crowds} denote the probability of eventually reaching $\lightning$ in the associated pMC of Crowds, and let:

- $\mathtt{b}_1$ be given by `bad < MAX`
- $\mathtt{b}_2$ be given by `connections < R`
- $\mathtt{b}_3$ be given by `connections >= R`
- $\mathtt{b}_4$ be given by `initial = 1`

9.5.1 *Using Theory of Chapter 8*

Through the theorems of Chapter 8 we cannot find monotonicity in the parameters of Crowds. First of all, $\Pr\left(\eta^{4-14}_{\text{post}} \models [\![\mathtt{b}_4]\!]\right)\uparrow_p$, so $\Pr\left(\eta^{4-14}_{\text{post}} \models [\![\mathtt{b}_1]\!]\right)(\eta)\downarrow_p$. At the same time $\Pr\left(\eta^{4-14}_{\text{post}} \models [\![\mathtt{b}_2]\!]\right)(\eta)\uparrow_p$, so both Theorems 8.17 and 8.18 are not applicable.

```
bad := 0;
connections := 0;
initial := 1;
while bad < MAX ∧ connections < R do
    badMember := 1 [q] badMember := 0;
    if badMember = 1 ∧ initial = 1 then
        bad := bad + 1;

    deliver := 0 [p] deliver := 1;
    if deliver = 1 then
        initial := 1;
        connections := connections + 1;
    else
        initial := 1 [1/(N+M)] initial := 0;
state := bad < MAX ∧ connections >= R;
return;
```

Listing 9.6: Crowds [3] in `pWhile`

For q we obtain in a similar manner as for BRP and Zeroconf:

$$\Pr\left(\eta^{15}_{\text{post}} \models [\![\mathrm{b}_1]\!]\right)\downarrow_q \text{ and } \Pr\left(\eta^{4-14}_{\text{post}} \models [\![\mathrm{b}_2]\!]\right)\downarrow_q$$

However, for Line 15, we cannot obtain monotonicity in q.

9.5.2 *Using Theory of Chapter 6*

As we cannot obtain monotonicity through the theorems of Chapter 8. We try to obtain the monotonicity from the structure of the pMC given goals `connections` $\geq$ `R` and `bad < MAX`. Figure 9.3, shows the sub-pMC of the pMC of Crowds ($\mathcal{M}_{\text{Crowds}}$). $\mathcal{M}_{\text{Crowds}}$ is build from this sub-pMC. Table 9.2 shows the values of the different program variables, $\perp$ implies that `initial` is either 0 or 1.

We observe that:

- If in state s_i `bad` $\geq$ `MAX`, then the s_i equals ☹.
- If in state s_i `bad` $<$ `MAX` and `connections >= R`, then s_i equals ☺.

As long as s_3, s_6 and s_7 do not equal to either ☺ or ☹, the state is replaced by the pMC of Figure 9.3. Eventually, s_6 and s_3 equal ☹ and s_7 equals ☺.

First of all, we look at Pr(◊(`connections >= R`)). Trough Theorem 6.3, we obtain that Pr(◊(`connections >= R`))$\downarrow_p$, Pr(◊`connections >= R`)$\downarrow_q$.

Secondly, we look at Pr(◊(`bad < MAX`)). Trough Theorem 6.3, we obtain that

state	`initial`	`connections`	`bad`
s_0	1	a_1	a_2
s_1	$\bot$	a_1	$a_2 + 1$
s_2	$\bot$	a_1	$a_2 + 1$
s_3	1	a_1	$a_2 + 1$
s_4	$\bot$	a_1	a_2
s_5	$\bot$	a_1	a_2
s_6	1	$a_1 + 1$	$a_2 + 1$
s_7	1	$a_1 + 1$	a_2

Table 9.2: Values of the program variables at different states

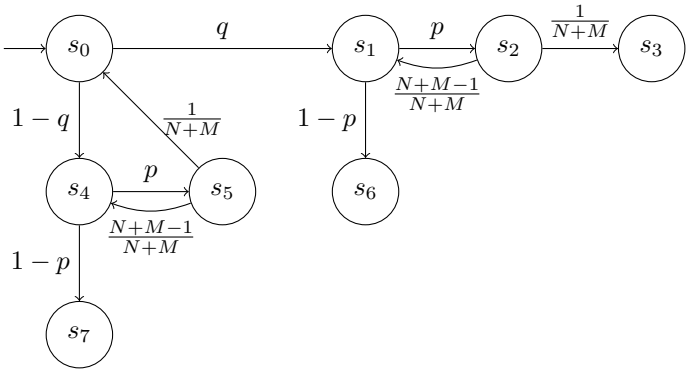


Figure 9.3: Part of the pMC of Crowds [2]

```
bad := 0 [f] bad := MAX;
connections := 0 [g] connections := R;
state := bad < MAX ∧ connections >= R;
return;
```

Listing 9.7: Crowds [3] in `pWhile` after looking at the associated pMC

$\Pr(\Diamond(\texttt{bad < MAX}))\downarrow_p$, $\Pr(\Diamond(\texttt{bad < MAX}))\downarrow_q$.

Let:

- f be $\Pr(\Diamond(\texttt{bad < MAX}))$ in $\mathcal{M}_{\text{Crowds}}$, and
- g be $\Pr(\Diamond(\texttt{connections >= R}))$ in $\mathcal{M}_{\text{Crowds}}$

By construction of $\mathcal{M}_{\text{Crowds}}$ and Corollary 6.4, we obtain $f\downarrow$ and $g\downarrow$. We now obtain the program in Listing 9.7, which is equivalent to the program in Listing 9.6. For this program we obtain $sol_{\text{Crowds}}\downarrow$.

9.5.3 *Adaptation of crowds*

Shmatikov [2] changes Crowds by letting a bad member deliver the message directly, so the second step of the protocol is replaced by:

- if c is an honest member, then c flips a biased coin and then handles in the following manner:
  - With probability p, c executes the protocol.
  - With probability $1-p$, c directly delivers the message to r.
- if c is a bad member, then c directly delivers the message to r.

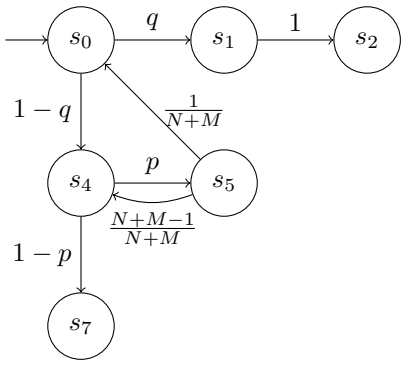


Figure 9.4: Part of the pMC of Crowds [3]

Figure 9.4 shows a part of the pMC of Crowds [2]. The difference between this

figure and Figure 9.3, is that from state s_1 in Figure 9.4 a connection is created directly. In a similar manner as for Crowds [3] we obtain through the theorems of Chapter 6 monotonicity for parameter p and q.

9.6 NAND Multiplexing

The goal of NAND Multiplexing [25] is to construct a reliable computation from unreliable devices. This is done by multiplexing through NAND gates. To transform NAND Multiplexing into an `pWhile` program, we tried to transform its PRISM model. However, we did not succeed as the program becomes too large due to the case distinctions introduced by the NAND gate. Therefore, we have not obtained for which parameters NAND Multiplexing is monotone. Furthermore, the pMC was too complex to analyze manually, as the identification of the different building blocks was not clear.

Part IV

Chapter 10

Conclusion

10.1 Summary

This thesis presents a formal framework to deduce monotonicity of MCs. The framework consists of two layers: the foundation, and the top-layer. The foundation is described in Chapters 5 and 6. In Chapter 5, we describe the mapping from a process in PA to a pMC. We use this mapping to describe pMCs in a more consise manner. In Chapter 6, we provide theorems on monotone structures of pMCs.

The top-layer consists of Chapters 7, 8 and 9. In Chapter 7, we provide the mapping from a `pWhile` program to a process in PA. Chapter 8 provides theorems on monotone substructures in `pWhile`. Finally, Chapter 9 shows the usability of the theorems for `pWhile` by applying them on case studies.

The case studies were originally described in the PRISM language, we rewrote these into `pWhile`. For BRP, Zeroconf, Load-unload, and Grids, we successfully found monotonicity for all monotone parameters. For Crowds [3], we found one monotone parameter by looking at the structure of its `pWhile` program. When calculating the rational function, we could find both monotone parameters. However, for the adaptation of Crowds [2] we could not find monotonicity from the structure of its `pWhile` program. Again, by calculating the rational function, we did obtain monotonicity. For NAND Multiplexing, we could not find monotone parameters, as the transformation of NAND Multiplexing into `pWhile`, yields into a large `pWhile` program, which was not easy to write down.

10.2 Future work

As described above, we could not rewrite all case studies in `pWhile`. Furthermore, we have only shown that we can find monotonicity, but not what the benefit of monotonicity is. Therefore, we see the following directions for future work:

Extend The set of monotone structures in `pWhile` could be extended, such that e.g. monotonicity in Crowds could be obtained.

Model language Most models of case studies are written in the input language of PRISM. The framework could be extended to a broader class of languages.

Automatize Whether or not a model is monotone in some of its parameters is obtained manually. We could try to automatically obtain monotonicity from the input model.

Exploit The monotonicity could be exploit in e.g. parameter lifting in PROPhESY.

Bibliography

[1] R. Dawkins, ***The blind Watchmaker***. W W Norton; American ed. edition, 1986.

[2] V. Shmatikov, "Probabilistic analysis of an anonymity system," ***J. Comput. Secur.***, vol. 12, no. 3,4, pp. 355–377, May 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=1297352.1297359

[3] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," ***ACM Trans. Inf. Syst. Secur.***, vol. 1, no. 1, pp. 66–92, Nov. 1998. [Online]. Available: http://doi.acm.org/10.1145/290163.290168

[4] L. Helmink, M. Sellink, and F. Vaandrager, "Proof-checking a data link protocol," in ***Proc. International Workshop on Types for Proofs and Programs (TYPES'93)***, ser. LNCS, H. Barendregt and T. Nipkow, Eds., vol. 806. Springer, Berlin, Heidelberg, 1994, pp. 127–165.

[5] M. Češka, F. Dannenberg, M. Kwiatkowska, and N. Paoletti, "Precise parameter synthesis for stochastic biochemical systems," in ***Computational Methods in Systems Biology (CMSB)***, ser. LNCS, P. Mendes, J. Dada, and K. Smallbone, Eds., vol. 8859. Springer, Cham, 2014, pp. 86–98.

[6] C. Baier and J.-P. Katoen, ***Principles of Model Checking***. The MIT Press, 2008.

[7] E. Bartocci, R. Grosu, P. Katsaros, C. Ramakrishnan, and S. Smolka, "Model repair for probabilistic systems," in ***International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)***, ser. LNCS, P. Abdulla and K. Leino, Eds., vol. 6605. Springer, Berlin, Heidelberg, 2011, pp. 326–340.

[8] G. Su and D. S. Rosenblum, "Asymptotic bounds for quantitative verification of perturbed probabilistic systems," in ***International Conference on Formal Engineering Methods (ICFEM)***, ser. LNCS, L. Groves and J. Sun, Eds., vol. 8144. Springer, Berlin, Heidelberg, 2013, pp. 297–312.

[9] C. Daws, "Symbolic and parametric model checking of discrete-time markov chains." in ***Theoretical Aspects of Computing (ICTAC)***, ser. LNCS, Z. Liu and K. Araki, Eds., vol. 3407. Springer, Berlin, Heidelberg, 2004, pp. 280–294.

[10] E. M. Hahn, H. Hermanns, and L. Zhang, "Probabilistic reachability for parametric markov models," ***International Journal on Software Tools for Technology Transfer***, vol. 13, no. 1, pp. 3–19, 2011.

[11] T. Quatmann, C. Dehnert, N. Jansen, S. Junges, and J.-P. Katoen, "Parameter synthesis for markov models: Faster than ever," in ***International Symposium on Automated Technology for Verification and Analysis (ATVA)***, ser. LNCS, C. Artho, A. Legay, and D. Peled, Eds., vol. 9938. Springer, Cham, 2016, pp. 50–67.

[12] J. Barnat, L. Brim, A. Krejci, A. Streck, D. Safranek, M. Vejnar, and T. Vejpustek, "On parameter synthesis by parallel model checking," ***IEEE/ACM Transactions on Computational Biology and Bioinformatics***, vol. 9, no. 3, pp. 693–705, May 2012.

[13] E. M. Hahn, Y. Li, S. Schewe, A. Turrini, and L. Zhang, "Iscasmc: A web-based probabilistic model checker," in ***International Symposium on Formal Methods (FM)***, ser. LNCS, C. Jones, P. Pihlajasaari, and J. Sun, Eds., vol. 8442. Springer, Cham, 2014, pp. 312–317.

[14] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen, "The ins and outs of the probabilistic model checker mrmc," ***Perform. Eval.***, vol. 68, no. 2, pp. 90–104, Feb. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.peva.2010.04.001

[15] G. Kant, A. Laarman, J. Meijer, J. van de Pol, S. Blom, and T. van Dijk, "Ltsmin: high-performance language-independent model checking," in ***International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)***, ser. LNCS, C. Baier and C. Tinelli, Eds., vol. 9035. Springer, Berlin, Heidelberg, 2015, pp. 692–707.

[16] M. Timmer, "Scoop: A tool for symbolic optimisations of probabilistic processes," in ***Proceedings of the 2011 Eighth International Conference on Quantitative Evaluation of SysTems***, ser. QEST '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 149–150. [Online]. Available: https://doi.org/10.1109/QEST.2011.27

[17] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," in ***International Conference on Computer Aided Verification (CAV)***, ser. LNCS, R. Majumdar and V. Kunčak, Eds., vol. 10427. Springer, Cham, Jul. 2017, pp. 592–600.

[18] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "Hytech: A model checker for hybrid systems," in ***International Conference on Computer Aided Verification (CAV)***, ser. LNCS, O. Grumberg, Ed. Springer, Berlin, Heidelberg, 1997, pp. 460–463.

[19] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in ***International Conference on Computer Aided Verification (CAV)***, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, Berlin, Heidelberg, 2011, pp. 585–591.

[20] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang, "Param: A model checker for parametric markov models," in ***International Conference on Computer Aided Verification (CAV)***, ser. LNCS, T. Touili, B. Cook, and P. Jackson, Eds., vol. 6174. Springer, Berlin, Heidelberg, 2010, pp. 660–664.

[21] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, and E. Ábrahám, "Prophesy: A probabilistic parameter synthesis tool," in ***International Conference on Computer Aided Verification (CAV)***, ser. LNCS, D. Kroening and C. Păsăreanu, Eds., vol. 9206. Springer, Cham, 2015, pp. 214–231.

[22] N. Jansen, F. Corzilius, M. Volk, R. Wimmer, E. Ábrahám, J.-P. Katoen, and B. Becker, "Accelerating parametric probabilistic verification," in ***International Conference on Quantitative Evaluation of Systems (QEST)***, ser. LNCS, G. Norman and W. Sanders, Eds., vol. 8657. Springer, Cham, 2014, pp. 404–420.

[23] B. Jonsson, W. Yi, and K. G. Larsen, "Probabilistic extensions of process algebras," in ***Handbook of Process Algebra***, J. A. Bergstra, A. Ponse, and S. A. Smolka, Eds. Elsevier, 2001, ch. 11, pp. 685–710.

[24] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston, "Performance analysis of probabilistic timed automata using digital clocks," ***Formal Methods in System Design***, vol. 29, no. 1, pp. 33–78, Jul 2006. [Online]. Available: https://doi.org/10.1007/s10703-006-0005-2

[25] J. Han and P. Jonker, "A system architecture solution for unreliable nanoelectronic devices," ***IEEE Transactions on Nanotechnology***, vol. 1, no. 4, pp. 201–208, Dec 2002.

Appendix: Proof of Lemma 6.9

In this Appendix, we provide the technical proof of Lemma 6.9.

Lemma 6.9
Let $\mathcal{M} = u_f?(l_{if}?\ddot{\smile})^m : \ddot{\frown}$. For any $m \in \mathbb{N}$:

$$f\uparrow_p \Longrightarrow sol_{\mathcal{M}}\uparrow_p \qquad *$$

Proof of Lemma 6.9
For $\mathcal{M}$ in Lemma 6.9 we obtain the following functions:

$$\begin{aligned} sol_{\mathcal{M}}(m) &= 1-(1-f)-f\cdot(f\cdot(1-f))^m \\ &= f - f\cdot(f\cdot(1-f))^m \\ \frac{\partial}{\partial p}sol_{\mathcal{M}}(m) &= \frac{\partial}{\partial p} f\cdot(1-(f\cdot(1-f))^m - f\cdot m\cdot(f(1-f))^{m-1}\cdot(1-2f)) \\ &= \frac{\partial}{\partial p} f\cdot(1-((f\cdot(1-f))^{m-1}(f\cdot(1-f)+f\cdot m\cdot(1-2f))) \end{aligned}$$

We want to prove: $f\uparrow_p \Longrightarrow sol_{\mathcal{M}}\uparrow_p$. We make the following observations:

- $\frac{\partial}{\partial p}sol_{\mathcal{M}} \geq 0 \Longleftrightarrow (f\cdot(1-f))^{m-1}\cdot(f\cdot(1-f)+(1-2f)\cdot m\cdot f) \leq 1$
  We know $\frac{\partial}{\partial p}f \geq 0$. Therefore we obtain:

$$\begin{aligned} &\frac{\partial}{\partial p}sol_{\mathcal{M}} \geq 0 \\ &\qquad \leftrightarrow (1-((f\cdot(1-f))^{m-1}(f\cdot(1-f)+f\cdot m\cdot(1-2f))) \geq 0 \\ &\qquad\qquad \leftrightarrow ((f\cdot(1-f))^{m-1}(f\cdot(1-f)+f\cdot m\cdot(1-2f)) \leq 1 \end{aligned}$$

- **Consider two cases:** $f \in [0,\frac{1}{2}]$ **and** $f \in (\frac{1}{2},1]$
  As $f \in [0,1]$ either $1-2f < 0$ or $1-2f \geq 0$, therefore either $f \in (\frac{1}{2},1]$ or $f \in [0,\frac{1}{2}]$.

By these observations we obtain that for both $f \in [0,\frac{1}{2}]$ and $f \in (\frac{1}{2},1]$ we need to prove:

$$(f\cdot(1-f))^{m-1}\cdot(f\cdot(1-f)+(1-2f)\cdot m\cdot f) \leq 1 \qquad (1)$$

1) $f \in (\frac{1}{2}, 1]$
We want to show Equation (1) holds. We know:

- $f \in (\frac{1}{2}, 1]$ so $0 \leq f(1-f) < \frac{1}{4}$,
- $(1-2f) < 0$, and
- $m \geq 0$.

Therefore:

$$\begin{aligned}(f \cdot (1-f))^{m-1} \cdot (f \cdot (1-f) + (1-2f) \cdot m \cdot f) &\leq (f \cdot (1-f))^{m-1} \cdot f \cdot (1-f) \\ &= (f \cdot (1-f))^{m}\end{aligned}$$

Furthermore, $0 \leq f(1-f) < \frac{1}{4}$, we obtain:

$$(f \cdot (1-f))^{m} \leq \frac{1}{4}^{m} \leq 1.$$

Thus for any $m \in \mathbb{N}$, Equation (1) holds.

2) $f \in [0, \frac{1}{2}]$
We want to show Equation (1) holds. The proof is by induction on the number m of nested l_{if}.

- $m = 0$
$$(f \cdot (1-f))^{0-1} \cdot (f \cdot (1-f) + (1-2f) \cdot 0 \cdot f) = 1$$
- $m = 1$
$$\begin{aligned}&(f \cdot (1-f))^{1-1} \cdot (f \cdot (1-f) + (1-2f) \cdot 1 \cdot f) \\ &\qquad = f \cdot ((1-f) + (1-2f)) < 1 \text{ as } f \in [0, \frac{1}{2}]\end{aligned}$$
- $m > 1$
Recall we need to show:
$$(f \cdot (1-f))^{m-1} \cdot (f \cdot (1-f) + (1-2f) \cdot m \cdot f) \leq 1$$
We assume that the lemma is true for some fixed $m = k \geq 1$ $sol_{\mathcal{M}}{\uparrow_p}$, so
$$(f \cdot (1-f))^{k-1} \cdot (f \cdot (1-f) + (1-2f) \cdot k \cdot f) \leq 1 \qquad \text{(IH)}$$
We want to show for $m = k+1$: $sol_{\mathcal{M}}{\uparrow_p}$, so:
$$(f \cdot (1-f))^{k} \cdot (f \cdot (1-f) + (1-2f) \cdot (k+1) \cdot f) \leq 1$$
Because of (IH) it is sufficient to show:
$$\begin{aligned}&(f \cdot (1-f))^{k} \cdot (f \cdot (1-f) + (1-2f) \cdot (k+1) \cdot f) \\ &\qquad \leq (f \cdot (1-f))^{k-1} \cdot (f \cdot (1-f) + (1-2f) \cdot k \cdot f)\end{aligned}$$
Dividing both sides by $(f \cdot (1-f))^{k-1}$ gives:
$$\begin{aligned}&(f \cdot (1-f)) \cdot (f \cdot (1-f) + (1-2f) \cdot (k+1) \cdot f) \\ &\qquad \leq f \cdot (1-f) + (1-2f) \cdot k \cdot f\end{aligned}$$

Dividing both sides by f gives:

$$f \cdot (1-f) \cdot ((1-f) + (1-2f) \cdot (k+1)) \leq (1-f) + (1-2f) \cdot k$$

As $f \in [0, \frac{1}{2}]$, $f \cdot (1-f)$ is at most $\frac{1}{4}$:

$$\frac{1}{4} \cdot ((1-f) + (1-2f) \cdot (k+1)) \leq (1-f) + (1-2f) \cdot k$$

Clearly, $\frac{1}{4} \cdot (1-f) \leq (1-f)$ when $f \in [0, \frac{1}{2}]$. Therefore, it suffices to show:

$$\frac{1}{4} \cdot (1-2f) \cdot (k+1) \leq (1-2f) \cdot k$$

$$\frac{1}{4} \cdot (k+1) \leq k, \text{ since } (1-2f) \geq 0$$

$$\frac{1}{4} \leq \frac{3}{4}k$$

Which holds as $k \geq 1$.

We obtain for $f \in [0, \frac{1}{2}]$:

- For $m = 0$ Lemma 6.9 holds.
- For $m = 1$ and $m = k$ and $m = k + 1$ Lemma 6.9 holds, so by induction on m, Lemma 6.9 holds for any $m \in \mathbb{N}^+$.

So for $f \in [0, \frac{1}{2}$ Lemma 6.9 holds for any $m \in \mathbb{N}$.

By Cases 1 and 2 we obtain Lemma 6.9 holds for any $m \in \mathbb{N}$. $*$