Saarland University
Faculty of Natural Sciences and Technology VI
Department of Computer Science

Master Thesis

Scheduler-Quantified Time-Bounded Reachability for Distributed Input/Output Interactive Probabilistic Chains

submitted by

Georgel Ionuţ Călin

Supervisor

Prof. Dr-Ing. Holger Hermanns

Advisor

Pepijn Crouzen, MSc.

Reviewers

Prof. Dr-Ing. Holger Hermanns

Prof. Bernd Finkbeiner, PhD

Acknowledgements

Many people have helped getting this thesis to its present state. I sincerely thank them all.

First of all, I would like to thank professor Holger Hermanns for bringing forth such an interesting topic of investigation. I am also grateful to professor Bernd Finkbeiner for the helpful recent discussions we've had and for accepting to be a reviewer of my thesis.

I am especially thankful to Pepijn Crouzen, my advisor, for his always competent and timely suggestions and remarks, as well as for his calm straightforwardness, which helped greatly to keep my work on track. Thank you very much!

Without the help of Ernst Moritz Hahn, Pedro D'Argenio and Lijun Zhang, much of the thesis content would not be as clear as one may find it now. You all have my gratitude!

Furthermore, I would like to thank my colleagues and friends, and especially to Raphael Reischuk, Markus Rabe and Mihai Grigore for reading parts of my thesis and offering helpful advice which I truly value.

Last, but certainly first in my heart, my deepest gratitude goes to my parents and elder sisters for their continued support during my studies. Thank you!

Contents

Chapter 0

Introduction

This thesis considers the computation of extremal reachability probabilities for compositional models that present both probabilistic and nondeterministic behavior. Such models arise, for instance, in the field of distributed algorithms, where probabilistic behavior is used to break symmetries in the system. Nondeterminism may appear due to the uncertain order of events occurring in separate processes or due to unspecified and some times unknown behavior.

Various examples of safety-critical and verifiable systems deployed nowadays can be abstracted through systems which present both nondeterministic and probabilistic behavior. Especially in automotive and aircraft control related settings (but not only) they are of high importance. Attempting to improve on the verification of such distributed systems will continue to be one of the cornerstones yet to be overcome in model checking.

Traditional analysis techniques for probabilistic models with nondeterminism compute the maximal and minimal probability to reach a set of configurations by considering all possible resolutions of the nondeterminism [1]. It has been shown that this approach may lead to unrealistic results for models of distributed systems or algorithms [2]. Briefly, the issue is that the traditional approach allows processes to use non-local information to influence their decisions. To avoid this problem, and guarantee an extended separation of local information between components, using a new type of schedulers has been proposed [2].

Distributed schedulers restrict the resolution of nondeterminism by assuring that local decisions of the processes are based solely on local knowledge. *Strongly distributed schedulers*, additionally, ensure that the relative probability of choosing between two different components does not change with time, provided these components remain idle and uninformed of the progress of the rest of the system.

When considering distributed (or strongly distributed) schedulers, bounds for reachability probabilities are both undecidable and unapproximable in general [3].

However *time-bounded* reachability probabilities, i.e., the probability to reach a set of configurations within a specified time-period, can be computed.

For distributed schedulers, this is due to the fact that optimal solutions in this setting can be computed by only taking into account the subset of deterministic distributed schedulers, which is finite if the system under consideration is finite and acyclic. The theoretical complexity of the method presented is exponential in the number of states and the given time bound.

The case of strongly distributed schedulers turns out to be more difficult. In this setting, optimal solutions may lie on pure probabilistic schedulers [2]. Therefore, exploring all possible solutions is not an option.

In this thesis, it is proposed to reduce the problem of computing time-bounded reachability probabilities for distributed, probabilistic, and nondeterministic models, under distributed (or strongly distributed) schedulers to a nonlinear optimization problem. As modeling vehicle, the formalism of *input/output interactive probabilistic chains* (I/O-IPCs) is used.

The computation of time-bounded reachability probabilities is achieved by reformulating the models as parametric Markov chains, where the parameters are the decisions of the schedulers and the distributed model is unrolled up to the specified time-point. The time-bounded reachability probability can therefore be expressed as a polynomial function and numerical bounds can be computed for it by optimizing the function under certain constraints – as indicated by the involved scheduler.

For distributed schedulers, the only restriction on the variables of the polynomials is that, appropriately grouped, they form a distribution (i.e. all variables take values between 0 and 1 and each group of variables sum up to 1). The case of strongly distributed schedulers, however, requires some additional and more complex restrictions, the optimal value of the property being calculated through more involved nonlinear programming techniques.

0.1 Structure of the Thesis

This thesis describes the means for computing extremal time-bounded reachability probabilities for distributed I/O-IPCs, a new formalism in the spirit of probabilistic input/output timed automata (PIOTA) [4, 5]. The thesis is structured as follows:

- Chapter 1 provides an introduction to interactive probabilistic chains and describes their restriction to input/output interactive probabilistic chains. Subsequently, how I/O-IPCs are parallelized, what are vanishing/tangible states and I/O-IPC paths is specified.

- Chapter 2 presents the adaptation of distributed and strongly distributed schedulers – initially introduced for PIOTA – to the I/O-IPC settings. The schedulers are introduced in a bottom-up manner, starting with schedulers for the components of a distributed I/O-IPC and continuing with distributed and strongly distributed schedulers. Last, the induced probability measure for scheduled distributed I/O-IPCs is presented.

- Chapter 3 introduces parametric Markov chains (PMCs) and describes how distributed I/O-IPCs, arbitrarily scheduled, can be interpreted as PMCs by unfolding up to a given time-bound. It is then proved that time-bounded reachability in a distributed I/O-IPC corresponds to time-unbounded reachability in the associated unfolded PMC.

- Chapter 4 describes the implementation workflow for checking time-bounded reachability of scheduler-quantified distributed I/O-IPCs. The implementation description focuses on the unfolder which produces the unfolded PMC out of a given distributed I/O-IPC.

- Chapter 5 illustrates several case studies that have been performed with our prototype implementation.

- Chapter 6 concludes by presenting remaining open questions as well as possible directions for future research on the topic.

0.2 Contribution

Although various model checkers for verifying reachability of distributed, nondeterministic and probabilistic systems exist, none of them has built-in the means of ruling out unrealistic behaviour due to inadequate scheduling.

The aim of this thesis is to describe an automated method for determining extremal time-bounded reachability probabilities. The method involves a new type of distributed formalism – derived from interactive probabilistic chains – which is subject to (timed) probabilistic and (non-timed) nondeterministic behaviours.

The computation is performed by quantifying over the classes of distributed and strongly distributed schedulers, thus casting out from the start unrealistic results produced by inadequate scheduling freedom.

An early version of this thesis has been accepted for publication at the "17^{th} International SPIN Workshop on Model Checking of Software" as joint work with Pepijn Crouzen, Pedro D'Argenio, Moritz Hahn and Lijun Zhang [6, 7].

Chapter 1

I/O Interactive Probabilistic Chains

Input/output interactive probabilistic chains represent a restricted compositional modeling formalism based on interactive probabilistic chains (IPCs) [8, 9].

The following brief description of interactive probabilistic chains is an adaptation of the information which can be found in [9]. Note that the process algebraic way of defining IPCs will not be used for presenting the I/O-IPC restricted formalism.

1.1 Interactive Probabilistic Chains

Interactive probabilistic chains (IPCs) are state-based models that combine discrete time Markov chains and labelled transition systems [9]. IPCs can be used to compositionally model probabilistic systems. An important feature of IPCs is that probabilistic transitions and action-labeled transitions are handled orthogonally.

For a probabilistic process calculus over a set of actions $\mathcal{A}$ (including internal action τ), assuming that actions are instantaneous and that probabilistic choices take precisely one time step, a *behaviour* is described by the following grammar:

$$B := \delta \mid a; B \mid \sum p::B \mid B[\,]B \mid B[A]B \mid B/A \mid \widetilde{B},$$

where $A \subseteq \mathcal{A} \setminus \{\tau\}$. The used operators are: termination (δ), sequentialization (;), probabilistic choice ($\sum$), nondeterministic choice ([ ]), parallel composition with synchronization set A ($[A]$), hiding of actions ($/$) and process calls ($\tilde{\bullet}$).

A (possibly recursive) process is defined by a rule of the form $\widetilde{B} = B$ and $\mathcal{B}$ is used to denote the set of all behaviours B. The semantics of the formalism can be seen as a probabilistic extension of labeled transition systems (see Figure 1.1).

Let $\rightharpoonup \subseteq X \times Y$ represent a partial function relation from X to Y and let $Dist(X)$ be the set of all probability distributions over X for any finite set X and any set Y.

$$\frac{}{\delta \overset{1}{\Longrightarrow} \delta} \qquad \frac{}{a; B \overset{1}{\Longrightarrow} a; B} \qquad \frac{}{\sum_i p_i :: B_i \overset{p_i}{\Longrightarrow} B_i} \qquad \frac{B_1 \overset{p_1}{\Longrightarrow} B_1' \quad B_2 \overset{p_2}{\Longrightarrow} B_2'}{B_1[\,]B_2 \overset{p_1 p_2}{\Longrightarrow} B_1'[\,]B_2'}$$

$$\frac{B_1 \overset{p_1}{\Longrightarrow} B_1' \quad B_2 \overset{p_2}{\Longrightarrow} B_2'}{B_1[A]B_2 \overset{p_1 p_2}{\Longrightarrow} B_1'[A]B_2'} \qquad \frac{\widetilde{B} = B \quad B \overset{p}{\Longrightarrow} B'}{\widetilde{B} \overset{p}{\Longrightarrow} B'} \qquad \frac{\widetilde{B} = B \quad B \overset{a}{\longrightarrow} B'}{\widetilde{B} \overset{a}{\longrightarrow} B'}$$

$$\frac{B_1 \overset{a}{\longrightarrow} B_1'}{B_1[\,]B_2 \overset{a}{\longrightarrow} B_1'[\,]B_2} \qquad \frac{B_2 \overset{a}{\longrightarrow} B_2'}{B_1[\,]B_2 \overset{a}{\longrightarrow} B_1[\,]B_2'} \qquad \frac{B_1 \overset{a}{\longrightarrow} B_1' \quad B_2 \overset{a}{\longrightarrow} B_2' \quad a \in A}{B_1[A]B_2 \overset{a}{\longrightarrow} B_1'[A]B_2'}$$

$$\frac{B_1 \overset{a}{\longrightarrow} B_1' \quad a \notin A}{B_1[A]B_2 \overset{a}{\longrightarrow} B_1'[A]B_2} \qquad \frac{B_2 \overset{a}{\longrightarrow} B_2' \quad a \notin A}{B_1[A]B_2 \overset{a}{\longrightarrow} B_1[A]B_2'} \qquad \frac{}{a; B \overset{a}{\longrightarrow} B}$$

$$\frac{B \overset{p}{\Longrightarrow} B'}{B/A \overset{p}{\Longrightarrow} B'/A} \qquad \frac{B \overset{a}{\longrightarrow} B' \quad a \notin A}{B/A \overset{a}{\longrightarrow} B'/A} \qquad \frac{B \overset{a}{\longrightarrow} B' \quad a \in A}{B/A \overset{\tau}{\longrightarrow} B'/A}$$

Figure 1.1: Operational Semantics of the IPC Modeling Language

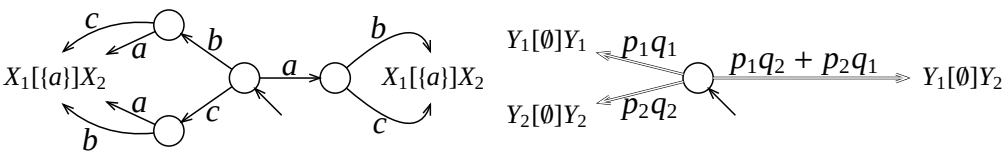
Definition 1. *An IPC is a quintuple $\mathcal{P} = \langle S, \mathcal{A}, \rightarrow, \Rightarrow, \hat{s} \rangle$, where: S is a finite set of states with $\hat{s} \in S$ the initial state, $\mathcal{A}$ is a finite set of actions including the internal action τ, $\rightarrow \subseteq S \times \mathcal{A} \times S$ is the set of interactive transitions and $\Rightarrow : S \rightharpoonup Dist(S)$ is the set of probabilistic transitions.*

Definition 2. *The operational semantics of a behaviour $\widehat{B}$ over $\mathcal{A}$ is defined as the IPC $\mathcal{P} = \langle \mathcal{B}, \mathcal{A}, \rightarrow, \Rightarrow, \widehat{B} \rangle$ with $\rightarrow$ and $\Rightarrow$ as given by the rules of Figure 1.1.*

The operators' binding order – from the strongest-binding one to the lowest-binding one – is the following: "$\tilde{\bullet}$" $\succ$ "/" $\succ$ "[]" $\approx$ "[•]" $\succ$ ";" $\succ$ "$\sum$".

The second rule of Figure 1.1 enriches the language with the *arbitrary waiting* property by assuring that time may advance even while part of the process is blocked, waiting for a synchronization to happen. As argued in [9], these rules inspired by Hansson [10] ensure that time may always advance synchronously.

Example 1. *Synchronization of nondeterministic and probabilistic behaviours.*



"(a[ ]b; X_1)[{a}](a[ ]c; X_2)" "(p_1::Y_1 + p_2::Y_2)[∅](q_1::Y_1 + q_2::Y_2)"

The formalism is also subject to the *maximal progress* assumption [11]: a process cannot delay an internal transition – if a choice exists between a probabilistic and an internal transition, the internal transition will have precedence. Although not integrated into the semantics, the maximal progress assumption can be taken care of through bisimulation equivalences [9].

For I/O-IPCs, the specific variant of the maximal progress assumption is handled through the induced path measure (see Definition 12).

1.2 Input/Output Interactive Probabilistic Chains

I/O-IPCs, a restricted variant of IPCs with a strict separation of local and non-local behavior, are used as modeling formalism in the present study. The restriction of IPCs to I/O-IPCs follows the one of interactive Markov chains (IMCs) to I/O-IMCs in the continuous-time setting [12].

The separation between local and non-local behavior is achieved by partitioning the I/O-IPC actions in *input*, *output*, and *internal* actions. Let $\uplus$ denote the "disjoint set" union. As we will not make use of the process algebra approach in the I/O-IPC setting, I/O-IPCs can be described in a more intuitive way as follows.

Definition 3. *A **basic** I/O-IPC $\mathcal{P}$ is a quintuple $\langle S, \mathcal{A}, \rightarrow_{\mathcal{P}}, \Rightarrow_{\mathcal{P}}, \hat{s}\rangle$, where: S is a finite set of states with $\hat{s} \in S$ the initial state, $\mathcal{A} = \mathcal{A}^I \uplus \mathcal{A}^O \uplus \mathcal{A}^{int}$ is a finite set of actions, $\rightarrow \subseteq S \times \mathcal{A} \times S$ is the set of interactive transitions and $\Rightarrow : S \rightharpoonup Dist(S)$ represents the set of probabilistic transitions.*

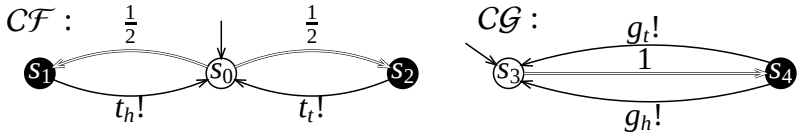
Input actions are suffixed by "?", output actions by "!" and we require that an I/O-IPC $\mathcal{P}$ is both input-enabled, i.e. for each state s and each input action a there is at least one state s' such that $(s, a, s') \in \rightarrow_{\mathcal{P}}$. We also require that the I/O-IPC is *action-deterministic*, that is, for each state s and each action a there is at most one state s' such that $(s, a, s') \in \rightarrow_{\mathcal{P}}$. The nondeterminism then stems from the choice between different actions. Finally we require that every state has at least one outgoing, internal, output, or probabilistic transition.

We say that an I/O-IPC is *closed* if it has no input actions, i.e., $A^I = \emptyset$. Note that the requirement of action-determinism is introduced only to simplify the theoretical framework around schedulers. Nondeterministic choices between input transitions can be handled in a similar way as nondeterministic choices between output or internal transitions [2].

Given an action a, we use the shorthand notation $s \xrightarrow{a}_{\mathcal{P}} s'$ for an interactive transition $(s, a, s') \in \rightarrow_{\mathcal{P}}$ of $\mathcal{P}$. Given a distribution μ over the states of $\mathcal{P}$ we use the shorthand notation $s \Rightarrow_{\mathcal{P}} \mu$ for $(s, \mu) \in \Rightarrow_{\mathcal{P}}$. We often leave out the subscript when it is clear from the context.

As a running example a simple repeating "coin flip & guess" experiment is used: one player repeatedly flips a coin, while a second player (nondeterministically) guesses the outcome (see Figure 1.2). We are interested in the probability that the second player guesses correctly at least once within t rounds.

Although this probability is $1 - \left(\frac{1}{2}\right)^t$, it has been shown that standard analysis methods produce a probability of 1 for any $t > 0$ [2]. The issue is that, from a *global* point of view, the optimal resolution of the nondeterministic guess uses the outcome of the flip as a guide and therefore knows which is the correct guess.



The coin-flip is depicted on the left-hand side and the coin-guess on the right-hand side. Initial states are indicated by arrows; interactive transitions are labelled with their actions and probabilistic transitions $s \Rightarrow \mu$ are depicted by arrows from s to the support of μ, where each arrow is labeled with the associated probability.

Figure 1.2: Basic I/O-IPC Models of the Repeated Coin-Flip Experiment

1.2.1 Parallel Composition

Distributed I/O-IPCs are obtained through parallelizing ($\|$) simpler I/O-IPCs.

Definition 4. *Two I/O-IPCs $\mathcal{P}$ and $\mathcal{Q}$ are composable if $\mathcal{A}^O_\mathcal{P} \cap \mathcal{A}^O_\mathcal{Q} = \mathcal{A}_\mathcal{P} \cap \mathcal{A}^{int}_\mathcal{Q} = \mathcal{A}^{int}_\mathcal{P} \cap \mathcal{A}_\mathcal{Q} = \emptyset$. If $\mathcal{P}$ and $\mathcal{Q}$ are composable then $C := \mathcal{P}\|\mathcal{Q}$ will be*

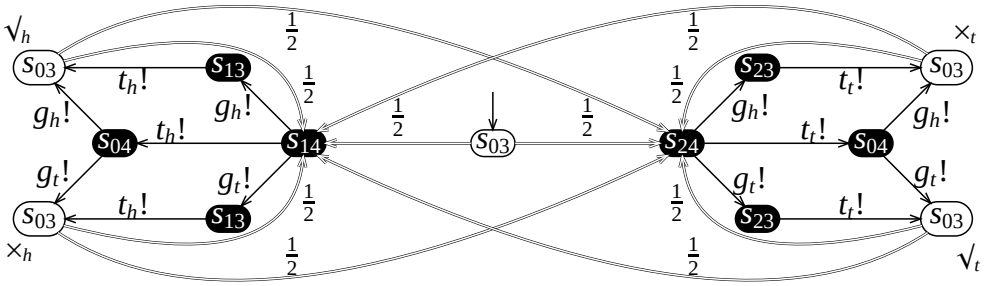
$$\langle S_\mathcal{P} \times S_\mathcal{Q}, \mathcal{A}^I_C \uplus \mathcal{A}^O_C \uplus \mathcal{A}^{int}_C, \rightarrow_C, \Rightarrow_C, (\hat{s}_\mathcal{P}, \hat{s}_\mathcal{Q})\rangle,$$

where $\mathcal{A}^O_C := \mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q}$, $\mathcal{A}^I_C := \left(\mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}\right) \setminus \mathcal{A}^O_C$, $\mathcal{A}^{int}_C = \mathcal{A}^{int}_\mathcal{P} \cup \mathcal{A}^{int}_\mathcal{Q}$ and the transition relations are

$$\begin{aligned}
\rightarrow_C = {} & \{\langle s,t\rangle \xrightarrow{a}_C \langle s',t\rangle \mid s \xrightarrow{a}_\mathcal{P} s',\ a \in \mathcal{A}_\mathcal{P} \setminus \mathcal{A}_\mathcal{Q}\} \\
& \cup \{\langle s,t\rangle \xrightarrow{a}_C \langle s,t'\rangle \mid t \xrightarrow{a}_\mathcal{Q} t',\ a \in \mathcal{A}_\mathcal{Q} \setminus \mathcal{A}_\mathcal{P}\} \\
& \cup \{\langle s,t\rangle \xrightarrow{a}_C \langle s',t'\rangle \mid s \xrightarrow{a}_\mathcal{P} s',\ t \xrightarrow{a}_\mathcal{Q} t',\ a \in \mathcal{A}_\mathcal{P} \cap \mathcal{A}_\mathcal{Q}\} \\
\Rightarrow_C = {} & \{\langle s,t\rangle \Rightarrow_C (\mu_s \times \mu_t) \mid s \Rightarrow_\mathcal{P} \mu_s \ \wedge\ t \Rightarrow_\mathcal{Q} \mu_t\}
\end{aligned}$$

with $\mu_s \times \mu_t$ denoting the product distribution on $S_\mathcal{P} \times S_\mathcal{Q}$. Parallel composition can be extended to any finite set of parallelizable I/O-IPCs in the usual way. Let #C denote the number of components of a distributed I/O-IPC C.

The result of synchronizing an input action with an output action through I/O-IPC parallelization will be an output action in the resulting model. As an example, the composition of the basic I/O-IPCs of Figure 1.2 is depicted in Figure 1.3.



In the labelled states, the two I/O-IPCs in Figure 1.2 distribute over the next states according to their possible combined choices. Otherwise, their actions are interleaved. The shorthand notation s_{ij} is used to describe the distributed state $\langle s_i, s_j \rangle$. The flip matching the guess is represented by the $\surd$-labeled "goal" states.

Figure 1.3: Distributed I/O-IPC Model of the Repeated Coin-Flip Experiment

1.2.2 Vanishing and Tangible States

The use of distinct probabilistic and instantaneous transitions separates the concerns of time and interaction. In essence, it allows us to specify interactions between components which are instantaneous and do not have to be modeled with explicit time steps.

Internal and output transitions are considered to be *immediate* while probabilistic transitions are *timed*. The maximal progress property for IPCs translates in the following way to I/O-IPCs: a process cannot delay an immediate transition – e.g. if given the choice between an immediate and a timed (probabilistic) transition, the immediate transition has precedence. The dissociation present between immediate and timed transitions is also reflected in the system states.

Definition 5 (Vanishing/Tangible States)**.** *A state is called vanishing if at least one outgoing immediate transition is enabled in it. Conversely, if only probabilistic transitions are enabled in a state then it is called tangible.*

In Figures 1.2 and 1.3, the black-colored nodes are vanishing states and the remaining ones are tangible states. For simplicity, in this thesis only non-Zeno models are considered: cycles consisting of only immediate actions are not reachable/present in the analyzed distributed models.

1.2.3 Paths in I/O-IPCs

An I/O-IPC *path* describes one possible *run* of the I/O-IPC. In such a run, we start in a particular state, follow a transition to another state, and so forth.

Definition 6. *A finite path of length $n \in \mathbb{N}$ of an I/O-IPC $\mathcal{P} = \langle S, \mathcal{A}, \rightarrow, \Rightarrow, \hat{s} \rangle$ is a sequence $\sigma := s_0 a_0 \dots a_{n-1} s_n$ of alternating states $s_i \in S$ and actions or distributions $a_i \in \mathcal{A} \cup Dist(S)$. For consecutive s_i, s_{i+1} in σ it must hold that: either $a_i \in \mathcal{A}$ and $s_i \xrightarrow{a_i} s_{i+1}$, or $a_i \in Dist(S)$, s_i is tangible, $s_i \Rightarrow a_i$ and $a_i(s_{i+1}) > 0$.*

The last state of a finite path σ is denoted by $last(\sigma)$. An infinite path of $\mathcal{P}$ is an infinite sequence $s_0 a_0 s_1 a_1 \dots$ of alternating states and actions/distributions.

For studying time-bounded reachability, we need a notion of *time*. We follow the definition of time in IPCs and say that only *probabilistic* transitions take time, while interactive transitions are considered to take place immediately [13].

Definition 7. *The* elapsed time *along a finite path σ – notation $t(\sigma)$ – is defined recursively, for states s, actions a and distributions μ over states:*

$$\mathrm{t}(\sigma) = \begin{cases} 0 & \text{if } \sigma = s \\ t(\sigma') & \text{if } \sigma = \sigma' a s \\ t(\sigma') + 1 & \text{if } \sigma = \sigma' \mu s \end{cases}$$

Example 2. *Let $S_{C\mathcal{F}} = \{s_0, s_1, s_2\}$, $S_{C\mathcal{G}} = \{s_3, s_4\}$ be the state spaces of the I/O-IPCs in Figure 1.2 and $\mu_{C\mathcal{F}} : S_{C\mathcal{F}} \rightharpoonup Dist(S_{C\mathcal{F}})$, $\mu_{C\mathcal{G}} : S_{C\mathcal{G}} \rightharpoonup Dist(S_{C\mathcal{G}})$ describe the next-state probability distributions for $C\mathcal{F}$ and $C\mathcal{G}$.*

Abstracting away the indexing, $\mu(s)$ is the probability distribution for states reachable through $\Rightarrow$ from s and $\mu(s)(s')$ is the probability of reaching s' from s.

Then (e.g.) for the path $\sigma = s_0 \mu_{C\mathcal{F}}(s_0) s_1 t_h s_0 \mu_{C\mathcal{F}}(s_0) s_2 t_t s_0$ of $C\mathcal{F}$ and for the path $\sigma' = s_3 \mu_{C\mathcal{G}}(s_3) s_4 g_t s_3 \mu_{C\mathcal{G}}(s_3) s_4 g_t s_3$ of $C\mathcal{G}$ we have:

$$\begin{aligned} \mathrm{t}(\sigma) &= \mathrm{t}(s_0 \mu_{C\mathcal{F}}(s_0) s_1 t_h s_0 \mu_{C\mathcal{F}}(s_0) s_2) = \mathrm{t}(s_0 \mu_{C\mathcal{F}}(s_0) s_1 t_h s_0) + 1 \\ &= \mathrm{t}(s_0 \mu_{C\mathcal{F}}(s_0) s_1) + 1 = \mathrm{t}(s_0) + 2 = 2 \\ \mathrm{t}(\sigma') &= \mathrm{t}(s_3 \mu_{C\mathcal{G}}(s_3) s_4 g_t s_3 \mu_{C\mathcal{G}}(s_3) s_4) = \mathrm{t}(s_3 \mu_{C\mathcal{G}}(s_3) s_4 g_t s_3) + 1 \\ &= \mathrm{t}(s_3 \mu_{C\mathcal{G}}(s_3) s_4) + 1 = \mathrm{t}(s_3) + 2 = 2 \end{aligned}$$

The (σ, σ')-pair of local paths induces all of the following paths in $C\mathcal{F} \| C\mathcal{G}$:

- $\langle s_0, s_3 \rangle \overset{1/2}{\Rightarrow} \langle s_1, s_4 \rangle \xrightarrow{t_h} \langle s_0, s_4 \rangle \xrightarrow{g_t} \langle s_0, s_3 \rangle \overset{1/2}{\Rightarrow} \langle s_2, s_4 \rangle \xrightarrow{t_t} \langle s_0, s_4 \rangle \xrightarrow{g_t} \langle s_0, s_3 \rangle$
- $\langle s_0, s_3 \rangle \overset{1/2}{\Rightarrow} \langle s_1, s_4 \rangle \xrightarrow{t_h} \langle s_0, s_4 \rangle \xrightarrow{g_t} \langle s_0, s_3 \rangle \overset{1/2}{\Rightarrow} \langle s_2, s_4 \rangle \xrightarrow{g_t} \langle s_1, s_3 \rangle \xrightarrow{t_t} \langle s_0, s_3 \rangle$
- $\langle s_0, s_3 \rangle \overset{1/2}{\Rightarrow} \langle s_1, s_4 \rangle \xrightarrow{g_t} \langle s_1, s_3 \rangle \xrightarrow{t_h} \langle s_0, s_3 \rangle \overset{1/2}{\Rightarrow} \langle s_2, s_4 \rangle \xrightarrow{t_t} \langle s_0, s_4 \rangle \xrightarrow{g_t} \langle s_0, s_3 \rangle$
- $\langle s_0, s_3 \rangle \overset{1/2}{\Rightarrow} \langle s_1, s_4 \rangle \xrightarrow{g_t} \langle s_1, s_3 \rangle \xrightarrow{t_h} \langle s_0, s_3 \rangle \overset{1/2}{\Rightarrow} \langle s_2, s_4 \rangle \xrightarrow{g_t} \langle s_1, s_3 \rangle \xrightarrow{t_t} \langle s_0, s_3 \rangle$

where the value $\mu(s)(s')$ is used to decorate the transition $s \overset{\mu(s)}{\Longrightarrow} s'$ instead of $\mu(s)$.

The highlighting of the switched labels in the above transitions underline that multiple "distributed" paths correspond to the same (σ, σ')-pair.

The latter is due to different interleaving possibilities in $C\mathcal{F} \| C\mathcal{G}$.

Chapter 2

I/O-IPC Nondeterminism Resolution

The probability of reaching a set of goal states in a distributed I/O-IPC depends on how the nondeterminism of chosing an action is handled. By assigning probabilities to the available actions, a scheduler can be seen as an I/O-IPC refinement such that the induced model becomes deterministic. It can thus be said that a scheduler enables us to determine reachability probabilities in a deterministic fashion.

However, the class of all schedulers for the model of a distributed system contains schedulers that are unrealistic in that they allow components of the system to use non-local information to guide their local decisions. To overcome this problem, *distributed* schedulers have been introduced, that restrict the possible choices of a scheduler in a distributed setting [2]. Distributed schedulers, originally introduced for (switched) probabilistic input/output timed automata [1], are adapted here for the input/output interactive Markov chains formalism.

To illustrate the necessity of distributed schedulers we consider the game described in Figure 1.3 where an unbiased coin is repeatedly tossed and guessed by two independent entities at the same time. We are interested in the probability to reach the set of states labelled $\surd$ within a specified number t of timed (probabilistic) steps. This is exactly the probability that the guessing player guesses correctly within at most t tries. Intuitively, for each matching toss/guess, since the tossing player makes its choice probabilistically and the guessing player does not observe the outcome, the guessing player should have a probability of one half to make the right guess and win the game.

However, it is clear that in the composed model there is a scheduler that arrives with probability one at a $\surd$ state within at most one timed step. This scheduler simply chooses the action t_h if heads is tossed and t_t if tails is tossed, thereby always winning. The purpose of distributed schedulers is to ensure that the decision between t_h and t_t is made only based on *local* information.

2.1 Local Schedulers

We have to associate paths of an I/O-IPC with probabilities. The usual way of doing it is by defining the probability of a path as the multiplication of the probabilities of its transitions. To define such a probability for paths in an I/O-IPC we need some way of resolving the nondeterministic choice between interactive transitions in vanishing states of an I/O-IPC.

For all states $s \in S$, let $A^{en}_{s,\mathcal{P}} = \{a \in \mathcal{A}^O | \exists s'.s \xrightarrow{a} s'\} \cup \{a \in \mathcal{A}^{int} | \exists s'.s \xrightarrow{a} s'\}$ be the set of enabled immediate actions for s.

Definition 8. *A function $\eta_{\mathcal{P}} : Paths(\mathcal{P}) \to Dist(\mathcal{A}_{\mathcal{P}})$ is a scheduler for an I/O-IPC $\mathcal{P}$ if positive probabilities are assigned only to immediate actions enabled in the last state of a path: $\forall \sigma \in Paths(\mathcal{P})$, $\eta_{\mathcal{P}}(\sigma)(a) > 0$ implies $a \in A^{en}_{last(\sigma),\mathcal{P}}$.*

If $\mathcal{P}$ is closed, then a scheduler determines the probability to observe a certain path, which also allows us to define time-bounded reachability probabilities. We give the details, in the context of distributed schedulers, in Chapter 3.

Example 3. *Let $S_{C\mathcal{F}} = \{s_0, s_1, s_2\}$, $\mu_{C\mathcal{F}} : S_{C\mathcal{F}} \rightharpoonup Dist(S_{C\mathcal{F}})$ and $S_{C\mathcal{G}} = \{s_3, s_4\}$, $\mu_{C\mathcal{G}} : S_{C\mathcal{G}} \rightharpoonup Dist(S_{C\mathcal{G}})$ as in Figure 1.2, i.e. the non-zero entries of the probabilistic transitions in $C\mathcal{F}$ and $C\mathcal{G}$ are given by:*

$$\mu_{C\mathcal{F}}(s_0)(s_1) = \mu_{C\mathcal{F}}(s_0)(s_2) = \frac{1}{2} \text{ and } \mu_{C\mathcal{G}}(s_3)(s_4) = 1.$$

Further consider the paths $\sigma = s_0\mu_{C\mathcal{F}}(s_0)s_1t_hs_0\mu_{C\mathcal{F}}(s_0)s_2$ of $C\mathcal{F}$, respectively $\sigma' = s_3\mu_{C\mathcal{G}}(s_3)s_4g_ts_3\mu_{C\mathcal{G}}(s_3)s_4$ of $C\mathcal{G}$.

Knowing that $\mathcal{A}^O_{C\mathcal{F}} = \{t_h!, t_t!\}$, respectively $\mathcal{A}^O_{C\mathcal{G}} = \{g_h!, g_t!\}$ and given a pair of schedulers $\eta_{C\mathcal{F}} : Paths(C\mathcal{F}) \to Dist(\mathcal{A}^O_{C\mathcal{G}})$, $\eta_{C\mathcal{G}} : Paths(C\mathcal{G}) \to Dist(\mathcal{A}^O_{C\mathcal{G}})$ of $C\mathcal{F}$ and $C\mathcal{G}$, for σ and σ' above it must hold that:

$$\eta_{C\mathcal{F}}(\sigma)(t_h) + \eta_{C\mathcal{F}}(\sigma)(t_t) = \eta_{C\mathcal{F}}(\sigma)(t_t) = 1 \text{ and } \eta_{C\mathcal{G}}(\sigma')(g_h) + \eta_{C\mathcal{G}}(\sigma')(g_t) = 1.$$

2.2 Distributed Schedulers

The main principle of distributed schedulers is to use a separate scheduler for each of the components of the system such that each has access only to their own scheduling history.

To be able to reason about *local* information we have to first introduce path projections. For any distributed I/O-IPC $C = \mathcal{P}_1 \| \ldots \| \mathcal{P}_n$ and path $\sigma \in Paths(C)$, the projection $\sigma[\mathcal{P}_i]$ of σ on C's i^{th} basic component is given by:

- $(\hat{s}_C)[\mathcal{P}_i] = \pi_i(\hat{s}_C)$

- $(\sigma a s)[\mathcal{P}_i] = \begin{cases} \sigma[\mathcal{P}_i] & \text{if } a \notin \mathcal{A}_{\mathcal{P}_i} \\ (\sigma[\mathcal{P}_i])a(\pi_i(s)) & \text{if } a \in \mathcal{A}_{\mathcal{P}_i} \end{cases}$
- $(\sigma(\mu_1 \times \cdots \times \mu_n)s)[\mathcal{P}_i] = (\sigma[\mathcal{P}_i])\mu_i(\pi_i(s))$.

where $\pi_i(\langle s_1, \ldots, s_n \rangle) = s_i$ for all $\langle s_1, \ldots, s_n \rangle \in S_C$.

A *local* scheduler for $\mathcal{P}$ is simply any scheduler for $\mathcal{P}$ as given by Definition 8. A local scheduler resolves the nondeterminism arising from choices between enabled output and internal actions in one of the components.

However, nondeterminism may also arise from the interleaving of the different components. In other words, if for some state in a distributed I/O-IPC, two or more components have enabled immediate actions, then it must be decided which component acts first. This decision is made by the *interleaving scheduler*.

Definition 9. *A function $\mathcal{I} : Paths(C) \to Dist(\{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ is an interleaving scheduler for the distributed I/O-IPC $C := \mathcal{P}_1 \| \ldots \| \mathcal{P}_n$ if it is defined for paths σ such that $last(\sigma)$ is vanishing and if it chooses probabilistically an enabled component of the distributed system:*

$$\mathcal{I}(\sigma)(\mathcal{P}_i) > 0 \text{ implies } A^{en}_{last(\sigma[\mathcal{P}_i]),\mathcal{P}_i} \neq \emptyset.$$

Example 4. *Let $\eta_{C\mathcal{F}}$, $\eta_{C\mathcal{G}}$ be local schedulers for the coin-flip $C\mathcal{F}$ and the coin-guess $C\mathcal{G}$ of the game described in Figures 1.2 and 1.3.*

Then every function $\mathcal{I} : Paths(C\mathcal{F} \| C\mathcal{G}) \to Dist(\{C\mathcal{F}, C\mathcal{G}\})$ such that for all paths $\sigma \in Paths(C\mathcal{F} \| C\mathcal{G})$:

$$last(\sigma[C\mathcal{F}]) \neq s_0 \wedge last(\sigma[C\mathcal{G}]) = s_4 \Rightarrow \mathcal{I}(\sigma)(C\mathcal{F}) + \mathcal{I}(\sigma)(C\mathcal{G}) = 1$$
$$last(\sigma[C\mathcal{F}]) \neq s_0 \wedge last(\sigma[C\mathcal{G}]) = s_3 \Rightarrow \mathcal{I}(\sigma)(C\mathcal{F}) = 1$$
$$last(\sigma[C\mathcal{F}]) = s_0 \wedge last(\sigma[C\mathcal{G}]) = s_4 \Rightarrow \mathcal{I}(\sigma)(C\mathcal{G}) = 1$$

belongs to the class of interleaving schedulers of $C\mathcal{F} \| C\mathcal{G}$.

Local schedulers and an interleaving scheduler form a *distributed scheduler*.

Definition 10. *A function $\eta_C : Paths(C) \to Dist(A_C)$ is a distributed scheduler for the I/O-IPC $C = \mathcal{P}_1 \| \ldots \| \mathcal{P}_n$ if, given local schedulers $\eta_{\mathcal{P}_1}, \ldots, \eta_{\mathcal{P}_n}$ and interleaving scheduler $\mathcal{I}$, for all $\sigma \in Paths(C)$ with $last(\sigma)$ vanishing and for all $a \in A_C$:*

$$\eta_C(\sigma)(a) = \sum_{i=1}^{n} \mathcal{I}(\sigma)(\mathcal{P}_i) \cdot \eta_{\mathcal{P}_i}(\sigma[\mathcal{P}_i])(a)$$

We denote the set of all distributed schedulers by *DS*.

Example 5. *Let $\eta_{C\mathcal{F}}$, $\eta_{C\mathcal{G}}$ be local schedulers for the coin-flip $C\mathcal{F}$ and coin-guess $C\mathcal{G}$, and $\mathcal{I}$ be an interleaving scheduler for the distributed I/O-IPC $C\mathcal{F}||C\mathcal{G}$.*

Then every function $\eta : Paths(C\mathcal{F}||C\mathcal{G}) \to Dist\,(A_{C\mathcal{F}||C\mathcal{G}})$ belongs to the class of distributed schedulers of $C\mathcal{F}||C\mathcal{G}$ if $\forall\sigma \in Paths(C\mathcal{F}||C\mathcal{G})$ it holds that:

- *if $last(\sigma[C\mathcal{F}]) = s_1 \wedge last(\sigma[C\mathcal{G}]) = s_4$ then*

$$\begin{aligned}\eta(\sigma)(t_h!) &= \mathcal{I}(\sigma)(C\mathcal{F})\\ \eta(\sigma)(g_h!) &= \mathcal{I}(\sigma)(C\mathcal{G}) \cdot \eta_{C\mathcal{G}}(\sigma[C\mathcal{G}])(g_h!)\\ \eta(\sigma)(g_t!) &= \mathcal{I}(\sigma)(C\mathcal{G}) \cdot \eta_{C\mathcal{G}}(\sigma[C\mathcal{G}])(g_t!)\end{aligned}$$

- *if $last(\sigma[C\mathcal{F}]) = s_2 \wedge last(\sigma[C\mathcal{G}]) = s_4$ then*

$$\begin{aligned}\eta(\sigma)(t_t!) &= \mathcal{I}(\sigma)(C\mathcal{F})\\ \eta(\sigma)(g_h!) &= \mathcal{I}(\sigma)(C\mathcal{G}) \cdot \eta_{C\mathcal{G}}(\sigma[C\mathcal{G}])(g_h!)\\ \eta(\sigma)(g_t!) &= \mathcal{I}(\sigma)(C\mathcal{G}) \cdot \eta_{C\mathcal{G}}(\sigma[C\mathcal{G}])(g_t!)\end{aligned}$$

- *if $last(\sigma[C\mathcal{F}]) = s_1 \wedge last(\sigma[C\mathcal{G}]) = s_3$ then*

$$\eta(\sigma)(t_h!) = \mathcal{I}(\sigma)(C\mathcal{F}) = 1$$

- *if $last(\sigma[C\mathcal{F}]) = s_2 \wedge last(\sigma[C\mathcal{G}]) = s_3$ then*

$$\eta(\sigma)(t_t!) = \mathcal{I}(\sigma)(C\mathcal{F}) = 1$$

- *if $last(\sigma[C\mathcal{F}]) = s_0 \wedge last(\sigma[C\mathcal{G}]) = s_4$ then*

$$\begin{aligned}\eta(\sigma)(g_h!) &= \mathcal{I}(\sigma)(C\mathcal{G}) \cdot \eta_{C\mathcal{G}}(\sigma[C\mathcal{G}])(g_h!)\\ &= \eta_{C\mathcal{G}}(\sigma[C\mathcal{G}])(g_h!)\\ \eta(\sigma)(g_t!) &= \mathcal{I}(\sigma)(C\mathcal{G}) \cdot \eta_{C\mathcal{G}}(\sigma[C\mathcal{G}])(g_t!)\\ &= \eta_{C\mathcal{G}}(\sigma[C\mathcal{G}])(g_t!).\end{aligned}$$

2.3 Strongly Distributed Schedulers

Although the class of distributed schedulers already realistically restricts the local decisions of processes in a distributed setting, in certain cases there exist distributed schedulers, where the interleaving schedulers are too powerful. In essence, the problem is that a distributed scheduler may use information from a component $\mathcal{P}_1$ to decide how to pick between components $\mathcal{P}_2$ and $\mathcal{P}_3$. In certain settings this is unrealistic. To counter this problem, strongly distributed schedulers have been introduced [2].

Given any two components $\mathcal{P}_i$, $\mathcal{P}_j$ of a distributed I/O-IPC $C = \mathcal{P}_1 \| \ldots \| \mathcal{P}_n$, consider the following property: for all σ, σ' such that $\sigma[\mathcal{P}_i] = \sigma'[\mathcal{P}_i]$ and $\sigma[\mathcal{P}_j] = \sigma'[\mathcal{P}_j]$, if $\mathcal{I}(\sigma)(\mathcal{P}_i) + \mathcal{I}(\sigma)(\mathcal{P}_j) \neq 0$ and $\mathcal{I}(\sigma')(\mathcal{P}_i) + \mathcal{I}(\sigma')(\mathcal{P}_j) \neq 0$ then

$$\frac{\mathcal{I}(\sigma)(\mathcal{P}_i)}{\mathcal{I}(\sigma)(\mathcal{P}_i) + \mathcal{I}(\sigma)(\mathcal{P}_j)} = \frac{\mathcal{I}(\sigma')(\mathcal{P}_i)}{\mathcal{I}(\sigma')(\mathcal{P}_i) + \mathcal{I}(\sigma')(\mathcal{P}_j)}. \tag{2.1}$$

Definition 11. *A scheduler η is strongly distributed if it is distributed and the restriction in Equation (2.1) holds for the interleaving scheduler $\mathcal{I}$ of η.*

We denote the set of all distributed schedulers by *SDS*.

The intuition behind strongly distributed scheduler is that the choices the interleaving scheduler makes between two components $\mathcal{P}_i$, $\mathcal{P}_j$ should be consistent with respect to the local paths of $\mathcal{P}_i$, $\mathcal{P}_j$. If for two global paths, the local paths of $\mathcal{P}_i$, $\mathcal{P}_j$ are identical, then the probability of choosing $\mathcal{P}_i$ under the condition that we choose either $\mathcal{P}_i$ or $\mathcal{P}_j$ should be identical for both global paths.

Example 6. *Let $\eta_{C\mathcal{F}}$, $\eta_{C\mathcal{G}}$ be local schedulers, $\mathcal{I}$ an interleaving scheduler, and η a strongly distributed scheduler for the game depicted by Figures 1.2 and 1.3. Further consider in $C\mathcal{F}\|C\mathcal{G}$ the paths*

$$\sigma = \langle s_0, s_3\rangle\, \mu_{C\mathcal{F}\|C\mathcal{G}}(\langle s_0, s_3\rangle)\, \langle s_1, s_4\rangle\, t_h\, \langle s_0, s_4\rangle\, g_t\, \langle s_0, s_3\rangle\, \mu_{C\mathcal{F}\|C\mathcal{G}}(\langle s_0, s_3\rangle)\, \langle s_2, s_4\rangle$$
$$\sigma' = \langle s_0, s_3\rangle\, \mu_{C\mathcal{F}\|C\mathcal{G}}(\langle s_0, s_3\rangle)\, \langle s_1, s_4\rangle\, g_t\, \langle s_1, s_3\rangle\, t_h\, \langle s_0, s_3\rangle\, \mu_{C\mathcal{F}\|C\mathcal{G}}(\langle s_0, s_3\rangle)\, \langle s_2, s_4\rangle,$$

where $\mu_{C\mathcal{F}\|C\mathcal{G}} : S_{C\mathcal{F}\|C\mathcal{G}} \rightharpoonup Dist(S_{C\mathcal{F}\|C\mathcal{G}})$ represents the next-state probability distribution in $C\mathcal{F}\|C\mathcal{G}$.

Notice that the given paths $\sigma, \sigma' \in Paths(C\mathcal{F}\|C\mathcal{G})$ have identical path projections onto $C\mathcal{F}$ and $C\mathcal{G}$: $\sigma[C\mathcal{F}] = \sigma'[C\mathcal{F}]$ and $\sigma[C\mathcal{G}] = \sigma'[C\mathcal{G}]$.

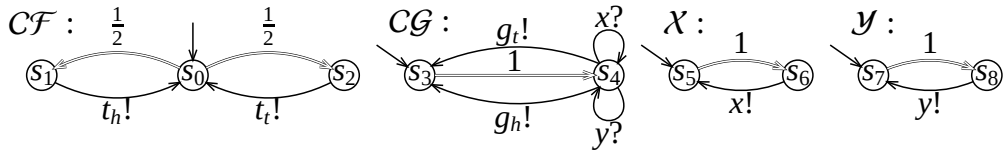
Since $\mathcal{I}(\sigma)(C\mathcal{F}) + \mathcal{I}(\sigma)(C\mathcal{G}) = 1 = \mathcal{I}(\sigma')(C\mathcal{F}) + \mathcal{I}(\sigma')(C\mathcal{G})$, Definition 11 implies that the interleaving scheduler $\mathcal{I}$ of η will be additionally restricted for all paths σ, σ' such that $\sigma[C\mathcal{F}], = \sigma'[C\mathcal{F}]$ and $\sigma[C\mathcal{G}] = \sigma'[C\mathcal{G}]$ by Equation (2.1):

$$\mathcal{I}_\eta(\sigma)(C\mathcal{F}) = \mathcal{I}_\eta(\sigma')(C\mathcal{F}) \text{ and } \mathcal{I}_\eta(\sigma)(C\mathcal{G}) = \mathcal{I}_\eta(\sigma')(C\mathcal{G}).$$

For a more specific case study, consider the modified "repeating coin flip & guess" game described in Figure 2.1 where two additional components are added.

Suppose that the distributed I/O-IPC $C\mathcal{F}\|C\mathcal{G}\|\mathcal{X}\|\mathcal{Y}$ is subject to a distributed scheduler η built of interleaving scheduler $\mathcal{I}$ and local schedulers $\eta_{C\mathcal{F}}$, $\eta_{C\mathcal{G}}$, $\eta_{\mathcal{X}}$ and $\eta_{\mathcal{Y}}$. Since the local history of the coin-flip $C\mathcal{F}$ is not available to the coin-guess $C\mathcal{G}$, it is expected that the probability of having a matching flip/guess within one step is $1/2$.

However, there are interleaving schedulers which guarantee that the system always arrives at a matching flip/guess within one time step. A possible example of such a scheduler is described below.



The coin-guess has additional transitions $s_4 \xrightarrow{x?} s_4$ and $s_4 \xrightarrow{y?} s_4$ matching the output transitions of $\mathcal{X}$ and $\mathcal{Y}$. Otherwise, $\mathcal{CF}$ and $\mathcal{CG}$ remain the same as in Figure 1.2.

Figure 2.1: I/O-IPC Models of the modified Repeated Coin-Flip Experiment

Assume that after the probabilitic step of the system, depending whether $\mathcal{CF}$ has progressed to s_1 or s_2, the scheduler $\mathcal{I}$ interleaves "$\mathcal{X}$ and then $\mathcal{Y}$" or, respectively, "$\mathcal{Y}$ and then $\mathcal{X}$". Afterwards, the scheduler $\eta_{\mathcal{GC}}$ for the coin-guess $\mathcal{CG}$ can enforce the match of coin-flip since the order in which $x?$ and $y?$ happened is part of $\mathcal{CG}$'s local history.

Formally, schedulers $\mathcal{I}$ and $\eta_{\mathcal{CG}}$ that enforce a matching flip/guess are described by the following restrictions: for any path $\sigma \in Paths(\mathcal{CF}\|\mathcal{CG}\|\mathcal{X}\|\mathcal{Y})$,

- if $\sigma[\mathcal{CG}] = \ldots \xrightarrow{x?} s_4 \xrightarrow{y?} s_4$ then $\eta(\sigma[\mathcal{CG}])(g_h!) = 1$
- if $\sigma[\mathcal{CG}] = \ldots \xrightarrow{y?} s_4 \xrightarrow{x?} s_4$ then $\eta(\sigma[\mathcal{CG}])(g_t!) = 1$
- if $last(\sigma[\mathcal{CF}]) = s_1$ then
  - if $last(\sigma[\mathcal{X}]) = s_5$ then $\mathcal{I}(\sigma)(\mathcal{X}) = 1$
  - otherwise, if $last(\sigma[\mathcal{Y}]) = s_7$ then $\mathcal{I}(\sigma)(\mathcal{Y}) = 1$
- if $last(\sigma[\mathcal{CF}]) = s_2$ then
  - if $last(\sigma[\mathcal{Y}]) = s_7$ then $\mathcal{I}(\sigma)(\mathcal{Y}) = 1$
  - otherwise, if $last(\sigma[\mathcal{X}]) = s_5$ then $\mathcal{I}(\sigma)(\mathcal{X}) = 1$

As the definitions and the restrictions already imposed on the used schedulers are not violated, the example above shows why distributed schedulers are not good enough for ruling out unrealistic behaviour.

On the other hand, if η would be a strongly distributed scheduler, Equation (2.1) should hold for $\mathcal{I}$. For example, for the paths σ_1, σ_2 of $\mathcal{CF}\|\mathcal{CG}\|\mathcal{X}\|\mathcal{Y}$ given by $\sigma_1 = \langle s_0, s_3, s_5, s_7\rangle \overset{1/2}{\Longrightarrow} \langle s_1, s_4, s_6, s_8\rangle$ and $\sigma_2 = \langle s_0, s_3, s_5, s_7\rangle \overset{1/2}{\Longrightarrow} \langle s_2, s_4, s_6, s_8\rangle$ it holds that $\sigma_1[\mathcal{X}] = \sigma_2[\mathcal{X}]$ and $\sigma_1[\mathcal{Y}] = \sigma_2[\mathcal{Y}]$ which by Equation (2.1) means that:

$$\frac{\mathcal{I}(\sigma_1)(\mathcal{X})}{\mathcal{I}(\sigma_1)(\mathcal{X}) + \mathcal{I}(\sigma_1)(\mathcal{Y})} = \frac{\mathcal{I}(\sigma_2)(\mathcal{X})}{\mathcal{I}(\sigma_2)(\mathcal{X}) + \mathcal{I}(\sigma_2)(\mathcal{Y})}.$$

The latter however – unless $1/(1+0)$ is equal to $0/(0+1)$ – is contradicting with the restrictions on $\mathcal{I}$ as imposed previously for the distributed case.

Strongly distributed schedulers are useful depending on which system is considered for study [2]. For example, when analyzing an auctioning protocol where each component models one of the bidders, then the order in which the bidders interact with the auctioneer should not leak information that can be used to the advantage of the other bidders. In such a situation, strongly distributed schedulers would provide more adequate worst-case/best-case probabilities.

However, if the interleaving scheduler should have access to the history of the components (as it might be the case for a kernel scheduler on a computer) then distributed schedulers should be considered, as the strongly distributed version might rule out valid possibilities.

2.4 Induced Probability Measure

When all the nondeterministic choices in a distributed I/O-IPC are resolved by a scheduler, we end up with a probability measure on sets of paths of the I/O-IPC. We define this probability measure in a similar way as is done for IPCs [14].

Let $C = \mathcal{P}_1 \| \ldots \| \mathcal{P}_n$ be an arbitrarily fixed, closed and distributed I/O-IPC with state space S_C, action set $\mathcal{A}_C$, and initial state $\hat{s}$. The *cylinder* induced by the finite path σ is the set of infinite paths $\sigma^\uparrow = \{\sigma' \mid \sigma' \text{ is infinite and } \sigma \text{ is a prefix of } \sigma'\}$. Let the set of cylinders generate the σ-algebra on infinite paths of C.

Definition 12. *Let η be a (possibly strongly) distributed scheduler on C. The* probability measure induced by *η on the set of infinite paths is the unique probability measure P_η such that, for any $s \in S_C$, $a \in \mathcal{A}_C$ and $\mu \in Dist(S_C)$:*

$$P_\eta(s^\uparrow) = \begin{cases} 1 & \text{if } s = \hat{s} \\ 0 & \text{otherwise} \end{cases}$$

$$P_\eta(\sigma a s^\uparrow) = \begin{cases} P_\eta(\sigma^\uparrow) \cdot \eta(\sigma)(a) & \text{if } last(\sigma) \text{ is vanishing and } last(\sigma) \xrightarrow{a} s \\ 0 & \text{otherwise} \end{cases}$$

$$P_\eta(\sigma \mu s^\uparrow) = \begin{cases} P_\eta(\sigma^\uparrow) \cdot \mu(s) & \text{if } last(\sigma) \text{ is tangible and } last(\sigma) \Rightarrow \mu \\ 0 & \text{otherwise} \end{cases}$$

We are now ready to define time-bounded reachability for I/O-IPCs.

Definition 13. *Given an I/O-IPC $\mathcal{P}$ together with an initial distribution over its states, a set of goal states $\mathcal{G}$ and a time-bound $t \in \mathbb{N}$, the probability to reach $\mathcal{G}$ within t time-steps – notation $P_\eta(\Diamond^{\leq t}\mathcal{G})$ – is given by:*

$$P_\eta(\Diamond^{\leq t}\mathcal{G}) = P_\eta(\bigcup\{\sigma^\uparrow \mid t(\sigma) \leq t \text{ and } last(\sigma) \in \mathcal{G}\})$$

Example 7. *Consider again the coin flip & guess game depicted in Figures 1.2 and 1.3 and let η be a distributed scheduler of $\mathcal{CF}\|\mathcal{CG}$ where $\eta_{\mathcal{CF}}$, $\eta_{\mathcal{CG}}$ and $\mathcal{I}$ are its local and interleaving schedulers.*

Further consider (e.g.) the path $\sigma_\blacktriangleleft$ as given by

$$\sigma_\blacktriangleleft = \overbrace{\underbrace{\langle s_0, s_3\rangle \overset{1/2}{\Rightarrow} \langle s_1, s_4\rangle}_{\sigma''_\blacktriangleleft} \xrightarrow{g_h} \langle s_1, s_3\rangle}^{\sigma'_\blacktriangleleft} \xrightarrow{t_h} \langle s_0, s_3\rangle$$

with its prefixes $\sigma'_\blacktriangleleft$ and $\sigma''_\blacktriangleleft$ as described by the over- and under-braces. This path describes the possibility of having a matching "heads" flip and guess by first performing the guess and then the flip.

Using the definition of the probability measure P_η and the appropriate scheduler restrictions as described by the previous examples, we have:

$$\begin{aligned} P_\eta(\sigma^\uparrow_\blacktriangleleft) &= P_\eta(\sigma'^\uparrow_\blacktriangleleft) \cdot \overbrace{\eta(\sigma'_\blacktriangleleft)(t_h)}^{=1} = P_\eta(\sigma''^\uparrow_\blacktriangleleft) \cdot \eta(\sigma''_\blacktriangleleft)(g_h) \\ &= P_\eta(\sigma''^\uparrow_\blacktriangleleft) \cdot \mathcal{I}(\sigma''_\blacktriangleleft)(\mathcal{CG}) \cdot \eta_{\mathcal{CG}}(\sigma''_\blacktriangleleft[\mathcal{CG}])(g_h) \\ &= P_\eta(\langle s_0, s_3\rangle^\uparrow) \cdot 1/2 \cdot \mathcal{I}(\sigma''_\blacktriangleleft)(\mathcal{CG}) \cdot \eta_{\mathcal{CG}}(\sigma''_\blacktriangleleft[\mathcal{CG}])(g_h) \\ &= 1/2 \cdot \mathcal{I}(\sigma''_\blacktriangleleft)(\mathcal{CG}) \cdot \eta_{\mathcal{CG}}(\sigma''_\blacktriangleleft[\mathcal{CG}])(g_h) \end{aligned}$$

By the same reasoning, for the path

$$\sigma_\blacktriangleright = \overbrace{\underbrace{\langle s_0, s_3\rangle \overset{1/2}{\Rightarrow} \langle s_1, s_4\rangle}_{\sigma''_\blacktriangleright} \xrightarrow{t_h} \langle s_0, s_4\rangle}^{\sigma'_\blacktriangleright} \xrightarrow{g_h} \langle s_0, s_3\rangle$$

which describes the possibility of having a matching "heads" by first performing the flip and then the guess, we get that $P_\eta(\sigma^\uparrow_\blacktriangleright) = 1/2 \cdot \mathcal{I}(\sigma''_\blacktriangleright)(\mathcal{CF}) \cdot \eta_{\mathcal{CG}}(\sigma'_\blacktriangleright[\mathcal{CG}])(g_h)$.

We now take $\mathcal{G}_{\surd_h} = \{s \in S_{\mathcal{CF}\|\mathcal{CG}} \mid \surd_h \text{ is the label of } s\}$ to be the the singleton set describing that the guess matches the "heads" flip. By Definition 13, we have that $P_\eta(\Diamond^{\leq 1}\mathcal{G}_{\surd_h}) = P_\eta(\sigma_\blacktriangleleft) + P_\eta(\sigma_\blacktriangleright)$. In addition, since $\sigma''_\blacktriangleleft \equiv \langle s_0, s_3\rangle \overset{1/2}{\Rightarrow} \langle s_1, s_4\rangle \equiv \sigma''_\blacktriangleright$, and respectively $\sigma''_\blacktriangleleft[\mathcal{CG}] \equiv s_3 \overset{1}{\Rightarrow} s_4 \equiv \sigma'_\blacktriangleright[\mathcal{CG}]$, it follows that

$$\begin{aligned} P_\eta(\Diamond^{\leq 1}\mathcal{G}_{\surd_h}) &= \frac{1}{2}\eta_{\mathcal{CG}}(s_3 \overset{1}{\Rightarrow} s_4)(g_h)\,(\mathcal{I}(\sigma''_\blacktriangleleft)(\mathcal{CF}) + \mathcal{I}(\sigma''_\blacktriangleleft)(\mathcal{CG})) \\ &= \frac{1}{2}\eta_{\mathcal{CG}}(s_3 \overset{1}{\Rightarrow} s_4)(g_h). \end{aligned}$$

For $\mathcal{G}_{\surd_t} = \{s \in S_{\mathcal{CF}\|\mathcal{CG}} \mid \surd_t \text{ is the label of } s\}$ describing the guess to match the "tails" flip we similary get that

$$P_\eta(\Diamond^{\leq 1}\mathcal{G}_{\surd_t}) = \frac{1}{2}\eta_{\mathcal{CG}}(s_3 \overset{1}{\Rightarrow} s_4)(g_t).$$

Combining the two results above it will result that the probability for the flip to match the guess within one time step is $1/2$.

Chapter 3

I/O-IPC Time-Bounded Reachability

From a theoretical point of view, the goal of this thesis is to prove that time-bounded reachability for any scheduler-quantified distributed I/O-IPC can be reduced to computing time-unbounded reachability for an equivalent PMC by *unfolding* the given model. This chapter provides the remaining background information and reasoning for seeing through the forementioned proofs.

3.1 Parametric Markov Chains

To compute time-bounded reachability probabilities we transform scheduler-quantified, distributed I/O-IPCs into parametric Markov models (see Chapter 3.2). In this section we give a brief overview of parametric Markov chains [15, 16, 17].

Let S be a finite set of states and $V = \{x_1, \ldots, x_n\}$ denote a set of variables with domain $\mathbb{R}$. An *assignment* ζ is a function $\zeta : V \to \mathbb{R}$. A *polynomial* g over V is a sum of monomials

$$g(x_1, \ldots, x_n) = \sum_{i_1,\ldots,i_n} a_{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n},$$

where each $i_j \in \mathbb{N}$ and each $a_{i_1,\ldots,i_n} \in \mathbb{R}$. A *rational function* f over V is a fraction $f(x_1, \ldots, x_n) = f_1(x_1, \ldots, x_n)/f_2(x_1, \ldots, x_n)$ of two polynomials f_1, f_2 over V.

Let $\mathcal{F}_V$ denote the set of rational functions from V to $\mathbb{R}$. Given $f \in \mathcal{F}_V$ and an assignment ζ, we let $\zeta(f)$ denote the rational function obtained by substituting each occurrence of $x \in V$ with $\zeta(x)$.

Definition 14. *A parametric Markov chain (PMC) is a tuple $\mathcal{D} = (S, \hat{s}, \mathbf{P}, V)$ where S is a finite set of states, $\hat{s}$ is the initial state, $V = \{v_1, \ldots, v_n\}$ is a finite set of parameters and $\mathbf{P}$ is the probability matrix $\mathbf{P} : S \times S \to \mathcal{F}_V$.*

The matrix $\mathbf{P}$ denotes the probabilities of going from one state to another in one step. Its straightforward generalization for k steps is given below.

Definition 15. *Given a PMC $\mathcal{D} = (S, \hat{s}, \mathbf{P}, V)$, the k-step probability matrix $\mathbf{P}_k$, $k \in \mathbb{N}$, is defined recursively for any $k' > 1$ and states $s, s' \in S$:*

$$\mathbf{P}_0(s, s') = \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{if } s \neq s' \end{cases}$$
$$\mathbf{P}_{k'}(s, s') = \sum_{s'' \in S} \mathbf{P}_{k'-1}(s, s'') \cdot \mathbf{P}(s'', s')$$

3.2 Scheduler-Quantified I/O-IPCs are PMCs

By having the scheduler η fixed, the probabilistic measure P_η together with the scheduled I/O-IPC C would become deterministic. To be more specific, by treating as unknowns the interleaving and local scheduler decisions we end up with analyzing parametric Markov chains. The parameters of this PMC correspond precisely the decisions that the interleaving and local schedulers perform.

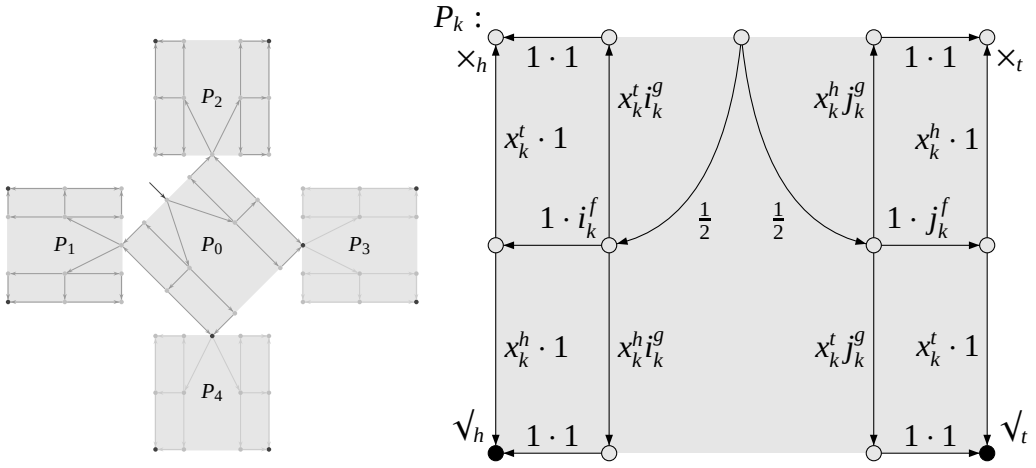
We have seen in Chapter 2.4 that fixing the scheduler of a distributed I/O-IPC induces a probability measure on its paths. Thus, by fixing the scheduler *parametrically*, i.e. by treating the probabilities chosen by the interleaving and local schedulers as parameters, we show that the *unfolding* of the I/O-IPC induces a PMC (see Chapter 3.1) whose states are paths of the distributed I/O-IPC.

To make sure the induced PMC is finite we generate it only for paths up to a specific time-bound t. We then prove that computing the probability to reach a set of states within t time-units is equivalent for the I/O-IPC and the induced PMC.

3.2.1 Repeated Coin Flip & Guess Revisited

To give an idea of how the unfolding works, consider again the repeated coin-flip experiment (Figures 1.2, 1.3). It should intuitively hold that $Pr(\Diamond^{\leq 2}\{\surd_h, \surd_t\}) = 3/4$ if we assume the guessing player has no information about the outcome of each coin-flip. Figure 3.1 describes the unfolding of the distributed I/O-IPC from Figure 1.3 up to time-point 2. On the right-hand side we see the structure of the PMC for one time-step. The unfolding up to 2 time steps is shown schematically on the left-hand side, where each square represents a copy of the right-hand side structure.

The local scheduler decisions in this case for each repeating structure P_k are x_k^h, x_k^t such that $x_k^h + x_k^t = 1$ and the interleaving scheduler decisions are $i_k^g, i_k^f, j_k^g, j_k^f$ such that $i_k^g + i_k^f = j_k^g + j_k^f = 1$. Here x_k^h, for example, denotes the probability assigned by the local scheduler for the guesser to pick "heads" for a local path



All transitions are parametric. Interleaving is used for compacting the model.

Figure 3.1: PMC Scheme up to time 2 for the Repeated Coin-Flip Experiment

ending in a "heads" vs. "tails" choice. The parameters i_k^g, i_k^f (as well as j_k^g, j_k^f) denote the probabilities the interleaving scheduler assigns to the "guessing" vs. the "flipping" model respectively, for a global path which enables them both.

Now, $Pr(\Diamond^{\leq 2}\{\surd_h, \surd_t\})$ can be computed as the sum of the cumulated probabilities on the paths leading to $\{\surd_h, \surd_t\}$ states by using the given unfolding in Figure 3.1 and the above parameter restrictions:

$$\frac{1}{2}\Big(x_0^h \cdot i_0^g + i_0^f \cdot x_0^h + (x_0^t \cdot i_0^g + i_0^f \cdot x_0^t) \cdot [\frac{1}{2}(x_1^h \cdot i_1^g + i_1^f \cdot x_1^h) + \frac{1}{2}(x_1^t \cdot j_1^g + j_1^f \cdot x_1^t)]\Big) +$$
$$\frac{1}{2}\Big(x_0^t \cdot j_0^g + j_0^f \cdot x_0^t + (x_0^h \cdot j_0^g + j_0^f \cdot x_0^h) \cdot [\frac{1}{2}(x_2^h \cdot i_2^g + i_2^f \cdot x_2^h) + \frac{1}{2}(x_2^t \cdot j_2^g + j_2^f \cdot x_2^t)]\Big) =$$
$$\frac{1}{2}(x_0^h + x_0^t \cdot (\frac{1}{2}x_1^h + \frac{1}{2}x_1^t)) + \frac{1}{2}(x_0^t + x_0^h \cdot (\frac{1}{2}x_2^h + \frac{1}{2}x_2^t)) = \frac{3}{4}$$

3.2.2 I/O-IPC and PMC Reachability

We are now ready to define formally the above interpretation of scheduler decisions over distributed I/O-IPCs as parameters.

Definition 16. *Let $S_C^t \subseteq Paths(C)$ be the set of all paths with time-length $\leq t$ in a closed, distributed I/O-IPC $C = \mathcal{P}_1 || \ldots || \mathcal{P}_n$ which does not exhibit Zeno-behaviour. Define the parameters set V by*

$$V = \{\, y_\sigma^i \mid \sigma \in S_C^t, 1 \leq i \leq \#C, A^{en}_{last(\sigma[\mathcal{P}_i]),\mathcal{P}_i} \neq \emptyset\}$$
$$\cup \{\, x^a_{\sigma[\mathcal{P}_i]} \mid \sigma \in S_C^t, 1 \leq i \leq \#C, a \in A^{en}_{last(\sigma[\mathcal{P}_i]),\mathcal{P}_i}\}$$

and let $\mathbf{P}$ *match the induced probability measure, namely for any path* $\sigma \in S^t_C$*, any state* s *of* C*, any action* a *of* C *and any distribution* μ *over the states of* C*:*

$$\begin{aligned}
\mathbf{P}(\sigma, \sigma a s) &= y^i_\sigma \cdot x^a_{\sigma[\mathcal{P}_i]} && \text{if } last(\sigma) \text{ is vanishing, } last(\sigma) \xrightarrow{a} s,\ a \in A^{en}_{last(\sigma[\mathcal{P}_i]),\mathcal{P}_i} \\
\mathbf{P}(\sigma, \sigma \mu s) &= \mu(s) && \text{if } last(\sigma) \text{ is tangible, } \mathrm{t}(\sigma) < t,\ last(\sigma) \Rightarrow \mu \\
\mathbf{P}(\sigma, \sigma) &= 1 && \text{if } last(\sigma) \text{ is tangible, } \mathrm{t}(\sigma) = t.
\end{aligned}$$

All other transition probabilities are zero. The unfolding of the I/O-IPC C *up to time bound* t *is then the PMC* $\mathcal{D} = (S^t_C, \hat{s}_C, \mathbf{P}, V)$*.*

Given a set of states $\mathcal{G}$ *of* C*, we write* $\overline{\mathcal{G}}$ *for the paths in* S^t_C *that end in a state in* $\mathcal{G}$*, but never visit a state in* $\mathcal{G}$ *on the way.*

The finiteness of S^t_C and V is guaranteed by the exclusion of infinite chains consisting of only immediate actions. This exclusion really implies that, for each state in C, a tangible state is reachable within a finite number of non-probabilistic steps.

The variables in Definition 16 can be restricted to ensure that they represent valid scheduler decisions in the following way:

$$\begin{aligned}
&0 \le v \le 1 && \text{if } v \in V \\
&\textstyle\sum_{a \in A} x^a_{\sigma[\mathcal{P}_i]} = 1 && \text{if } \sigma \in S^t_C \text{ and } 1 \le i \le \#C \text{ with } A = A^{en}_{last(\sigma[\mathcal{P}_i]),\mathcal{P}_i} \\
&\textstyle\sum_{i \in I} y^i_\sigma = 1 && \text{if } \sigma \in S^t_C \text{ with } I = \{i \mid 1 \le i \le \#C, last(\sigma[P_i]) \text{ vanishing}\}
\end{aligned} \tag{3.1}$$

We write $\zeta \vdash (3.1)$ if $\zeta : V \to [0,1]$ satisfies (3.1). I/O-IPC path probabilities are related to k-step transition probabilities of the induced PMC by the following.

Lemma 1. *For a closed, distributed I/O-IPC* C*, let* $\mathcal{D}$ *be as in Definition 16. Then*

(i) For every distributed scheduler η *there is an assignment* $\zeta \vdash (3.1)$ *such that for all* $\sigma \in S^t_C$*:* $P_\eta(\sigma^\uparrow) = \zeta(\mathbf{P}_k(\hat{s}, \sigma))$ *where* k *is the length of* σ*.*

(ii) Reciprocally, for every assignment $\zeta : V \to [0,1]$ *with* $\zeta \vdash (3.1)$ *there is a distributed scheduler* η *such that for all* $\sigma \in S^t_C$*:* $P_\eta(\sigma^\uparrow) = \zeta(\mathbf{P}_k(\hat{s}, \sigma))$*.*

Proof. For both (i) and (ii) set the assignment ζ, respectively the distributed scheduler η such that for $1 \le i \le \#C$:

$$\begin{aligned}
\zeta(y^i_\sigma) &= \mathcal{I}(\sigma)(\mathcal{P}_i) && \text{if } A^{en}_{last(\sigma[\mathcal{P}_i]),\mathcal{P}_i} \neq \emptyset \\
\zeta(x^a_{\sigma[\mathcal{P}_i]}) &= \eta_{\mathcal{P}_i}(\sigma[\mathcal{P}_i])(a) && \text{if } a \in A^{en}_{last(\sigma[\mathcal{P}_i]),\mathcal{P}_i}
\end{aligned}$$

This gives indentical mappings from assignments to distributed schedulers and back for (i) and (ii), which means both cases can be proved simultaneously.

For a distributed scheduler η and its associated assignment ζ, we now show that $P_\eta(\sigma^\uparrow) = \zeta(\mathbf{P}_k(\hat{s}, \sigma))$ by induction on the length k of σ:

- for paths of length 0 we have that

$$P_\eta(\hat{s}^\uparrow) = 1 = \zeta(\mathbf{P}_0(\hat{s},\hat{s})) \text{ and, for } s \neq \hat{s},\ P_\eta(s^\uparrow) = 0 = \zeta(\mathbf{P}_0(\hat{s},s)).$$

- for the inductive step, let the induction hypothesis (IH) be as follows: *Given a path $\sigma \in S_C^t$ of length $k > 0$, for any path σ' of length $k-1$:*

$$P_\eta(\sigma') = \zeta(\mathbf{P}_{k-1}(\hat{s},\sigma')).$$

By case distinction:

1. For $last(\sigma)$ vanishing we have:

$$\begin{aligned} P_\eta(\sigma^\uparrow) &\overset{\text{Def 12}}{=} \sum_{\sigma=\sigma' as} P_\eta(\sigma'^\uparrow)\cdot\eta(\sigma')(a) \overset{IH}{=} \sum_{\sigma=\sigma' as} \zeta(\mathbf{P}_{k-1}(\hat{s},\sigma'))\cdot\eta(\sigma')(a)\\ &\overset{\text{Def 10}}{=} \sum_{\sigma=\sigma' as} \zeta(\mathbf{P}_{k-1}(\hat{s},\sigma'))\cdot\sum_{i=1,e(i,a,\sigma)}^{n} \mathcal{I}(\sigma)(\mathcal{P}_i)\cdot\eta_{\mathcal{P}_i}(\sigma[\mathcal{P}_i])(a)\\ &\overset{\text{Def 15}}{=} \sum_{\sigma=\sigma' as} \zeta(\mathbf{P}_{k-1}(\hat{s},\sigma'))\cdot\zeta(\mathbf{P}(\sigma',\sigma)) = \zeta(\mathbf{P}_k(\hat{s},\sigma)). \end{aligned}$$

2. For $last(\sigma)$ tangible we have:

$$\begin{aligned} P_\eta(\sigma^\uparrow) &\overset{\text{Def 12}}{=} \sum_{\sigma=\sigma'\mu s} P_\eta(\sigma'^\uparrow)\cdot\mu(s)\\ &\underset{IH}{\overset{\text{Def 15}}{=}} \sum_{\sigma=\sigma'\mu s} \zeta(\mathbf{P}_{k-1}(\hat{s},\sigma'))\cdot\zeta(\mathbf{P}(\sigma',\sigma)) = \zeta(\mathbf{P}_k(\hat{s},\sigma)). \end{aligned}$$

□

The bounded reachability problem for I/O-IPCs subject to distributed scheduling can now be reformulated as an unbounded reachability problem for the associated induced PMC.

Theorem 1. *Time-bounded reachability for an I/O-IPC C subject to distributed scheduling is equivalent to checking time-unbounded reachability on the PMC $\mathcal{D} = (S_C^t, \hat{s}_C, \mathbf{P}, V)$ as in Definition 16 for assignments that satisfy (3.1). In particular:*

$$\sup_{\eta\in DS} P_\eta(\Diamond^{\leq t}\mathcal{G}) = \sup_{\zeta\vdash(3.1)} \zeta(P_{\mathcal{D}}(\Diamond\,\overline{\mathcal{G}})) \;\&\; \inf_{\eta\in DS} P_\eta(\Diamond^{\leq t}\mathcal{G}) = \inf_{\zeta\vdash(3.1)} \zeta(P_{\mathcal{D}}(\Diamond\,\overline{\mathcal{G}})).$$

Proof. For a distributed scheduler η with associated assignment ζ, as defined in Lemma 1, we have $P_\eta(\Diamond^{\leq t}\mathcal{G}) = P_\eta(\bigcup\{\sigma^\uparrow \mid \mathrm{t}(\sigma) \leq t \text{ and } \mathit{last}(\sigma) \in \mathcal{G}\})$.

Recall that the set of goal states $\overline{\mathcal{G}} \subset S^t_C$ of the PMC corresponds to the set of paths that end in a state in $\mathcal{G}$, but do not pass through $\mathcal{G}$ under way in the analyzed I/O-IPC.

It is obvious that the cylinders induced by these paths do not overlap and their union is the set of all paths reaching $\mathcal{G}$ within t time-units.

We then have:

$$P_\eta(\Diamond^{\leq t}\mathcal{G}) = P_\eta\Big(\bigcup\{\sigma^\uparrow \mid \sigma \in \overline{\mathcal{G}}\}\Big) = \sum_{\sigma\in\overline{\mathcal{G}}} P_\eta(\sigma^\uparrow) = \sum_{\sigma\in\overline{\mathcal{G}},|\sigma|=k} \zeta(\mathbf{P}_k(\hat{s},\sigma))$$
$$= \zeta\Big(\sum_{\sigma\in\overline{\mathcal{G}},|\sigma|=k} \mathbf{P}_k(\hat{s},\sigma)\Big) = \zeta(\mathbf{P}(\Diamond\,\overline{\mathcal{G}})).$$

The last equality stems from the fact that a path σ of length k can only be reached for the first time after exactly k steps in $\mathcal{D}$.

We now have a one-to-one correspondence between distributed schedulers of $\mathcal{C}$ and assignments of $\mathcal{D}$ which satisfy (3.1) that preserves the probability to reach $\mathcal{G}$, respectively $\overline{\mathcal{G}}$.

It immediately follows that the infimum and supremum probabilities over these schedulers and, respectively, assignments must be equal. □

To extend this result to strongly distributed schedulers we must further restrict the variables of the induced PMC such that the allowed assignments match the strongly distributed schedulers. First we introduce new variables which represent the conditional probabilities in (2.1).

For every i, j, $1 \leq i, j \leq \#\mathcal{C}$, $i \neq j$, and $\sigma \in S^t_C$, we define a new variable $z^{i,j}_{\sigma[\mathcal{P}_i],\sigma[\mathcal{P}_j]} \notin V$. Notice that two different $\sigma, \sigma' \in S^t_C$ may induce the same variable if $\sigma[\mathcal{P}_i] = \sigma'[\mathcal{P}_i]$ and $\sigma[\mathcal{P}_j] = \sigma'[\mathcal{P}_j]$.

We write V_z for the set of all such variables $z^{i,j}_{\sigma[\mathcal{P}_i],\sigma[\mathcal{P}_j]}$.

Using these new variables we pose new restrictions on the variables of the induced PMC of a distributed I/O-IPC.

$$z^{i,j}_{\sigma[\mathcal{P}_i],\sigma[\mathcal{P}_j]}(y^i_\sigma + y^j_\sigma) = y^i_\sigma \quad \text{if } 1 \leq i, j \leq \#\mathcal{C},\ i \neq j, \text{ and } \sigma \in S^t_C \tag{3.2}$$

Theorem 2. *Time-bounded reachability for an I/O-IPC $\mathcal{C}$ subject to strongly distributed scheduling is equivalent to checking time-unbounded reachability on the PMC $\mathcal{D} = (S^t_C, \hat{s}_C, \mathbf{P}, V \cup V_z)$ resulted through unfolding as in Definition 16 under the assumptions* (3.1) *and* (3.2)*. In particular:*

$$\sup_{\eta\in SDS} P_\eta(\Diamond^{\leq t}\mathcal{G}) = \sup_{\zeta\vdash(3.1)\wedge(3.2)} \zeta(P_\mathcal{D}(\Diamond\,\overline{\mathcal{G}})) \;\&\; \inf_{\eta\in SDS} P_\eta(\Diamond^{\leq t}\mathcal{G}) = \inf_{\zeta\vdash(3.1)\wedge(3.2)} \zeta(P_\mathcal{D}(\Diamond\,\overline{\mathcal{G}})).$$

Proof. We associate strongly distributed schedulers η to assignments ζ following Lemma 1. For the extra variables in V_z we choose

$$\zeta(z^{i,j}_{\sigma[\mathcal{P}_i],\sigma[\mathcal{P}_j]}) := \begin{cases} \zeta\left(\frac{y^i_\sigma}{y^i_\sigma + y^j_\sigma}\right) & \text{if } \zeta(y^i_\sigma + y^j_\sigma) > 0 \\ 1 & \text{otherwise.} \end{cases}$$

Note that the value 1 is chosen arbitrarily here.

We now show that any assignment that satisfies (3.2) is associated to a strongly distributed scheduler and that any strongly distributed scheduler is associated to an assigment that satisfies (3.2).

First, notice that for a path σ ending in a vanishing state and distinct I/O-IPCs $\mathcal{P}_i$ and $\mathcal{P}_j$ that have immediate actions enabled after σ, we have that if $\zeta(y^i_\sigma) = 0 = \zeta(y^j_\sigma)$ then $z^{i,j}_{\sigma[\mathcal{P}_i],\sigma[\mathcal{P}_j]}(y^i_\sigma + y^j_\sigma) = y^i_\sigma$ holds, regardless the value of $\zeta(z^{i,j}_{\sigma[\mathcal{P}_i],\sigma[\mathcal{P}_j]})$.

Now, consider an assignment ζ with associated distributed scheduler η, and suppose we find two paths σ, σ' as above with $\zeta(y^i_\sigma + y^j_\sigma) \neq 0 \neq \zeta(y^i_{\sigma'} + y^j_{\sigma'})$, $\sigma[\mathcal{P}_i] = \sigma'[\mathcal{P}_i]$ and $\sigma[\mathcal{P}_j] = \sigma'[\mathcal{P}_j]$. Then (3.2) gives us that:

$$\zeta\left(\frac{y^i_\sigma}{y^i_\sigma + y^j_\sigma}\right) = \zeta(z^{i,j}_{\sigma[\mathcal{P}_i],\sigma[\mathcal{P}_j]}) = \zeta(z^{i,j}_{\sigma'[\mathcal{P}_i],\sigma'[\mathcal{P}_j]}) = \zeta\left(\frac{y^i_{\sigma'}}{y^i_{\sigma'} + y^j_{\sigma'}}\right)$$

As $\zeta(y^i_\sigma)$ corresponds to $\mathcal{I}(\sigma)(\mathcal{P}_i)$ in η – and similar correspondences can be found for $y^j_\sigma, y^i_{\sigma'}, y^j_{\sigma'}$ – we have that now Equation (2.1) holds for η, which means that η is indeed strongly distributed.

Since we have a one-to-one correspondence between distributed schedulers and assigments, this also proves the reverse, that the assigment associated to a strongly distributed scheduler satisfies (3.2).

Following the proof of Theorem 1, for a strongly distributed scheduler η with associated assignment ζ, we again have

$$P_\eta(\Diamond^{\leq t}\mathcal{G}) = P_\eta(\bigcup\{\sigma^\uparrow \mid \mathrm{t}(\sigma) \leq t \text{ and } \mathit{last}(\sigma) \in \mathcal{G}\}).$$

Recall that the set of paths $\overline{\mathcal{G}} \subset S^t_C$ is the set of paths that end in a state in $\mathcal{G}$, but do not pass through $\mathcal{G}$ under way.

It is obvious that the cylinders induced by these paths do not overlap and their union is the set of all paths reaching $\mathcal{G}$ within t time-units.

It again holds that:

$$\begin{aligned} P_\eta(\Diamond^{\leq t}\mathcal{G}) &= P_\eta\Big(\bigcup\{\sigma^\uparrow \mid \sigma \in \overline{\mathcal{G}}\}\Big) = \sum_{\sigma\in\overline{\mathcal{G}}} P_\eta(\sigma^\uparrow) = \sum_{\sigma\in\overline{\mathcal{G}},|\sigma|=k} \zeta(\mathbf{P}_k(\hat{s},\sigma)) \\ &= \zeta\Big(\sum_{\sigma\in\overline{\mathcal{G}},|\sigma|=k} \mathbf{P}_k(\hat{s},\sigma)\Big) = \zeta(\mathbf{P}(\Diamond\,\overline{\mathcal{G}})). \end{aligned}$$

We now have a one-to-one correspondence between strongly distributed schedulers of C and assignments of $\mathcal{D}$ which satisfy (3.1) and (3.2) that preserves the probability to reach $\mathcal{G}$, respectively $\overline{\mathcal{G}}$.

Again, it immediately follows that the infimum and supremum probabilities over these schedulers and, respectively, assignments must be equal. □

Chapter 4

Implementation Workflow

Time-unbounded reachability probabilities for PMCs can be computed using the tool PARAM [17], which results in analyzing a set of polynomial functions over the variables. These functions can be optimized under the imposed constraints on variables using standard numerical solvers.

The prototype implementation of the algorithm requires as inputs: a closed, distributed I/O-IPC C subject only to output nondeterminism which exhibits no Zeno-behavior (no infinite immediate loops allowed) and a time-bound t.

The following steps are sequentially executed:

1. the I/O-IPC is unfolded up to time-bound t, yielding a PMC with goal states $\mathcal{G}$. At the same time, constraints over the introduced parameters are generated depending on the type of scheduler used;
2. the probability of reaching $\mathcal{G}$ in the generated PMC is computed parametrically using the PARAM tool. The result is a polynomial function.
3. the computed polynomial function is optimized under the generated constraints using non-linear programming. In principle, any non-linear programming tool can be used.

The novel addition to the toolchain depicted in Figure 4.1 is the "unfolder" which is implemented in Java on top of straightforward classes for both main data structures used: basic I/O-IPCs and PMCs.

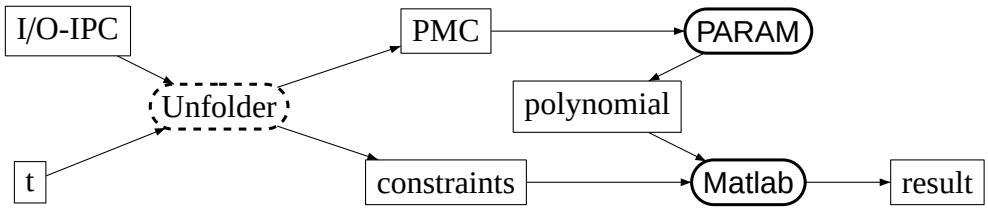


Figure 4.1: Implementation Toolchain (ellipses $\rightsquigarrow$ tools, boxes $\rightsquigarrow$ data)

The unfolder is basically an algorithm which performs the expansion of a scheduled distributed I/O-IPC into a PMC up to the given time bound. While performing the expansion, local and global parameters corresponding to local and interleaving scheduler decisions, are produced.

Paths of the basic I/O-IPCs describing the distributed I/O-IPC model are represented and created on-the-fly through a dynamic tree structure. Since such local paths can be easily tested for equality against other such paths (e.g. when generating additional constraints of strongly distributed schedulers) this simplifies the implementation while also accounting for a bit of memory saving.

The remainder of this chapter gives more information on the "unfolder" which is the central developed part of the toolchain.

4.1 Unfolder Overview

To begin with, the inputted representation of the basic I/O-IPCs is tokenized and the objects representing the components of the distributed I/O-IPC are created. At the same time, depending on the input given, the created I/O-IPC goal states are represented by either of

- `NonSticky` configurations: specified by a set of distributed I/O-IPC states
- `Sticky` configurations: specified by a subset of all components' states

In the Sticky case, if a state of the given subset is part of a state of the distributed I/O-IPC, the distributed state in question will be a goal state.

The input is structured as follows. The first line names how many basic components are parallelized (`#IOIPCs`) and – if the goal states should be specified as `NonSticky` configurations – the number of configurations describing the goal states ($\#\mathcal{G}$) is also provided.

If the goal states are known to be `NonSticky` from the first line, each of the following $\#\mathcal{G}$ lines of input contain one goal configuration. Otherwise, the goal states are specified by listing on each of the following `#IOIPCs` lines the basic I/O-IPC `Sticky` states.

The remaining lines contain triples describing basic I/O-IPC transitions by:

- the transition starting state: added only if it has not been named by any of the goal states/configurations or previously processed states
- the transition label: if ending in "?" or "!" it is seen as an input, respectively output label; otherwise it is parsed as the `double` value of the probability to go from the starting state to the target state
- the transition target state: added only if it has not been named by any of the goal states/configurations or previously processed states

Note that the current implementation does not perform checks whether transition labels are allowed for the specific I/O-IPC transitions for which they are used.

Also, at the moment only input labels and output labels are taken into consideration.

The PMC model expansion follows the initialization of the I/O-IPC components and goal states. Creating the PMC corresponding to the distributed I/O-IPC up to the given time bound requires:

- handling synchronization of input and output transitions enabled in distributed vanishing states
- producing all possible combinations of concurrent local probabilistic transitions enabled in distributed tangible states
- parameter and (strongly) distributed scheduling constraint generation

The generation of additional constraints for strongly distributed schedulers is, for now, treated separately after the PMC expansion.

In the end, the created PMC is handed over to PARAM for computing the polynomial describing the parametric probability of reaching the goal states. The polynomial returned by PARAM is combined with the generated constraints and passed on to a nonlinear programming tool.

In the tests and case studies performed, the *active-set* algorithm [18, 19] provided by the *fmincon* function of Matlab[1] was used.

4.2 Object-Oriented Basic I/O-IPCs

The data structures through describing the implementation of basic I/O-IPCs with goal states are given in Figure 4.2. The design was aimed at being minimalistic and complete while also providing for an efficient implementation of the PMC generation.

Each basic I/O-IPC is described by its `states`, of which the `initial` state, needed for creating the initial PMC state, is singled out.

The centerpiece information in the connected structure is the one representing local I/O-IPC states. For simplicity, the enabled transitions (notation `edge`) are split through corresponding state attributes into input (notation `in`), output (notation `out`) and probabilistic (notation `pr`) transitions.

The states are identifyed through their unique `name/id`. Each transition (notation `Edge`) is part of one of the `in`, `out` or `prob` attributes of a state. For this reason, each transition could be compressed to holding information only on its `target` state and `label`.

The different types of labels are implementing classes of the common `Label` interface. Labels of input (notation `In`) and output (notation `Out`) transitions have as attribute a `name/id` which determines them uniquely. Labels of probabilistic

[1] see http://www.mathworks.com/.../fmincon

(notation `Pr`) transitions have as attribute the probability (notation `prob`) corresponding to reaching the target `State` via the given labeled `Edge`.

As argued before `Goals` may be `Sticky` or `NonSticky`. In either case, they are described by their sole attribute consisting of one or more `State` objects.

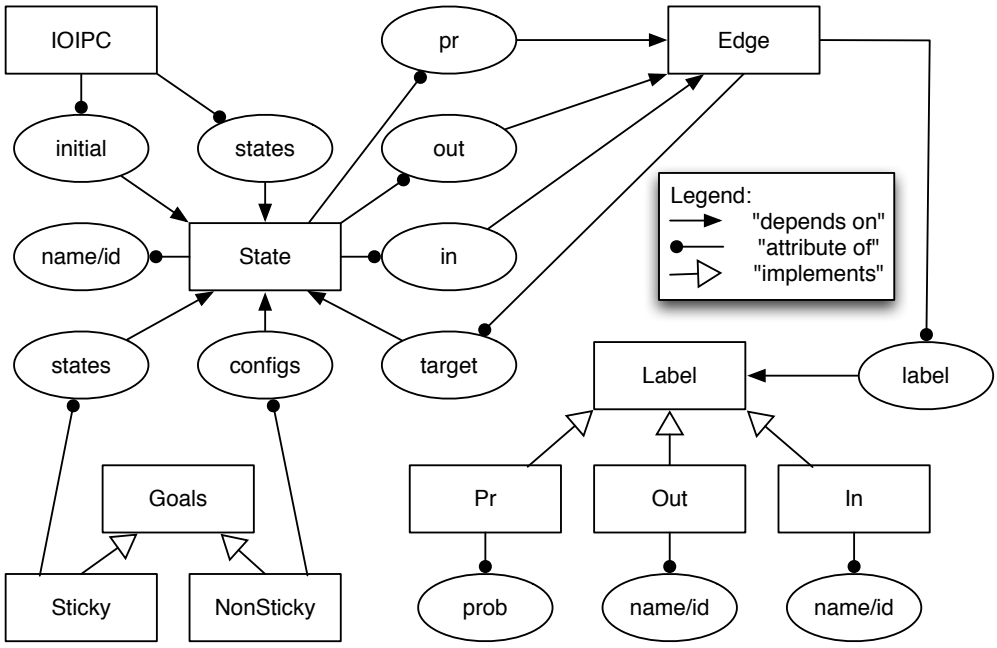


Figure 4.2: Dependency Diagram: Basic IOIPC

4.3 Tree Representation of Local I/O-IPC Paths

Each PMC state will be described mainly by a path of the distributed I/O-IPC. Paths of the distributed I/O-IPC have associated a set of local I/O-IPC paths (their projections onto the different components).

An argument of why different local I/O-IPC paths correspond to various distributed paths has been briefly explained in Example 2.

Since paths of the distributed I/O-IPC and the PMC are so closely related, distributed I/O-IPC paths need not be specifically used by the “unfolder”. However, a representation for local paths of the given model components is needed.

Paths of basic I/O-IPCs could have been represented straightforwardly as ordered lists of states and transitions ending in a state. However, such a representation would have imposed a high memory usage for the multiple different paths that have to be stored with each PMC state.

For example, local path comparison, needed for constraint generation in the strongly distributed scheduler case, would have been pretty expensive.

Also, creating new lists describing such paths whenever a new PMC state is created and iterating through such lists would have been undesirable (both generally and complexity-wise).

Local paths are dynamically constructed when new PMC states are created during model expansion. The data structures describing the used representation of local paths is depicted in Figure 4.3.

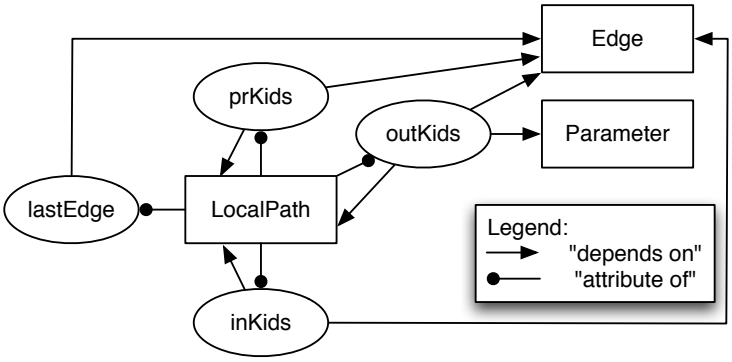


Figure 4.3: Dependency Diagram: IOIPC Local Path

For each local path, its `inKids`, `outKids` and `prKids` attributes will point, for each possible path expanding input, output or probabilistic transition (i.e. for each `Edge`), to another `LocalPath` object.

The nice thing about the `LocalPath` objects to which the `*Kids` attributes point to, is that at first they are not initialized. However, when needed to create new PMC states they get initialized and, if reached subsequently within the model expansion algorithm, they will be detected.

In addition, the `outKids` attribute will associate, upon initialization, each expanding `LocalPath` with the parameter corresponding to its `lastEdge`, thus being equipped for use when creating parametric PMC transitions.

Constraints on local parameters are created within each `LocalPath` upon initialization and added to the separately stored constraint set.

In general, the `lastEdge` will represent the transition seen last on the described path. However, for the local path consisting only of the `initial` basic I/O-IPC state, the `lastEdge` is initialized to an edge with the `initial` state as target and a non-initialized `label`.

Such a special-case handling is needed by the expansion algorithm which makes use of the `target` state of the `lastEdge` of local paths when determining PMC goal states.

Every parameter associated with a path-expanding `Edge` and, respectively, an expansion `LocalPath` is iteratively created by using the same static local parameter generator for all paths of a given IOIPC.

Example 8. *Consider again the repeated coin flip and guess experiment depicted by the I/O-IPCs in Figures 1.2 and 1.3.*

Each PMC state for the distributed I/O-IPC $\mathcal{CF}\|\mathcal{CG}$ expanded up to time bound 1 corresponds to one of the paths starting from the root of the tree on the bottom side in Figure 8 and ending in one of its leaves.

The trees on the top-left and top-right sides in Figure 8 represent the way local paths are expanded in a similar manner for the projections of the depicted tree paths of $\mathcal{CF}\|\mathcal{CG}$.

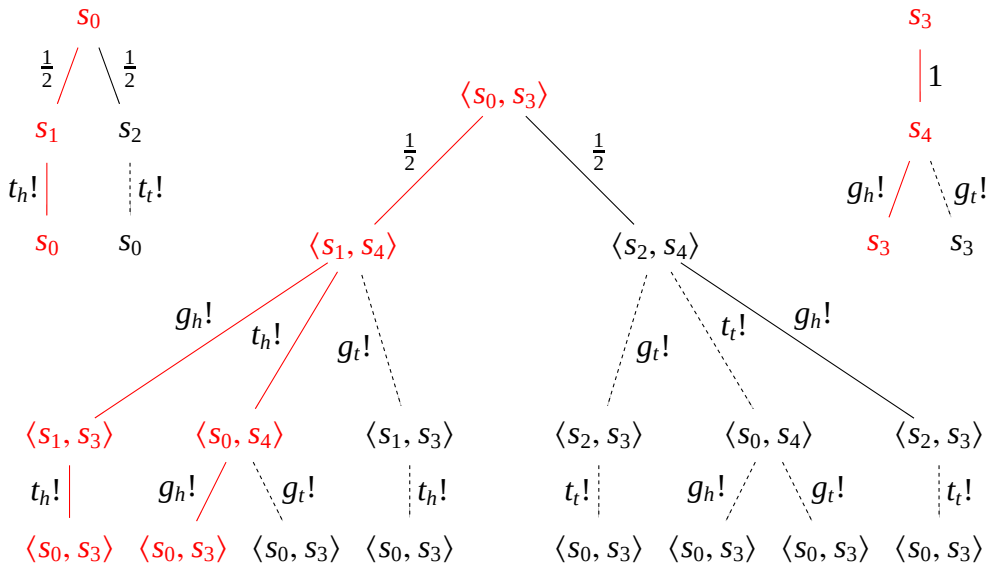


Figure 4.4: Tree Path Representations for $\mathcal{CF}$, $\mathcal{CF}\|\mathcal{CG}$ and $\mathcal{CG}$

With this view in sight, local paths are represented through the states of the fringe in the trees for $\mathcal{CF}$ and $\mathcal{CG}$ – i.e. the leaf states connected to the root of the tree through undotted lines.

The dotted lines would represent the possible continuations of the paths which have not yet been entirely expanded by the unfolder.

From an implementation point of view, the states not connected by straight lines to the root of the tree are the ones which have not yet been initialized/created.

As it can be seen in the depicted example, the two red-colored distributed I/O-IPC paths have identical path projections in $\mathcal{CF}$ and $\mathcal{CG}$.

4.4 Object-Oriented PMCs

The data structures by which PMCs are implemented are described in Figure 4.5. Each PMC is described by its states (notation `nodes`), of which, the initial state is singled out and used to begin the recursion of the PMC expansion.

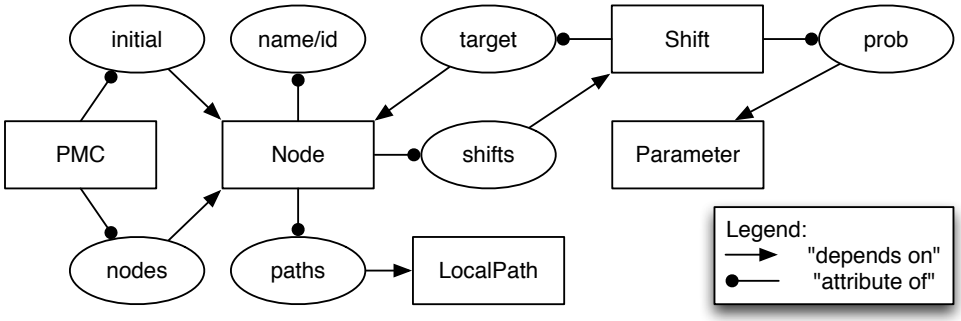


Figure 4.5: Dependency Diagram: Parametric Markov Chain

Each PMC state (notation `Node`) will have as attributes the associated set of local I/O-IPC `paths`. These paths pertain to the components of the scheduled distributed I/O-IPC analyzed. The other attribute of each `Node` is the set of enabled parametric transitions (notation `shifts`).

The PMC transitions (notation `Shift`) are always being addressed from a PMC state attribute. Thus, they are fully described by the additional information of their `target` PMC state and the corresponding parametric probabilistic value (notation `prob`). The probabilistic value of a PMC transition is the result of combining local and global/inteleaving parameters.

Global parameters, PMC state names (notation `name/id`) as well as global constraints are produced using an additional generator associated with the created PMC. Goal states of the PMC are flagged upon their initialization.

4.5 Model Expansion

The PMC resulted from the given basic I/O-IPCs is constructed by a depth-first recursion on its states. The PMC start state from which the recursion is initiated is created from the local paths generated by the start states of the input I/O-IPCs.

The algorithm proceeds by a tangible vs. vanishing case distinction on the `node` (i.e. on its associated distributed I/O-IPC state) as in Algorithm 4.1.

Tangible `PMC` states (seen as their corresponding distributed I/O-IPC state) are expanded like described in Algorithm 4.2. If the check on the time `bound` fails then the `node` is made absorbing and the recursion is halted.

Algorithm 4.1 PMC Construction – Main Recursion: `expand`

Require: node and bound
  if node is tangible **then**
    $\texttt{expand}_t$(node, bound)
  else
    $\texttt{expand}_v$(node, bound)
  end if

On the other hand, if the time bound limit is not reached, the algorithm proceeds by creating `vec`. Each item of the vector `vec` contains all the local paths corresponding to expanding the local path associated with the PMC node.

The possible path expansions are according to the enabled probabilistic transitions for each of the basic I/O-IPC components analyzed.

Algorithm 4.2 Expansion of Tangible PMC States: $\texttt{expand}_t$

Require: node and bound
  if bound $\neq 0$ **then**
    vec := $\emptyset$
    for all i $\in \{1, \ldots, \#\texttt{IOIPCs}\}$ **do**
      vec.add(node.paths[i].prKids)
    end for
    prVec := oneOfEach(vec)
    for all proj $\in$ prVec **do**
      pmc.nodes := pmc.nodes $\cup \{\text{node}_{\text{proj}}\}$
      ($*$ α is the parameters' product needed to expand node into $\text{node}_{\text{proj}}$ $*$)
      node. $\rightarrow$:= node. $\rightarrow \cup \{\overset{\alpha}{\rightarrow} \text{node}_{\text{proj}}\}$
      expand($\text{node}_{\text{proj}}$, bound $-$ 1)
    end for
  else
    $\rightarrow$:=$\rightarrow \cup \{\text{node} \overset{1}{\rightarrow} \text{node}\}$
  end if

The vector prVec is computed according to the self-contained Algorithm 4.3. Its items are ordered sets of local paths. These sets fully describe the projections of the PMC states reachable with a Shift from the analyzed PMC node.

For each projection proj describing the new PMC state $\text{node}_{\text{proj}}$ the following are subsequently performed:

- the set of PMC nodes is enriched with the newly created PMC state $\text{node}_{\text{proj}}$
- the parametric probability to reach $\text{node}_{\text{proj}}$ from node (denoted α) is computed using the information within the local paths describing $\text{node}_{\text{proj}}$

Algorithm 4.3 All ways to pick an Item of every Box out of N given: `oneOfEach`

```
Require: boxes: a size-N vector containing vectors
  result := ∅
  if boxes.size = 1 then
    result.add(boxes[1])
  else
    head := boxes.pop()
    tmp := oneOfEach(boxes)
    for all item ∈ head do
      for all vec ∈ tmp do
        result.add(vec.add(item))
      end for
    end for
  end if
  (* items of the result vector have size N *)
  return result
```

- the set of enabled transitions (`shifts`) for `node` is enriched with the newly created `Shift`: $\xrightarrow{\alpha}$ $\texttt{node}_{\texttt{proj}}$
- the recursion proceeds with $\texttt{node}_{\texttt{proj}}$ and a decreased-by-1 time bound.

Vanishing `PMC` states (seen as their corresponding distributed I/O-IPC state) are expanded like described by Algorithms 4.4 and 4.5.

Algorithm 4.4 is solely a helper subroutine which breaks down the case analysis. It makes possible dealing separately – for each of the basic I/O-IPC components – with expanding local paths of the analyzed PMC state (`node`) corresponding to enabled output transitions.

Algorithm 4.4 Expansion of Vanishing PMC States: $\texttt{expand}_{\texttt{v}}$

```
Require: node and bound
  for all i ∈ {1, ..., #IOIPCs} do
    matchOutputs(node, bound, i, node.paths[i].outKids)
  end for
```

The worker subroutine for performing the expansion of "vanishing" PMC states is described by Algorithm 4.5 and argumented below.

For each `LocalPath` in the i^{th} basic I/O-IPC, all enabled output transitions (and thus possible path expansions) are considered. For each of them a vector of projections `proj` is created which will determine a new `PMC` state.

The local paths relative to the other components $j \in \{1, \ldots, N\},\ j \neq i$ have to be constructed, being part of the projections vector.

It will be tested whether there is any "input transition"-expansion of the `LocalPath` for component *j* matching the output transition of component *i*'s path expansion.

If this is the case then the `LocalPath` for *j*, expanded by the input transition, will be added to the projections vector. Otherwise, the `LocalPath` for *j* will be added to the projections vector as it is.

This is also where input determinism comes into play. In general, it might be the case that, for a given output transition in component *i*, there are multiple matching input transitions enabled for the `LocalPath` of component *j*.

Handling input nondeterminism would require that all these possible choices are considered and projection vectors are created for each matching input. The rest of the algorithm would require no additional changes.

Upon the successful creation of the projections vector `proj`, the new associated `PMC` state will be created and added to the state set of the `PMC`.

The parametric probability to reach $\texttt{node}_{\texttt{proj}}$ from `node` (denoted α) needs also to be computed. Its computation uses the local parameter information within the local paths describing $\texttt{node}_{\texttt{proj}}$ as well as the interleaving parameter information within `node`.

Thereafter, the set of enabled transitions (`shifts`) of the object `node` is enriched with the newly created `Shift` (namely, $\xrightarrow{\alpha} \texttt{node}_{\texttt{proj}}$). The recursion then proceeds with $\texttt{node}_{\texttt{proj}}$ and the same time bound.

Algorithm 4.5 Match Outputs to corresponding Inputs: `matchOutputs`

Require: `node`, `bound`, `i`: IOIPC number,
`outputs`: path expansions from `node` for output transitions of the i^{th} I/O-IPC
for all `out` $\in$ `outputs` **do**
  `proj` := $\emptyset$
  for all `j` $\in \{1, \ldots, \#\texttt{IOIPCs}\}$ **do**
    if `j` = `i` **then**
      `proj`.add(`out`)
    else if $\exists$`in` $\in$ `node`.`paths`[`j`].`inKids` to match `out` **then**
      `proj`.add(`in`)
    else
      `proj`.add(`node`.`paths`[`j`])
    end if
  end for
  `pmc`.`nodes` := `pmc`.`nodes` $\cup$ $\{\texttt{node}_{\texttt{proj}}\}$
  ($*$ α is the parameters' product needed to expand `node` into $\texttt{node}_{\texttt{proj}}$ $*$)
  `node`. $\rightarrow$:= `node`. $\rightarrow$ $\cup \{\xrightarrow{\alpha} \texttt{node}_{\texttt{proj}}\}$
  `expand`($\texttt{node}_{\texttt{proj}}$, `bound`)
end for

Chapter 5

Case Studies

This chapter is used to present five case studies for which the algorithm has been used and their results.

For the first 3 case studies the PARAM tool has been run on a computer with a 3 Ghz processor and 1 GB of memory, while Matlab was run on a computer with two 1.2 Ghz processors and 2 GB of memory. The last 2 case studies were tested by running the unfolder on a computer with two 2.13 Ghz processors and 4 GB of memory.

5.1 Mastermind

In the game of Mastermind [20] one player, the *guesser*, tries to find out a *code*, generated by the other player, the *encoder*. The code consists of a number of tokens of fixed positions, where for each token one color (or other labelling) is chosen out of a pre-specified set. Colors can appear multiple times.

At each round, the guesser guesses a code. This code is then compared to the correct one by the encoder. The encoder answers by telling the guesser:

a) how many tokens were of the correct color and at the correct place

b) how many tokens were *not* at the correct place, but have a corresponding token of the same color in the code.

Assume, for instance, the correct code is $\bullet\circ\star\star$ and the guesser guesses $\bullet\star\circ\circ$. In this case, the encoder answers 1 for a), because of the $\bullet$ at the very left. For b), the encoder answers 2, because for one of the $\circ$ tokens and for the $\star$ token there is a correspondence in the $\bullet\circ\star\star$ correct code.

Notice that the decisions of the encoder during the game are fully deterministic, while the guesser has the choice between all valid codes. We assume that

[0]The Mastermind case study was initially suggested and adapted by Moritz Hahn

the encoder chooses the code probabilistically with a uniform distribution over all options. The goal of the guesser is to find out the code as fast as possible. Our aim is to compute the maximal probability that the guesser correctly guesses the code within t rounds.

However, the latter target is not as clear as it may seem. It could mean to minimize the expected number of rounds [21], to bound the maximal number of rounds [22], or to maximize the probability that the code is broken within a given limit of rounds. All of the previous, except for bounding the maximal number of rounds, depend on the probability with which codes are chosen.

We formalize the game as follows: let n denote the number of tokens and m denote the number of colors of the code. This means there are m^n possible codes.

The symbols $\mathfrak{e}$ and $\mathfrak{g}$ are used as markers to distinguish between tuples which may not be distinguishable otherwise. Define $I_m = \{1, \ldots, m\}$ and let $C_{n,m} = \{\mathfrak{e}\} \times (I_m)^n$ denote the set of possible codes. With $G_n = \{\mathfrak{g}\} \times I_n \times I_n$ we denote the set of all tuples (a, b) of fully (a) and partially (b) correct tokens.

We use $\mu_{code} \in \mathit{Dist}(C_{n,m})$ as the distribution according to which the encoder chooses the code which the guesser is supposed to break.

The guesser is the basic I/O-IPC $\mathcal{G} = \langle \{s_\mathcal{G}, s'_\mathcal{G}, s''_\mathcal{G}, \overline{s}_\mathcal{G}\}, A_\mathcal{G}, \rightarrow_\mathcal{G}, \Rightarrow_\mathcal{G}, s_\mathcal{G} \rangle$ where $A^I_\mathcal{G} = G_n$, $A^O_\mathcal{G} = C_{n,m}$ and $A^{int}_\mathcal{G} = \emptyset$. Interactive transitions of the guesser are given by $\rightarrow_\mathcal{G} = \{(s'_\mathcal{G}, a, s''_\mathcal{G}) \mid a \in A^O_\mathcal{G}\} \cup \{(s''_\mathcal{G}, a, s''_\mathcal{G}) \mid a \in A^I_\mathcal{G} \setminus \{\langle \mathfrak{g}, n, 0 \rangle\}\} \cup \{(s''_\mathcal{G}, \langle \mathfrak{g}, n, 0 \rangle, \overline{s}_\mathcal{G}\}$.

Thus, $\mathcal{G}$ can nondeterministically propose codes and can receive feedback on their correctness. Probabilistic transitions are $\Rightarrow_\mathcal{G} = \{(s_\mathcal{G}, \mu_{s'_\mathcal{G}}), (s''_\mathcal{G}, \mu_{s'_\mathcal{G}})\}$ where $\mu_s(s) = 1$ and $\mu_s(s') = 0$ for $s \neq s'$. The first pair in the probabilistic transition is used to follow a step the encoder needs in order to choose an initial code. The second pair encodes a delay after each guess, to allow to reason about the number of steps needed to guess the code.

The encoder is seen as $\mathcal{E} = \langle \{s_\mathcal{E}\} \cup C_{n,m} \cup (C_{n,m} \times C_{n,m}), A_\mathcal{E}, \rightarrow_\mathcal{E}, \Rightarrow_\mathcal{E}, s_\mathcal{E} \rangle$, where $A^I_\mathcal{E} = C_{n,m}$, $A^O_\mathcal{E} = G_n$ and $A^{int}_\mathcal{E} = \emptyset$. We have $\Rightarrow_\mathcal{E} = \{(s_\mathcal{E}, \mu_{code})\} \cup \{(s, \mu_s) \mid s \in C_{n,m}\}$.

Here, $(s_\mathcal{E}, \mu_{code})$ is used for the probabilistic choice of the code. The function μ_{code} is a distribution over the states in $C_{n,m}$. The rest of $\Rightarrow_\mathcal{E}$ is used to follow the delay transition of the guesser.

Interactive transitions of the encoder are separated $\rightarrow_\mathcal{E} = \rightarrow^I_\mathcal{E} \cup \rightarrow^O_\mathcal{E}$ into two parts. Using $\rightarrow^I_\mathcal{E} = \{(a, b, (a, b)) \mid a, b \in C_{n,m}\}$ the encoder receives the choice of the guesser and using $\rightarrow^O_\mathcal{E} = \{((a, b), f(a, b), a) \mid a, b \in C_{n,m}\}$ the encoder provides answers to the guesser.

The function $f : C_{n,m} \times C_{n,m} \rightarrow G_n$ compares two codes and is defined as $f(a, b) = (\mathfrak{g}, f_1(a, b), f_2(a, b))$ with f_1 and f_2 defined as it follows.

Let $\overline{a} = \langle a_1, \ldots, a_n \rangle$ and $\overline{b} = \langle b_1, \ldots, b_n \rangle$. The function f_1 answers how many tokens were of the correct color and at the correct position:

$$f_1((\mathfrak{e}, \overline{a}), (\mathfrak{e}, \overline{b})) = |A(\overline{a}, \overline{b})|,$$

where $A(\overline{a}, \overline{b}) = \{i \in I_n \mid a_i = b_i\}$ contains the position numbers in which the actual and the guessed code completely agree.

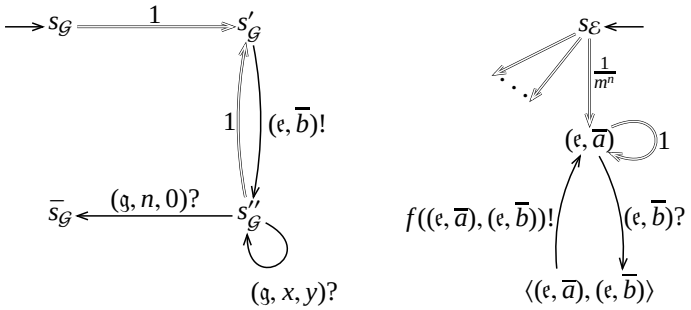
The function f_2 answers how many tokens were not at the correct place, but have a corresponding token of the same color in the code:

$$f_2((\mathfrak{e}, \overline{a}), (\mathfrak{e}, \overline{b})) = \sum_{j=1}^{m} \min\{|B(\overline{a}, \overline{b}, j)|, |B(\overline{b}, \overline{a}, j)|\},$$

where $B(\overline{a}, \overline{b}, j) = \{i \in I_n \mid i \notin A(\overline{a}, \overline{b}) \wedge a_i = j\}$.

To find out about the partially correct guesses, we consider each possible color separately. The total sum of partial guesses is then the sum of partial guesses of the individual colors. We count the number of tokens of a given color at positions which have not been guessed absolutely correctly. On the one hand we count them in the actual code and on the other hand in the code proposed by the guesser.

We have to take the minimum of these two numbers: if they are equal, there is a corresponding token for each token the guesser took (though not at the same position). If the guesser guessed less tokens than exist, only the ones guessed can be considered correct. If he/she guessed more than exist, there remain tokens of which there is no correspondence in the actual code.



The $(\mathfrak{g}, x, y)?$ and $(e, \overline{b})!$ transitions on the left denote sets of transitions quantified over $(x, y) \in I_n^2,\ (x, y) \neq (n, 0)$ and $\overline{b} \in (I_m)^n$.
The state $(\mathfrak{e}, \overline{a})$ is arbitrary (i.e. $\overline{a}$ is arbitrarily fixed) and the $(\mathfrak{e}, \overline{b})?$ transition together with the $\langle(\mathfrak{e}, \overline{a}), (\mathfrak{e}, \overline{b})\rangle$ state are quantified over $\overline{b} \in (I_m)^n$ again.

Figure 5.1: Mastermind Schematic Models of $\mathcal{G}$ (left) and $\mathcal{E}$ (right)

The Mastermind game is the composition $C := \mathcal{G} \| \mathcal{E}$ of the two basic I/O-IPCs. A schematic graphical representation of the game is depicted in Figure 5.1.

Using the tool described in Chapter 4 we can reason about the maximal probability $\sup_\eta Pr(\Diamond^{\leq t} \overline{s}_{\mathcal{G}})$ to break the code within a given number t of guesses.

We consider the set of all distributed schedulers as we obviously want that the guesser uses only local information to make its guesses.

If we were to consider the set of all schedulers, the maximum probability for the guesser to find the code would be 1 for any time-bound t, as for some scheduler the guesser would immediately choose the correct code with probability 1.

Note that it does not make sense to consider strongly distributed schedulers for this case study as it never occurs that the I/O-IPCs $\mathcal{G}$ and $\mathcal{E}$ both have immediate actions enabled. In other words, the players act in turn.

Settings			PMC			PARAM			NLP	
n	m	t	$\#S$	$\#T$	$\#V$	Time	Mem	$\#V$	Time	Pr
2	2	2	197	248	36	0.0492	1.43	17	0.0973	0.750
2	2	3	629	788	148	0.130	2.68	73	0.653	1.00
3	2	2	1545	2000	248	0.276	5.29	93	1.51	0.625
3	2	3	10953	14152	2536	39.8	235	879	1433	1.00
2	3	2	2197	2853	279	0.509	6.14	100	2.15	0.556

Table 5.1: Results of Mastermind Case Study

Results are given in Table 5.1. In addition to the model parameters (n, m), the time bound (t) and the result (Pr) are given. We provide statistics for the various phases of the algorithm. For the unfolded PMC the number of states ($\#S$), transitions ($\#T$), and variables ($\#V$) are given.

For the PARAM tool the time (in seconds) needed to compute the polynomial, the memory (in MB) required, and the number of variables that remain in the resulting polynomial are also given.

Finally we give the time (in seconds) needed for Matlab to optimize the polynomial provided by PARAM under the linear constraints that all scheduler decisions lie between 0 and 1. For this case study the PMC models and linear constraints were generated semi-automically given the parameters n, m, and t.

5.2 Dining Cryptographers

The dining cryptographers problem is a classical anonymity problem [23]. The cryptographers must work together to deduce a particular piece of information using their local knowledge, but at the same time each cryptographers' local knowledge may not be discovered by the others.

The problem can be summarized as follows: three cryptograpers have just finished dining in a restaurant when their waiter arrives to tell them their bill has been paid anonymously. The cryptographers decide they wish to respect the anonimity of the payer, but they wonder if one of the cryptographers has paid or someone

[0]The Dining Cryptographers case study was initially suggested and adapted by Pepijn Crouzen

else. They resolve to use the following protocol to discover whether one of the cryptographers paid, without revealing which one.

Each cryptographer flips a fair coin such that the others cannot see the outcome. Thereafter each of them sees the outcome of his own coin and shows the flipped coin to his right-hand neighbour (actions h_i for heads and t_i for tails). This happens in a fixed order. They all now know the outcome of two coins (for instance, cryptographer one knows the outcome of his own flip and the outcome of that of cryptographer two).

Again in a fixed order, they proclaim whether the two coins were the same or different (actions s_i for same and d_i for different). However, if a cryptographer has paid he/she will *lie* when proclaiming whether the two coins were identical or not.

Now we have that if there is an even number of "different" proclamations, then all of the cryptographers told the truth and it is revealed that someone else paid. If, on the other hand, there is an odd number of "different" proclamations, one of the cryptographers must have paid the bill, but it has been shown that there is be no way for the other two cryptographers to know which one has paid.

In the described model each cryptographer first attempts to guess whether a cryptographer has paid (actions c_i to guess that a cryptographer has paid, n_i if not). In case the cryptographer decides a cryptographer has paid, he guesses which one (action $g_{i,j}$ denotes that cryptographer i guesses cryptographer j has paid).

We depict part of the I/O-IPC models in Figure 5.2. On the right-hand side of the figure we have the I/O-IPC $\mathcal{F}$ that simply decides who paid (actions p_i) and then starts the protocol. Each cryptographer has a probability of $\frac{1}{6}$ to have paid and there is a probability of $\frac{1}{2}$ that none of them has paid.

On the left-hand side of Figure 5.2 we see part of the I/O-IPC $\mathcal{G}_1$ for the first cryptographer (the case when the cryptographer flips heads and has not paid and therefore proclaims the truth).

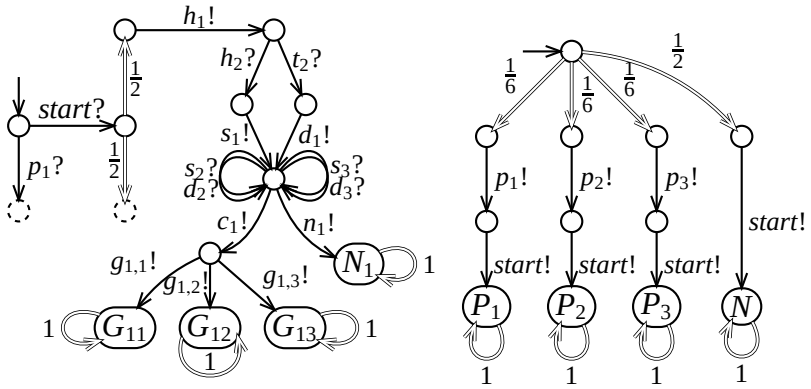


Figure 5.2: Part of the I/O-IPC model $\mathcal{G}_1$ (left) of the first dining cryptographer and the I/O-IPC $\mathcal{F}$ (right) that probabilistically decides who has actually paid.

We can see that a "run" of the distributed I/O-IPC $C = \mathcal{F}||\mathcal{G}_1||\mathcal{G}_2||\mathcal{G}_3$ takes two time-units, since there is one probabilistic step to determine who paid and one probabilistic step where all coins are tossed simultaneously. We are interested in two properties of this algorithm: first, all cryptographers should be able to determine whether someone else has paid or not. We can express this property, for example for the first cryptographer, as a reachability probability property:

$$P(\Diamond^{\leq 2}\{P_1, P_2, P_3\} \times \{G_{11}, G_{12}, G_{13}\} \times S_2 \times S_3 \cup \{N\} \times \{N_1\} \times S_2 \times S_3) = 1. \quad (5.1)$$

Here S_2 and S_3 denote the complete state spaces of the second and third cryptographer I/O-IPCs. For the other cryptographers we find similar reachability probability properties.

Secondly, we must check that the payer remains anonymous. This means that, in the case that a cryptographer pays, the other two cryptographers cannot guess this fact. This can be formulated as a conditional reachability probability:

$$\frac{P(\Diamond^{\leq 2}\{P_2\} \times \{G_{12}\} \times S_2 \times S_3 \cup \{P_3\} \times \{G_{13}\} \times S_2 \times S_3)}{P(\Diamond^{\leq 2}\{P_2, P_3\} \times S_1 \times S_2 \times S_3)} = \frac{1}{2}, \quad (5.2)$$

i.e., the probability that the first cryptographer guesses correctly which other cryptographer has paid, under the condition that one of the other cryptographers has paid is one half.

	PMC			PARAM			NLP	
Property	*#S*	*#T*	*#V*	Time	Mem	*#V*	Time	*Pr*
(5.1)	294	411	97	9.05	4.11	24	0.269	1.00
(5.2), top	382	571	97	9.03	4.73	16	0.171	0.167
(5.2), bottom	200	294	97	8.98	4.14	0	N/A	1/3

Table 5.2: Results of Dining Cryptographers Case Study.

Table 5.2 shows the results for the dining cryptographers case study. We calculate the conditional probability in (5.2) by computing the top and bottom of the fraction separately. We observe that both properties (5.1) and (5.2) are fulfilled. Table 5.2 also lists statistics on the tool performances and model sizes as described for Table 5.1. Note especially that the third reachability probability was computed directly by PARAM. I.e., this probability is independent of the scheduler decisions and PARAM was able to eliminate all variables.

5.3 Randomized Scheduler Example

For the class of strongly distributed schedulers it may be the case that the maximal or minimal reachability probability can not be attained by a deterministic scheduler, i.e. a scheduler always choosing one action/component with probability 1.

To exemplify this situation we use a small example of an I/O-IPC as depicted by Figure 4 in [2] through a PIOTA model. For convenience, the example adapted to the I/O-IPC settings is depicted in Figure 5.3.

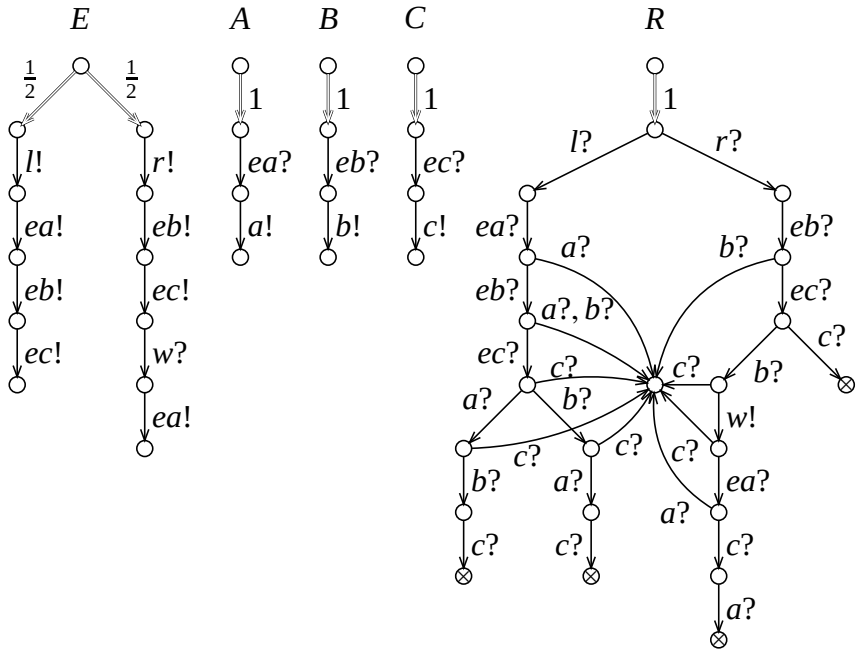


Figure 5.3: The Randomized Scheduler Example

In this example, the maximal probability of reaching a state $\otimes$ for deterministic strongly distributed schedulers is $1/2$. However the scheduler that prioritizes doing an action from E, then from R and then chooses among the rest (i.e. A, B, or C) with uniform probability, yields a probability result of $13/14$.

PMC			PARAM			NLP	
$\#S$	$\#T$	$\#V$	Time	Mem	$\#V$	Time	Pr
13	23	12	0.00396	1.39	11	0.241	$0.545^{\ddagger}$

Table 5.3: Results of Randomized Scheduler Case Study.
(‡ For certain settings, Matlab reports a maximal probability of 0.500)

Table 5.3 shows the result of applying our tool chain to this example. We see that we can find a scheduler with maximal reachability probability 0.545, which is even greater than $13/24$. Note that we can express the maximal reachability probability as a time-bounded property because the example is acyclic.

However, for this case, the Matlab numerical result depends on the initial assignment given to the solver. For certain initial assignments the solver returns a maximal probability of only 0.500. This indicates that further investigation is required in the appropriate nonlinear programming tool for our algorithm.

5.4 Distributed Random Bit Generator

Random bit generators (notation RBGs) are abstract algorithms that produce random sequences of bits. For a very good analysis of existing (pseudo)random number generators and statistical tests for them to [24].

In our current case study RBGs are treated as black boxes. To be specific, a RBG is understood as an algorithm which, after performing some computation, outputs either 0 or 1 with probability $1/2$ (see Figure 5.4). In a sense, RBGs are identical to the coin-flip used early in the thesis.

We propose to combine RBGs in a distributed setting by having the bits produced by different RGBs interleaved. At the same time, we assume that in parallel with the given RBGs there is an "attacker" (much like the coin-guess) who tries to discover the produced bit sequence. We show that by using strongly distributed schedulers the "attacker" can only guess what the sequence produced by the distributed RBG is.

The example proposed is a nice generalization of RBGs to distributed settings and a distributed RBG as the one described could be used by multiple parties who do not trust the randomness of eachother's RBG.

This case study also motivates and hints toward the possiblity of how and why to use distributed and strongly distributed schedulers at the same time in a distributed system.

Namely, we assume that a group of entities trust eachother and thus use a distributed RBG subject to distributed scheduling. We further assume that a new, yet untrusted entity, wants to join the others by extending the distributed RBG.

There is no need to use strongly distributed scheduler restrictions between all entities. Instead, strongly distributed scheduling (and corresponding restrictions) have to be generated only between the new entity and each of the initial ones.

The distributed RBG can be seen as the composition of RBGs $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and an attacker $\mathcal{X}_n$. The attacker $\mathcal{X}_n$ tries to guess the bits output by the other RBGs. We are interesting in finding the probability for the attacker to guess the RBG-produced bits within one of the first t time steps.

Each RBG is a "coin-flip" I/O-IPC, i.e. $\mathcal{P}_i = \langle \{s_0^i, s_1^i, s_2^i\}, \mathcal{A}_i, \rightarrow_i, \Rightarrow_i, s_0^i \rangle$ where $\mathcal{A}_i^O = \{0_i, 1_i\}$ and $\mathcal{A}_i^I = \mathcal{A}_i^{int} = \emptyset$. Probabilistic transitions are given as $\Rightarrow_i = \{(s_0^i, \mu) \mid \mu(s_1^i) = \mu(s_2^i) = 1/2\}$ and interactive transitions are described by $\rightarrow_i = \{(s_1^i, 0_i!, s_0^i), (s_2^i 1_i!, s_0^i)\}$.

The attacker is a generalized "coin-guess": $X_n = \langle \{s_0, s_1\}, \mathcal{A}_\mathcal{X}, \rightarrow_\mathcal{X}, \Rightarrow_\mathcal{X}, g_0 \rangle$ where $\mathcal{A}_\mathcal{X}^O = \{g_x \mid x \in \{0,1\}^n\}$ and $\mathcal{A}_\mathcal{X}^I = \mathcal{A}_\mathcal{X}^{int} = \emptyset$. Probabilistic transitions of the attacker are given by $\Rightarrow_\mathcal{X} = \{(s_0, \mu) \mid \mu(s_1) = 1\}$ while interactive transitions are described by $\rightarrow_\mathcal{X} = \{(s_1, g_x, s_0) \mid x \in \{0,1\}^n\}$.

Notice that for $n = 1$ we find ourselves in the case of the repeated coin flip and guess as depicted in Figure 1.2. The basic I/O-IPCs for the system when $n = 2$

are described in Figure 5.4.

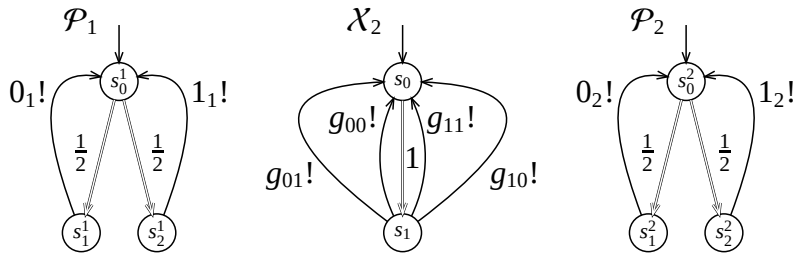


Figure 5.4: Random Bit Generators $\mathcal{P}_1$, $\mathcal{P}_2$ and attacker $\mathcal{X}_2$

Table 5.4 shows the partial results for the distributed random bit generator case study. Computing the reachability probability that within t time steps the attacker guesses at least once the bits produced by the parallelized RBGs can be performed using the PMC generated by the unfolding.

The model for the attacker may be extended by additional components to justify the use of strongly distributed schedulers but, to begin with, the test above (which requires only the use of distributed schedulers) has to be performed.

	Unfolder			PMC		
#RBGs	t	Time	Mem	#S	#T	#V
1	2	0.015	2	81	116	13
1	5	0.548	11.98	5457	7844	713
1	6	2.607	32.2	21841	31396	2793
1	7	28.358	14.35	87377	125604	11049
2	1	0.027	2.66	197	292	35
2	2	1.469	14.16	11605	17840	2223

Table 5.4: Results of the Distributed Random Bit Generator Case Study.

Similar to the previous case studies, statistics on the unfolder's performance and model sizes are given. In addition, the number of RBGs taken in parallel as well as the time (in seconds) and memory needed (in Mb) for the unfolder to generate the PMC are specified.

Another interesting reachability probability which could be investigated, but for which the current implementation is not enhanced enough to see through, is the probability that within at most t time steps, the attacker $\mathcal{X}$ guesses at most x n-tuples of bits.

Also, by taking the interleaved guesses for the bits of the given RBGs in the attacker, instead of the condensed version which tries to guess n-tuples of bits produced by all the RBGs at one step, one could compute (e.g.) the probability for the attacker to guess entirely the bits generated by only one of the given RBGs.

5.5 Car Platooning

As a last case study we consider a simplified version of the car platooning design and verification. The case study was initally introduced by the "Partners for Advanced Transit and Highways (PATH)" [25] at the University of California, Berkeley. More information on car platooning is available in the literature [26, 27]. The main idea of the study is that traffic density on highways can be maximized by merging autonomous cars into platoons.

The adaptation of car platooning to the I/O-IPC settings focuses on finding maximal reachability probabilities for an abstract platoon to be filled with a specified, smaller then its capacity, number of cars.

The behavior of a car will be modeled by an I/O-IPC and a car is seen as being able to have the role of either a free agent, or a platoon member. The platoon is also represented by an I/O-IPC and its capacity is reflected by the models's number of states. The key to platoon verification resides in formal analysis of *merge & split maneouvers* which describe cars joining or leaving the platoon.

The distributed system analyzed is the result of parallelizing a platoon model and a number of I/O-IPCs car models. Although various tests can be performed, the initial test scenario we used requires to compute the maximal probability that a platoon of capacity N consists of $k \leq N$ cars within at most t timed steps.

The formal description of the platoon and car I/O-IPCs are given below. The I/O-IPC model of a car and of a platoon of size N are depicted in Figure 5.5.

The behavior of a car is given as the I/O-IPC $C_i = \langle S_i, \mathcal{A}_i, \rightarrow_i, \Rightarrow_i, \mathtt{free_i}\rangle$ where the I/O-IPC states are $S_i = \{\mathtt{free_i}, \mathtt{busy_i}, \mathtt{x_i}, \mathtt{x'_i}, \mathtt{x''_i}, \bar{\mathtt{x}}_\mathtt{i}, \mathtt{y_i}, \mathtt{y'_i}, \mathtt{y''_i}, \bar{\mathtt{y}}_\mathtt{i}\}$, the non-timed transitions are described by

$$\begin{aligned}\rightarrow_i = {} & \{(\mathtt{free_i}, \mathtt{m_i}!, \mathtt{x_i}), (\mathtt{x_i}, \mathtt{ok_i}, \mathtt{x'_i}), (\mathtt{x'_i}, \mathtt{no}!, \mathtt{x''_i}), (\mathtt{x_i}, \mathtt{no}?, \bar{\mathtt{x}}_\mathtt{i}), (\mathtt{free_i}, \mathtt{no}?, \bar{\mathtt{x}}_\mathtt{i})\} \\ & \cup \{(\mathtt{busy_i}, \mathtt{s_i}!, \mathtt{y_i}), (\mathtt{y_i}, \mathtt{ok_i}, \mathtt{y'_i}), (\mathtt{y'_i}, \mathtt{no}!, \mathtt{y''_i}), (\mathtt{y_i}, \mathtt{no}?, \bar{\mathtt{y}}_\mathtt{i}), (\mathtt{busy_i}, \mathtt{no}?, \bar{\mathtt{y}}_\mathtt{i})\}\end{aligned}$$

and the probabilistic transitions are given by

$$\Rightarrow_i = \{(\bar{\mathtt{x}}_\mathtt{i}, 1, \mathtt{free_i}), (\bar{\mathtt{y}}_\mathtt{i}, 1, \mathtt{busy_i}), (\mathtt{x''_i}, 1, \mathtt{busy_i}), (\mathtt{y''_i}, 1, \mathtt{free_i})\}.$$

The states $\mathtt{free_i}$ and $\mathtt{busy_i}$ mean that the car is either a free agent or part of a platoon. The actions $\mathtt{m_i}$ and $\mathtt{s_i}$ signal that the car wants to merge with the platoon or split from it. The $\mathtt{no}!$ output signal is produced by the car which is given permission ($\mathtt{ok}?$) to join or leave the platoon.

The platoon behavior is given by the I/O-IPC $\mathcal{P} = \langle S_\mathcal{P}, \mathcal{A}_\mathcal{P}, \rightarrow_\mathcal{P}, \Rightarrow_\mathcal{P}, \mathtt{p_0}\rangle$ where $S_\mathcal{P} = \{\mathtt{p_i} \mid i \in \overline{0, N}\} \cup \{+_{\mathtt{i,i+1}}, +'_{\mathtt{i,i+1}}, -_{\mathtt{i+1,i}}, -'_{\mathtt{i+1,i}} \mid i \in \overline{0, N-1}\}$, the non-timed transitions are described by

$$\begin{aligned}\rightarrow_\mathcal{P} = {} & \{(\mathtt{p_i}, \mathtt{m_i}?, +_{\mathtt{i,i+1}}), (+_{\mathtt{i,i+1}}, \mathtt{ok_i}!, +'_{\mathtt{i,i+1}}) \mid i \in \overline{0, N-1}\} \\ & \cup \{(\mathtt{p_{i+1}}, \mathtt{s_i}?, -_{\mathtt{i+1,i}}), (-_{\mathtt{i+1,i}}, \mathtt{ok_i}!, -'_{\mathtt{i+1,i}}) \mid i \in \overline{0, N-1}\}\end{aligned}$$

and $\Rightarrow_{\mathcal{P}} = \{(+'_{i,i+1}, 1, p_{i+1}), (-'_{i+1,i}, 1, p_i) \mid i \in \overline{0, N-1}\}$ represents the probabilistic transitions of the model.

Each p_i state of the platoon denotes that there are $i \leq N$ cars in the platoon. The platoon listens to merging ($m_i?$) and splitting ($s_i?$) signals of the cars and each time it responds to the signal of one of them.

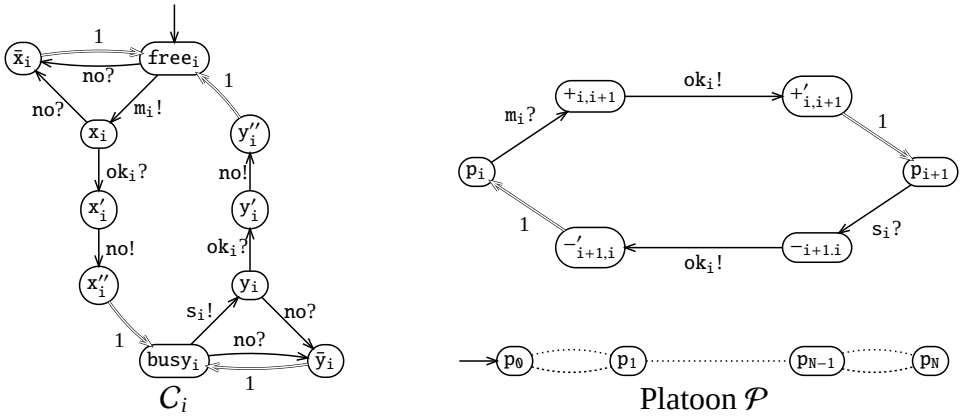


Figure 5.5: I/O-IPC models of a Car (C_i) and a Schematic Platoon of capacity N

Table 5.5 shows partial results for the initial test scenario used for the car platoon case study. The reachability probability which the system may be given to compute is the maximal probability that within t time steps the car platoon consists of precisely k cars. The platoon capacity is taken to be the same as the number of cars of the distributed model.

		Unfolder			PMC		
k	N	t	Time	Mem	#S	#T	#V
2	2	1	0.043	2.33	119	154	35
2	2	2	0.057	3.66	443	568	143
2	2	3	0.366	11.98	2279	2944	683
2	2	4	1.838	12.65	8111	10396	2627
2	2	5	32.585	40.11	41159	53164	12347
2	3	1	0.492	13.99	2788	3876	1088
2	3	2	69.606	44.53	33280	45984	13914

Table 5.5: Results of the Car Platoon Case Study.

Similar to the previous case studies, statistics on the unfolder's performance and model sizes are given. In addition, the number of cars taken in parallel and the capacity of the platoon (both N), the fill-quota against which the tests are performed (k), as well as the time (in seconds) and memory needed (in Mb) for the unfolder to generate the PMC are specified.

For the car platoon study, another interesting test we thought of was to determine the maximal probability that, starting from a platoon consisting already of a number k_0 of cars, the platoon would not decrease under $k_0 - k$ nor increase over $k_0 + k$ cars.

The differences for the platoon and car models in this scenario are small compared to the previous. The initial state of the platoon has to be set to p_{k_0} and k_0 cars must have their `busy_`, instead of their `free_` states marked as initial.

The testing for this study settings however had to be postponed, as further improvements in the implementation (especially for the deployment of strongly distributed schedulers) are needed beforehand.

Chapter 6

Conclusions

Although various model checkers for verifying reachability of distributed, nondeterministic and probabilistic systems exist, none of them has the built-in means of ruling out unrealistic behavior due to inadequate scheduling.

The main contribution of the described work consists in an automated method for determining extremal time-bounded reachability probabilities of distributed I/O-IPCs. The method used is based on reformulating the problem as a polynomial optimization problem under linear and – for strongly distributed schedulers – polynomial constraints.

Both restricting interactive probabilistic chains to I/O-IPCs and adapting the distributed (and strongly distributed) schedulers to I/O-IPCs had to be formalized. Providing the "I/O-IPC to PMC" model construction and proving the equivalence of the reachability properties analyzed for the two dependent models has been the most challenging.

The main drawback of the presented approach is that the generated unfolded parametric Markov model grows exponentially with the size of the original model and the specified time bound.

However, to our knowledge, there is no other algorithm able to compute extremal reachability probabilities of distributed models under (strongly) distributed schedulers out there.

6.1 Related Work

The problem that global schedulers may be too optimistic or pessimistic in the verification of distributed, probablistic, and nondeterministic systems has been noted in several different settings [2, 28, 29, 30].

One approach to resolve the issue is to use *partial-information* schedulers [31].

Using partial-information schedulers allows the hiding of information that a global scheduler should not realistically use.

However, this approach still assumes there is only one global scheduler, instead of several local schedulers as presented in this thesis. For the class of memoryless partial-information schedulers, the extremal long-run average outcomes of tasks can be calculated by reformulating the problem as a non-linear programming problem [31].

A testing preorder for distributed models with probabilistic and nondeterministic choices has been suggested which is aimed at realistically representing the power of schedulers in a distributed setting [32]. In this context, reachability probabilities are defined in a similar way as in this paper, but no algorithm to compute extremal probabilities or to compute the preorder are given.

It would be interesting to study if the preorder [32] indeed preserves extremal time-bounded reachability probabilities when lifted to the setting of I/O-IPCs.

6.2 Future Work

A prototype implementation of the “unfolder”, the missing link in the toolchain, has been developed but it is still subject to changes. Some study directions could be followed for both algorithmic and implementation improvements.

As pointed out by the “Distributed Random Bit Generator” case study, the use of a combination of distributed and strongly distributed schedulers may have practical applicability. In general, adapting various probabilistic and nondeterministic distributed models to the I/O-IPC settings will provide more insight in the area.

It may also be investigated if special purpose algorithms can be used for the specific type of nonlinear programming problems encountered. Further, extending the algorithm for handling (I/O-IPC specific) Zeno behavior could be tackled.

Memory-usage may be optimized by using the fact that only polynomial functions (instead of their rational counterpart) are needed. Implementation-wise, input nondeterminism, transitions labeled by internal actions (and thus action hiding) are not yet integrated in the prototype tool.

Producing the parametric reachability probability can also be shifted to the unfolder. This would be useful since the expansion of the distributed state space is already performed by the unfolder.

On a high level, the limits of I/O-IPCs versus PIOTAs can also be looked into. This might lead to proving the equivalence of the formalisms, the inclusion of one into another or their incomparability.

Bibliography

[1] Andrea Bianco and Luca de Alfaro. Model Checking of Probabalistic and Nondeterministic Systems. In *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 499–513. Springer-Verlag, 1995.

[2] Sergio Giro and P. R. D'Argenio. On the Expressive Power of Schedulers in Distributed Probabilistic Systems. *Electronic Notes in Theoretical Computer Science*, 253(3):45–71, 2009.

[3] Sergio Giro and Pedro R. D'Argenio. Quantitative Model Checking revisited: neither Decidable nor Approximable. In Jean-François Raskin and P. S. Thiagarajan, editors, *FORMATS*, volume 4763 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2007.

[4] Ana Sokolova and Erik. P. De Vink. Probabilistic Automata: System Types, Parallel Composition and Comparison. In *Validation of Stochastic Systems: A Guide to Current Research*, pages 1–43. Springer, 2004.

[5] Mariëlle Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-Time and Parametric Systems*. PhD thesis, University of Nijmegen, Netherlands, 2002. Available via `http://www.soe.ucsc.edu/~marielle`.

[6] Georgel Calin, Pepijn Crouzen, Ernst Moritz Hahn, Pedro D'Argenio, and Lijun Zhang. Time-Bounded Reachability in Distributed Input/Output Interactive Probabilistic Chains. In *SPIN*, pages 193–211, 2010.

[7] Georgel Calin, Pepijn Crouzen, Ernst Moritz Hahn, Pedro D'Argenio, and Lijun Zhang. Time-Bounded Reachability in Distributed Input/Output Interactive Probabilistic Chains. Reports of SFB/TR 14 AVACS 64, SFB/TR 14 AVACS, June 2010.

[8] Holger Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[9] Nicolas Coste, Holger Hermanns, Etienne Lantreibecq, and Wendelin Serwe. Towards Performance Prediction of Compositional Models in Industrial GALS Designs. In *Computer Aided Verification*, pages 204–218, 2009.

[10] Hans A. Hansson. *Time and Probability in Formal Design of Distributed Systems*. Elsevier Science Inc., 1994.

[11] Xavier Nicollin and Joseph Sifakis. An Overview and Synthesis on Timed Process Algebras. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 526–548, London, UK, 1992. Springer-Verlag.

[12] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. A Compositional Semantics for Dynamic Fault Trees in Terms of Interactive Markov Chains. In *ATVA*, pages 441–456, 2007.

[13] Nicolas Coste, Hubert Garavel, Holger Hermanns, Richard Hersemeule, Yvain Thonnart, and Meriem Zidouni. Quantitative Evaluation in Embedded System Design: Validation of Multiprocessor Multithreaded Architectures. In *DATE*, pages 88–89, 2008.

[14] Lijun Zhang and Martin R. Neuhäußer. Model Checking Interactive Markov Chains. In *TACAS*, pages 53–68, 2010.

[15] Conrado Daws. Symbolic and Parametric Model Checking of Discrete-Time Markov Chains. In *ICTAC*, pages 280–294, 2004.

[16] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic Reachability for Parametric Markov Models. *STTT*, page N/A, 2010.

[17] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. PARAM: A Model Checker for Parametric Markov Models. In *CAV*, 2010.

[18] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, London, 1981.

[19] S.P. Han. A Globally Convergent Method for Nonlinear Programming. *Journal of Optimization Theory and Applications*, 22, 1977.

[20] Leslie H. Ault. *Das Mastermind-Handbuch*. Ravensburger Buchverlag, 1982.

[21] Kenji Koyama and Tony W. Lai. An optimal Mastermind strategy. *J. Recr. Math.*, 25:251–256, 1993.

[22] Donald E. Knuth. The Computer as Master Mind. *J. Recr. Math.*, 9:1–6, 1976.

[23] D. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[24] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Third edition, 1997.

[25] Partners for Advanced Transit and Highways. http://www.path.berkeley.edu/.

[26] Jorg Bauer, Ina Schaefer, Tobe Toben, and Bernd Westphal. Specification and Verification of Dynamic Communication Systems. In *Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, pages 189–200, Washington, DC, USA, 2006. IEEE Computer Society.

[27] Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog, Andreas Podelski, and Reinhard Wilhelm. SFB/TR 14 AVACS – Automatic Verification and Analysis of Complex Systems. *IT – Information Technology*, 49(2):118–126, 2007.

[28] G. Lowe. Representing Nondeterministic and Probabilistic Behaviour in Reactive Processes. Technical Report PRG-TR-11-93, Oxford Univ. Comp. Labs, 1993.

[29] C. Morgan, A. McIver, K. Seidel, and M. Massink. Refinement-oriented Probability for CSP. *Formal Aspects of Computing*, 8:617–647, 1996.

[30] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-time Systems*. PhD thesis, MIT, 1995.

[31] Luca de Alfaro. The Verification of Probabilistic Systems under Memoryless partial–information Policies is Hard. In *Proceedings of the Workshop on Probabilistic Methods in Verification*, 1999.

[32] Sonja Georgievska and Suzana Andova. Retaining the Probabilities in Probabilistic Testing Theory. In *FOSSACS*, pages 79–93, 2010.