QUANTITATIVE MODEL CHECKING OF DISTRIBUTED PROBABILISTIC SYSTEMS

RATUL SAHA

(B.Sc in Mathematics and Computer Science,
Chennai Mathematical Institute, India.)

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE

2017

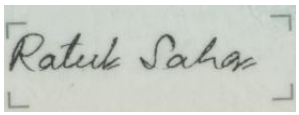
Supervisor:
Professor David Samuel Rosenblum

Examiners:
Associate Professor Chin Wei Ngan
Professor Dong Jin Song
Professor Joost-Pieter Katoen,
RWTH Aachen University

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.



Ratul Saha

August 2017

Acknowledgements

I would like to thank my advisors, Prof. P S Thiagarajan and Prof. David S Rosenblum, for their guidance and support throughout my candidature. My sincere thanks goes to Prof. Madhavan Mukund, for guiding me through since my undergraduate.

I am grateful to Prof. Javier Esparza for the collaboration and his invaluable suggestions that shaped my thesis. I would also like to thank Prof. Esparza and Philipp Hoffmann for the warm hospitality in TU Munich, Germany.

My sincere gratitude goes to Dr. R P Jagadeesh Chandra Bose for the collaboration, and shedding a light in the application area of business processes. I am thankful to Dr. Ansuman Banerjee for introducing me to Dr. Bose, and his invitation to visit ISI Kolkata. Heartfelt thanks goes to Dr. S Akshay for the invitation to visit IIT Bombay, India and later the Mysore Workshop. I have also enjoyed the discussions with Prof. Supratik Chakraborty, Dr. Krishna S, Dr. Amit K Dhar, and Prof. Joost-Pieter Katoen – and I thank them for their time.

I would also like to thank my friends Ramanathan, Sucheendra, Benjamin, Charlie, Narmada, Akshay, Abha, Loi, Shweta, Shruti, Inian. My thanks goes to the deans office and especially Loo Line Fong for their administrative support.

My parents and their support have always been a major drive behind my work. My love to Sharmistha for supporting me through thick and thin for what has been almost a decade now. They are what mattered the most.

List of Figures

Contents

Summary

We study formal models for distributed probabilistic systems to facilitate quantitative analysis using simulation-based techniques.

Real-life systems are inherently large, distributed in nature and exhibit quantifiable aspects such as time and cost. Their dynamics also include uncertainty, which can be measurable or non-measurable. It is very hard to analyze such systems at scale.

In this thesis, we present Distributed Stochastic Model (DSM), a framework for modeling real-life distributed systems, with an analysis paradigm of simulation-based model checking techniques. The design is along the theme of asynchronous transition systems, extended with quantitative aspects.

We first restrict our framework to exclude nondeterminism and illustrate how that paves the way for a succinct distributed representation of finite-state Markov chains. We define both interleaved and non-interleaved semantics for such systems and define a probability measure over both of them. This allows for a simpler and more scalable technique for statistical model checking of Markov chains with inherent concurrency. We then present experimental results on a couple of case studies on distributed probabilistic algorithms.

In the next part of the thesis, we show the behavior of a real-time distributed probabilistic system where atomic events take a fixed duration of time and non-negative cost. We define a highly expressive finite-memory scheduler that also respects the concurrency present in the system, and achieve a semantic association with a countably-infinite Markov chain under such a scheduler. This allows for an approximate verification procedure

using statistical model checking.

As an application, we illustrate resource-constrained processes that can be used to model the workflow of operational processes. We then use our framework for modeling and performance evaluation of such systems. This improves the state-of-the-art simulation techniques for process analysis by providing rigorous sample size analysis and provable error bounds. The proof-of-concept of our technique is presented with a workflow of the loan/overdraft process of a real Dutch bank. The result illustrates how predictive analysis can be performed with bounded error under a varied set of Key Performance Indicators (KPIs) of a business.

Chapter 1

Introduction

Complex systems are prevalent in modern society. From software and hardware to other real-life processes, the systems are often inherently large, distributed in nature, and exhibit quantifiable qualities. They typically consist of independent sub-systems that interact with each other to dictate their dynamics. The interactions in such a concurrent system are identified as atomic events. On top of that, real-life systems involve both measurable and non-measurable uncertainty. It is a common practice to characterize measurable uncertainty by associating probabilities to future events, while non-measurable uncertainty is modeled as nondeterminism that is resolved by an outsider scheduler. Other than probabilities, the most common quantitative aspects of real-life systems are cost and time. An event incurs some cost and takes some time to complete. Both cost and time can be modeled as non-negative real values, however, their accumulation over the dynamics of the system is incommensurable. While the total cost of a set of events is simply the sum of their individual costs, the total time is dependent on the concurrency among the events.

Our increasing reliance on such systems mandates not only ensuring cor-

rectness of the system but also rigorous quantitative analysis. For example for a pizza company preparing and delivering hundreds of pizzas everyday, it is not only important to ensure that every pizza ordered eventually gets delivered, but also analyze claims such as "most of the orders in a given day are delivered within the deadline". The most common procedure for such analysis is *testing* (see [BJK+05] for a detailed discussion). After the system is designed, it is tested for a desired quantitative property against a finite set of test cases. While test cases chosen with sufficient domain expertise have shown effectiveness, it is irrefutable that testing does not have any guarantee of success across all possible behaviors of the system.

On the other end of the spectrum of the analysis paradigm, *formal methods* use mathematical techniques to verify whether a system conforms to a certain property for all possible outcomes. Within the rich history of formal methods, an important sub-discipline is *model checking* [CGP99], the focus of this thesis.

Traditionally, model checking is carried out in the following fashion. First, the system is mathematically modeled with appropriate abstraction, e.g. as a state-transition graph. Then the desirable property is formally specified in a logic whose semantics is supported by the model abstraction, e.g. a suitable variation of temporal logic. Finally, the property is verified for all possible behaviors of the model using a suitable automated algorithm. This approach has been extensively extended to accommodate for different models and analysis techniques in last 30 years (see [GV08, BK08] for an overview).

The core question in model checking is the following: is a given property *definitely* satisfied for *all possible* behaviors of the given system? For systems

with measurable uncertainty, one can transcend qualitative questions and ask: is a given property satisfied *within a probability bound* for all possible behaviors of the given system? Such questions are the foundation of quantitative analysis using *probabilistic model checking*, a strong area of research that has recently emerged with model checking techniques for stochastic models. A significant amount of research [BCHG$^+$97, HKNP06, BK08, Kat16] has been done on exact probabilistic verification – algorithms where the property is checked to be satisfied for all possible behaviors of the given system.

The genesis of model checking was the need to verify correctness of concurrent software programs. With a growing palette of analysis techniques for a variety of models, there is now scope for using model checking techniques for analysis and performance evaluation of many real-life systems [BHHK10, AHV15]. However, the issue is multifaceted. One should be careful in defining a powerful semantics to capture the system dynamics. For example, for distributed systems including probabilistic events, a well-defined measure across the set of outcomes is required, but not necessarily obvious to come up with. On the other hand, the algorithms should ideally be easy to implement and optimize. For many numerical algorithms in model checking, specification of the model in a rigid language is hard for an industry professional. Last but not the least, quantitative analysis algorithms must allow hyper-scalability. While proving correctness is often feasible in a small-scale system with suitable imposed abstraction, scalability has a direct impact on the performance evaluation of real-life systems. For large distributed systems, numerical algorithms often suffers from the infamous state-explosion problem. Even for polynomial algorithms, as sys-

tems grow, the computational complexity of the model checking procedure becomes intractable.

In some performance-driven areas of application (e.g. operational processes) that are not safety-critical in nature, it is acceptable to provide erroneous results of the model checking procedure as long as the error is sufficiently small and provably bounded. This introduced approximate simulation based techniques [You04, YS06], a complementary approach that lies between testing and probabilistic model checking. Instead of using numerical algorithms that take into account the entire state space, the model is simulated for finitely many executions and hypothesis testing is used to infer if the collected samples provide statistical evidence for the satisfaction of the specification. This approach is called *statistical model checking*, and is increasingly popular with verifying cyber-physical systems [CZ11], mixed-signal circuits [CDL10], and biology systems [PGL+13, CFL+08].

To formally specify the properties of interest, we use linear temporal logic [Pnu77]. Temporal operators are provided for describing events along a single future and formulae are designed to provide an implicit universal quantification over all possible futures. Depending on the model specification, we may use different variations of linear temporal logic. For example, when using simulation techniques, we may restrict the logic to a bounded variation to accommodate specification of finite-length samples.

In this thesis, we focus on quantitative analysis of distributed systems using linear temporal logic. It is hard to provide efficient verification algorithms for distributed models that involves probability, nondeterminism, real-time and cost. Also, a generalized solution is often futile and an overkill when applied to real-life systems. An alternate approach is to achieve tar-

geted domain-specific solutions that restrict the most generic model and develop efficient techniques that answer questions relevant to a particular application domain. As George Box said "All models are wrong but some are useful" [Box79], we focus on the efficiency, simplicity, and usefulness of the variations of distributed stochastic models.

1.1 Contribution of the Thesis

On the eve of 25 years of model checking, Edmund Clarke identified probabilistic model checking as an important area ripe for major breakthrough [Clao8]. Since then, quite a number of techniques and tools has been developed for both exact and approximate verification of probabilistic systems. A recent promising direction is to transcend beyond formally verifying probabilistic correctness and use model checking techniques for performance evaluation of real-life systems [BHHK10, AHV15]. This thesis is an attempt of advancement towards this direction for distributed systems.

- We propose a general framework for distributed stochastic models. It can be adopted to model various real-life systems that exhibit concurrency, probability and various quantitative aspects such as time and cost.

- We first restrict the model to exclude nondeterminism and showcase how that paves the way for a succinct distributed representation of finite state Markov chains. We provide both interleaved and non-interleaved semantics for such systems and develop probability measure over both of them. This allows for simpler and more scalable technique for statistical model checking of Markov chains with inher-

ent concurrency. We present the quantitative analysis of a couple of case studies on distributed algorithms.

- We then show how a real-time distributed probabilistic system behaves where the atomic events take a fixed duration of time. We define finite-memory schedulers that respect concurrency and propose an approximate verification procedure using statistical model checking.

- We illustrate resource-constrained processes that can be used to model the workflow of operational processes. We then discuss how to model such systems using our framework. This improves the state-of-the-art statistical methods for performance evaluation for resource-constrained processes by providing rigorous sample size analysis and provable error bounds. We show our technique on the workflow of the loan/overdraft process of a real Dutch bank.

1.2 Outline of the Thesis

The thesis is organized as follows:

Chapter 2 discusses the preliminaries on Markov models, which is a prerequisite to understand the underlying distributed model in subsequent chapters. It also discusses the basics of probability spaces, and for Markov chain in general, which is a required understanding for the discussion in Chapter 4. Then, we present linear temporal properties, which is the base for the property specifications used in the thesis throughout. As a modeling technique, we then detail statistical model checking, which is later used in the rest of the thesis.

Chapter 3 presents Distributed Stochastic Model (DSM), the framework

for modeling real-life probabilistic systems. We discuss various modeling formalisms and their scope of application with respect to quantitative analysis, before extending asynchronous transition systems to define DSM, the bedrock of the rest of the thesis. We also discuss well-known properties such as conflict and confusion that frequents in concurrency theory in reference with our framework.

Chapter 4 then restricts the nondeterminism that exists in our model and shows how we can achieve a succinct distributed representation of Markov chains. We also provide experimental results on two probabilistic distributed algorithms.

Chapter 5 presents resource-constrained processes, and how they can be used as a clean representation of complicated operational processes. We then showcase that our framework, coupled with fixed-duration real-time can facilitate powerful schedulers that respects concurrency. Then we use this model to represent resource-constrained processes and showcase novel simulation-based analysis using data from the operations of a Dutch bank.

Chapter 6 then concludes the thesis, with a direction towards possible future work.

1.3 Declaration

Major portions of this thesis are based on the following works:

- Ratul Saha, Javier Esparza, Sumit Kumar Jha, Madhavan Mukund, and P. S. Thiagarajan. *Distributed Markov chains*. In Proc. of VMCAI 2015 [SEJ+15].

- Ratul Saha, Madhavan Mukund, and R. P. Jagadeesh Chandra Bose.

Time-bounded statistical analysis of resource-constrained business processes with distributed probabilistic systems. In Proc. of SETTA 2016 [SMB16].

- Javier Esparza, Philipp Hoffmann, and Ratul Saha. *Polynomial analysis algorithms for free choice probabilistic workflow nets*. In Proc. of QEST 2016 [EHS16].

An extended version of [EHS16] is also going to appear in the Performance Evaluation journal. The names of the authors in [EHS16] are listed alphabetically.

Chapter 2

Preliminaries

In this chapter, we briefly establish some basic definitions and results that are needed throughout the thesis. We start with Markov models and probability spaces over Markov models. A thorough discussion can be found at [BK08]. We then discuss Statistical Model Checking (SMC), an approximate verification technique well-suited for such models. For a comprehensive overview, we refer to [LDB10].

2.1 Markov Models

Markov models are a popular modeling formalism for systems featuring random phenomena and time. Such models consist of a state space, which is assumed to be finite throughout the thesis. The dynamics of such a system is captured through transitions among the states. Such models exhibit the *Markov property*: the probability of a transition depend only on the state attained by the previous event. Thus a probability distribution is attached to each state, which governs the outgoing transitions from that state. On top of probabilities, a source of uncertainty in systems is nondeterminism. A

finite number of probability distributions are associated with each state. At any given state, an external scheduler is assigned to choose a distribution, which then governs the next set of states.

The two most widely used Markov models are Markov chains [Nor98]—a purely probabilistic model—and Markov Decision Processes [Put94]—a model with both probability and nondeterminism. The flow of time, whenever stated in the thesis, is assumed to be approximated in discrete steps.

Definition 1 (Discrete-time Markov chains)**.** *Let S be a set of states. A* discrete-time Markov chain *is defined as $\mathcal{M} = (S, P, s^{in})$ where P is a probability distributed over $S \times S$ such that for all states s, $\sum_{s' \in S} P(s, s') = 1$ and $s^{in} \in S$ is the initial state.*

Definition 2 (Markov Decision Processes)**.** *Let S be a set of states. Let $dist(S)$ denote the set of probability distributions over S. A (discrete-time)* Markov Decision Process (MDP) *is defined as $\mathcal{M} = (S, s^{in}, Step)$, where s^{in} is the initial state and $Step : S \to 2^{dist(S)}$ is the transition function.*

At a state s of a Markov chain, the next state is chosen according to the probability distribution $P(s)$. However, at a state s of a Markov decision process, a probability distribution $\mu \in Step(s)$ is chosen nondeterministically and then the next state is chosen according to μ.

Both Markov chains and Markov decision processes have been well-studied and a plethora of quantitative analysis techniques have been developed. However, when using Markov models as semantics, we should be cautious about providing a formalization of the probabilities of the runs of system. This formalization is based on the theory of probability spaces and σ-algebras.

2.2 Probability Spaces

Let us consider a random experiment where exactly one *outcome* from a nonempty set *Outc* occurs. A σ-algebra is a pair $(Outc, \mathcal{C})$ where $\mathcal{C} \subseteq 2^{Outc}$ is a set consisting of subsets of *Outc* such that the following is true:

- $\varnothing \in \mathcal{C}$,
- if $E \in \mathcal{C}$, then $Outc \setminus E \in \mathcal{C}$,
- if $E_1, E_2, \cdots \in \mathcal{C}$, then $\cup_{i \geq 1} E_i \in \mathcal{C}$.

We note that $\mathcal{C}$ contains *Outc* and is closed under complementation and countable unions. A σ-algebra represents a collection of experiment outcomes. Often when *Outc* is fixed, $\mathcal{C}$ is called the σ-algebra.

We note that the powerset 2^{Outc} of *Outc* is a σ-algebra and the intersection of σ-algebras is a σ-algebra. Hence, for each set *Outc* and each subset $\Pi \subseteq 2^{Outc}$, there exists a smallest σ-algebra that contains Π.

Definition 3. *For a given $\Pi \subseteq 2^{Outc}$, $\mathcal{C}_\pi$ is the σ-algebra* generated *by Π defined as follows $\mathcal{C}_\Pi = \bigcap_{\mathcal{C}} \mathcal{C}$, where $\mathcal{C}$ ranges over all σ-algebras on Outc that contain Π. We call Π the* basis *for $\mathcal{C}_\Pi$.*

In general, it is not possible to assign probability to arbitrary subsets of a set. We define *probability measure* to assign probabilities to elements of a σ-algebra.

Definition 4 (Probability measure and probability space)**.** *A probability measure on $(Outc, \mathcal{C})$ is a function $Pr : \mathcal{C} \to [0, 1]$ such that:*

- $Pr(Outc) = 1$,

- *if* $(E_i)_{i\geq 1}$ *is a family of pairwise disjoint events* $E_i \in \mathcal{C}$*, then*

$$Pr(\cup_{i\geq 1} E_n) = \sum_{i\geq 1} Pr(E_i).$$

A probability space is a σ*-algebra equipped with a probability measure, i.e.* $(Outc, \mathcal{C}, Pr)$.

The value $Pr(E)$ is called the probability measure of E, or simply the probability of E. In the context of probability measures, the elements of $\mathcal{C}$ are called to be *measurable*. In other words, measurability of a set $E \subseteq Outc$ means that $E \in \mathcal{C}$ and hence a valid probability measure of E is established. When analyzing any probabilistic systems, it is important to clearly identify the set of outcomes and establish a valid probability space.

2.2.1 Probability Space for Markov Chains

We now discuss the probability space over Markov chains. The set of infinite runs serves as the set of outcomes. The set of runs of a Markov chain $\mathcal{M}$ is defined as the infinite sequences $s_0 s_1 s_2 \cdots \in S^\omega$ such that $Pr(s_i, s_{i+1}) > 0$ for all $i \geq 0$. The σ-algebra associated with $\mathcal{M}$ is generated by the *cylinder sets* spanned by the finite path segments in $\mathcal{M}$.

The *cylinder set* of a finite path $\widehat{\pi}$ is defined as $Cyl(\widehat{\pi})$, the set of all infinite paths that start with $\widehat{\pi}$. The cylinder sets serve as basis events of the σ-algebra $\mathcal{C}^{\mathcal{M}}$ associated with the Markov chain $\mathcal{M}$. We define $\mathcal{C}^{\mathcal{M}}$ to be the smallest σ-algebra that contains all cylinder sets $Cyl(\pi)$ where π ranges over all finite path fragments in $\mathcal{M}$.

We define the probability measure $Pr^{\mathcal{M}}$ on the σ-algebra $\mathcal{C}^{\mathcal{M}}$ as probability function over the cylinder sets:

$$Pr^{\mathcal{M}}(Cyl(s_0 \dots s_n)) = \prod_{0 \leq i < n} P(s_i, s_{i+1}).$$

The probability of a cylinder generated by a run is defined as the product of the probabilities of consecutive state transitions along the path. For runs of length 0, we define $P(s_0) = 1$. The uniqueness of the probability measure can be found in the classical literature (e.g. see [Felo8]).

2.3 Specifying Properties

Throughout the thesis, we use *linear temporal logic (LTL)* [Pnu77] to specify properties of the Markov models. We assume a set of atomic propositions AP. The syntax of LTL formulae over AP is defined as follows:

$$\phi := true \mid false \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid O\,\phi \mid \phi_1\, U\, \phi_2, \text{ where } a \in AP.$$

The intuitive meaning of the formulae are as follows: $\phi_1 \wedge \phi_2$ indicates the conjunction of the formulae ϕ_1 and ϕ_2, $\neg\phi$ denotes the negation of ϕ, $O\,\phi$ denotes ϕ holds the next time, and $\phi_1\, U\, \phi_2$ indicates ϕ_2 eventually holds and ϕ_1 holds continuously until then.

We note that the until operator (U) allows defining modalities over infinite runs. For simulation techniques where the runs are bounded in length, we restrict the LTL syntax to only allow time bounded until (U^t). Here $\phi_1\, U^t\, \phi_2$ indicates that ϕ_2 holds within t unit of time and ϕ_1 holds continuously until then. The syntax of *bounded linear temporal logic (BLTL)* is hence as follows:

$$\phi := true \mid false \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid O\,\phi \mid \phi_1\, U^t\, \phi_2, \text{ where } a \in AP.$$

Since the system under consideration is probabilistic in nature, the formulae has to be quantified by probabilistic bounds.

Definition 5. *A* probabilistic bounded linear temporal logic property (PBLTL) *is a formula of the form* $P_{\geq\theta}(\phi)$*, where* ϕ *is a BLTL formula and* $\theta \in [0, 1]$*.*

We say that a system $\mathcal{M}$ satisfies PBLTL property $P_{\geq\theta}(\phi)$, denoted by $\mathcal{M} \models P_{\geq\theta}(\phi)$, if and only if the probability that $\mathcal{M}$ satisfies the BLTL property ϕ is greater than or equal to θ.

2.4 Statistical Model Checking

In this thesis, we analyze properties of Markov models specified in linear temporal logic. In this section, we briefly review the *statistical model checking (SMC)* procedure for verifying probabilistic bounded linear temporal (PBLTL) properties for Markov chains. This method will then be adopted for different Markov models in coming chapters.

The crux of the SMC procedure is as follows. We are interested in verifying a PBLTL property $P_{\geq\theta}(\phi)$ for a Markov chain $\mathcal{M}$. The model checking question is phrased as a hypothesis testing problem. Although a simulation based approach does not *guarantee* a correct result, the probability of error is bounded in advance. A number of simulations of the system is generated and used to provide statistical evidence of the hypothesis testing problem, and in turn, the model checking question with provable error bound.

2.4.1 Hypothesis Testing

We briefly overview the preliminaries of hypothesis testing before detailing the simulation procedure. For more details, see [LDB10, You04].

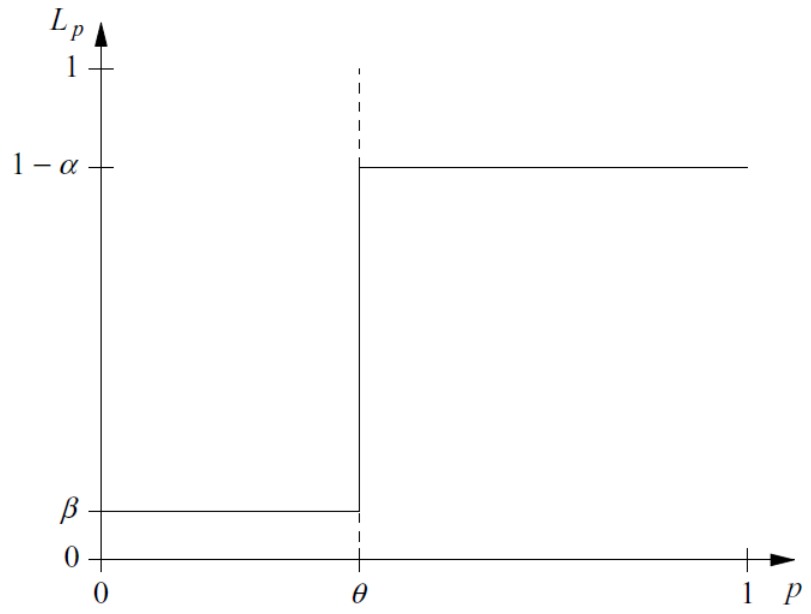


Figure 2.1: Probability (L_p) of accepting the hypothesis H as a function of p for a hypothetical statistical test.

Let p be the probability of satisfying ϕ. To verify whether $p \geq \theta$, we test the hypothesis $H : p \geq \theta$ against $K : p < \theta$. There are two types of errors we would like to minimize: (i) Type-I (false positive) error: accepting H when K holds, and (ii) Type-II (false negative) error: accepting K when H holds. We would like to ensure that the probabilities of Type-I and Type-II errors are bounded by pre-defined values (say) α and β, respectively. A hypothesis test will be ideal if the Type-I and Type-II errors are simultaneously exactly α and β respectively. Figure 2.1 demonstrates the probability of accepting H as a function of p, denoted H_p, in the X-axis for a hypothetical acceptance sampling test with ideal performance.

However, enforcing exact bounds on Type-I and Type-II errors simultaneously is flawed when $p = \theta$, in which case the two error probabilities can not be controlled independently [You04]. We relax the test by providing a range (p_1, p_0) such that $p_0 \geq p_1$. In this context, the hypothesis test

is rephrased as $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$ instead of H against K. The indifference region is generally defined in terms of a single threshold δ, hence as $(\theta - \delta, \theta + \delta)$. If the value of p is between $\gamma - \delta$ and $\gamma + \delta$, we say that the probability is sufficiently close to γ, so we are indifferent with respect to which of the hypothesis K or H are accepted.

2.4.2 Sequential Probability Ratio Test

Traditional sampling theory fixes the sample size in advance based on the Type-I and Type-II error thresholds α and β (e.g. single sampling plan [You04]). Computationally, it is often more efficient to estimate the sampling size adaptively, based on the observations made so far. Such an approach, proposed by Wald, is called sequential probability ratio test (SPRT) [Wal45].

In SPRT, two values A and B are chosen to ensure that the strength of the test is respected. We repeatedly simulate the system. Let B_i be a discrete random variable with a Bernoulli distribution of parameter p. The variable takes two values 0 and 1 with $Pr[B_i = 1] = p$ and $Pr[B_i = 0] = 1 - p$. In the context of SMC, each variable is associated with one simulation of the system. The outcome of B_i, denoted b_i, is 1 if the simulation satisfies ϕ or 0 otherwise.

After m simulation runs, let d_m be the number of runs with a positive outcome so far. We calculate a ratio

$$quo = \frac{p_{1m}}{p_{0m}} = \prod_{i=0}^{m} \frac{Pr(B_i = b_i \mid p = p_1)}{Pr(B_i = b_i \mid p = p_0)} = \frac{p_1^{d_m}(1 - p_1)^{m - d_m}}{p_0^{d_m}(1 - p_0)^{m - d_m}},$$

that takes into account the number of successes and failures seen so far. We accept H_0 if $quo \leq \frac{\beta}{1-\alpha}$ and H_1 if $quo \geq \frac{1-\beta}{\alpha}$. Otherwise, we continue the

simulation. The simulation is guaranteed to halt with probability 1 [You04] and will typically converge much before the number of samples required by a traditional static estimate.

Chapter 3

Distributed Stochastic Models

Throughout the thesis, we present a number of different distributed Markov models involving probability, time, and cost along with restrictions on expressiveness. In this chapter, we introduce Distributed Stochastic Models (DSM) – a general framework to unify the approach towards modeling concurrent systems with nondeterminism, probability and quantitative properties such as cost and time.

3.1 Models for Distributed Probabilistic Systems

Real-life systems are rarely sequential. From computational systems, biological phenomena to operational processes, the behavior of a system is often expressed across multiple agents that run in parallel and interact among each other. An extensive body of work has been done in last few decades to model distributed systems – see [WN95] for an overview.

Across various modeling formalisms, the key theme of modeling distributed systems is *atomic actions* – the smallest unit of uninterruptible system activity that may involve a number of components. Atomic actions

dictate the level of abstraction applied to the model. A system is executed by performing atomic actions that changes the dynamics of the system.

Distributed systems are composed of individual components, often interacting with and influencing each other. In some formalisms, the individuality of each component is abstracted away and the behavior of the system is purely expressed in terms of sequential outcomes of the system. Examples include transition systems [Kel76], which provides operational semantics to languages such as Calculus of Communicating Systems (CCS) [Mil82] and Communicating Sequential Processes (CSP) [BHR84]. On the other hand, it is often desirable to keep the "distributed-ness" of the model and regard components of the model as independent entities that communicate with each other. Prominent examples include Petri nets [RT86], event structures [Win89] and asynchronous transition systems [Shi85, Bed87].

Other than being distributed in nature, a key characteristic of real-life systems is that there is often no clearly defined termination. Hence, traditional input-output models are not a desirable formalism to capture the behavior of such systems. These are often termed as *reactive systems*.

Computational systems then also incorporate uncertainty – both measurable and non-measurable. One way to think of uncertainty is that within the system, alternatives are provided at decision points. The most popular formalism for modeling such systems is Markov models. For measurable uncertainty, the occurrence of the alternatives are *probabilistic* – often estimated from historical data and domain knowledge. An example of such model is Markov chains [Nor98]. When the information over the occurrence of the alternatives is not well known, it is abstracted as *nondeterminism*. Markov decision processes [Put94] is a widely used model that incorporates both

nondeterminism and probability.

Different traditional models for concurrency have been extended to accommodate uncertainty. Then, other quantitative features such as time and cost have also been modeled into concurrency formalisms in the last couple of decades. In this thesis, we aim to provide a model for concurrency that is well-equipped with uncertainty as well as quantitative properties such as time and cost. Before we discuss extensions of concurrency frameworks with uncertainty and quantitative aspects, we would like to reiterate that the goal of this thesis is to model real-life systems in such a way that quantitative analysis using simulation-based techniques can be adopted.

Labeled transition systems have been extended to incorporate randomization [Seg95], and languages such as CSP have been extended with real-time and probability [SSLD12]. The works of [Seg95, SSLD12] do not often have the ability to accommodate interleaved semantics with suitable restrictions. Also, the focus has been on numerical analysis, and not to connect it with simulation-based techniques (not to be confused with the very different simulation techniques in hierarchical verification [Seg95]). A rich body of work has been carried out on probabilistic timed systems [NPS13], with a variety of clock types – with application in verifying and optimizing rich time related properties. In general, it is not clear how simulation-based scalable model checking techniques can be readily adopted in such models.

Distributed models with probability and nondeterminism are also present in application areas such as planning and robotics. For example, in the work of Distributed MDPs [LCJ11], each MDP makes a local decision but can look at the states of other MDPs. A key application of such models is optimization using numerical techniques, and not quantitative analysis using

statistical methods.

Our framework—Distributed Stochastic Model (DSM)—is suitable for intensional description of systems that exhibit concurrency, probabilistic and nondeterministic behavior. It is closely related to Probabilistic Asynchronous Automata [JPS96] and Probabilistic Product Automata [BL03]. However, the intended utility of DSM is to model and verify quantitative properties of real-life systems in contrast to the language-theoretic approach in [JPS96] to generalize Zielonka's theorem [Zie87] to a probabilistic setting. The work in [BL03] defines a product automata with nondeterminism and probability to verify properties specified in product linear temporal logic. We agree with the approach in [BL03] to start with a distributed model, which provides a direct way to model a distributed system and implement on-the-fly verification techniques. Our approaches differ in the sense that in this thesis, we do not provide a general model checking algorithm for our framework. Instead, we focus on simulation-based techniques and develop variations of our framework which is then applied to different application domains.

3.2 Formal Definition

We start with defining distributed state spaces over a set of finite agents. Each agent has a finite set of local states and a unique initial state. The agents each independently start at their initial state. The global configuration of the system is captured by global states, represented by the tuple of local states of all the agents. The configuration of the model can also be viewed partially, due to the distributed nature of the model. For instance, for a given set of agents, the tuple of its local states represent their collective configuration.

Definition 6 (Distributed State Space)**.** *Let us assume that we have a system of n agents* $[n] = \{1, 2, \ldots, n\}$*. A* distributed state space *over n agents is a tuple* $\mathcal{S} = (n, \{S_i\}, \{s_i^{in}\})$*, such that for each agent* $i \in [n]$*,* S_i *denotes its finite set of local states and* s_i^{in} *denotes its local initial state. We may abbreviate* $[n]$*-indexed sets* $\{X_i\}_{i \in [n]}$ *as* $\{X_i\}$ *when the context is clear.*

- *For non-empty* $u \subseteq [n]$*,* $\mathbf{S}_u = \prod_{i \in u} S_i$ *denotes the set of joint u-states of agents in u. We denote* $\mathbf{S} = \mathbf{S}_{[n]}$ *to be the set of* global states*.*
- *For an u-state* $\mathbf{s} \in \mathbf{S}_u$ *and* $v \subseteq u$*,* $\mathbf{s}_v$ *denotes the projection of* $\mathbf{s}$ *onto v. We do not distinguish between* $\mathbf{S}_{\{i\}}$ *and* $\mathbf{S}_i$*, nor between* $\mathbf{s}_{\{i\}}$ *and* $\mathbf{s}_i$*.*

The dynamics of the distributed system is captured by *events*. An event represents a set of agents (not necessarily all of them) synchronizing together and changing their corresponding local states. An event only affects the agents involved and not the rest of the agents, hence multiple independent events can be triggered concurrently. (We will formalize the notion of independence later on). A set of agents may synchronize and then take part in multiple events. A set of events participated by the same agents on the same set of their corresponding local states are grouped together to form an *action*. Thus, an action represents a set of agents in their corresponding local states synchronizing and then choosing an event to trigger. We make this choice probabilistic, i.e. there is a probability distribution over the set of events attached to the action. An action is enabled when the participating agents are in the required local states.

Definition 7 (Events and Actions)**.**

- *An* event *over a distributed state space* $\mathcal{S}$ *is a tuple* $e = (src_e, tgt_e)$*, such that* $\varnothing \neq loc(e) \subseteq [n]$ *specifies the agents that participate in e and* $src_e, tgt_e \in$

$\mathbf{S}_{loc(e)}$ *denote the source and target* $loc(e)$*-states of* e*.*

- *An* action *over* $(\mathcal{S}, \Sigma)$ *is a collection of co-located events with the same source state, equipped with a probability distribution. Formally, an action is a pair* $a = (E_a, \pi_a)$*, where* $E_a \subseteq \Sigma$ *such that for each* $e, e' \in E_a$*,* $src_e = src_{e'}$*, and* $\pi_a : E_a \to [0, 1]$ *is a probability distribution.*

- *We assume that each event belongs to exactly one action. In other words* $\Sigma = \bigcup_{a \in A} E_a$ *and for each* $a, b \in A$ *such that* $a \neq b$*,* $E_a \cap E_b = \varnothing$*.*

- *We write* $loc(a)$ *for the set of agents participating in action* a *and* $src(a)$ *for the common start state of the events in* E_a*. Thus,* $loc(a) = loc(e)$ *and* $src(a) = src(e)$ *for all* $e \in E_a$*. An action* a *is* internal *of some agent* $i \in [n]$ *iff* $loc(a) = \{i\}$*. Let* A *denote the set of all actions over* $(\mathcal{S}, \Sigma)$*.*

- *For a global state* $\mathbf{s} \in \mathbf{S}$*,* $en(\mathbf{s})$ *is the set of actions* enabled *at* $\mathbf{s}$*. Formally,* $en(\mathbf{s}) = \{a \mid src(a) = \mathbf{s}_{loc(a)}\}$*.*

We now define Distributed Stochastic Models (DSM), which will then be restricted and extended for different forms of results in coming chapters of the thesis.

Definition 8 (Distributed Stochastic Models)**.** *A Distributed Stochastic Model (DSM) is a tuple* $\mathcal{D} = (\mathcal{S}, \Sigma, \chi, A)$ *such that*

- $\mathcal{S} = (n, \{S_i\}, \{s_i^{in}\})$ *is a distributed state space,*

- Σ *is the set of events over* $\mathcal{S}$*,*

- A *is the set of actions over* $(\mathcal{S}, \Sigma)$*,*

- *Each event* $e \in \Sigma$ *has a non-negative real-valued* cost $\chi(e)$*, given by a cost function* $\chi : \Sigma \to \mathbb{R}_{\geq 0}$ *over* Σ*.*

Example 1. *In Figure 3.1, we illustrate a simple coin toss game with two players and one shared coin. The game is modeled as a DSM where the players as well as the coin are modeled as agents.*

Two players each toss the unbiased coin. If both toss tails, the players toss again. If both tosses heads, with a small probability the game is undecided and otherwise both toss again. If the outcomes are different, then the player who tossed heads wins and the other player loses. Once one of them wins, the game is finished.

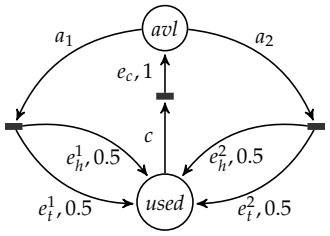
The distributed state space is defined as follows. This is a 3*-agent system, i.e.* $n = 3$*. The coin—agent* 0*—has two states: avl, where the coin is available for the players to pick up, and used, when the coin has been used by a player and is not available yet.*

For each agent $i \in \{1, 2\}$*, the set of local states is* $S_i = \{IN_i, T_i, H_i, L_i, W_i, U_i, F_i\}$ *where (i)* IN_i *denotes the agent* i *ready to toss, (ii)* T_i / H_i *denote that a tail/head was tossed by agent* i*, (iii)* L_i / W_i *denote agent* i *losing/winning, (iv)* U_i *denotes the local state indicating the game being undecided, (v)* F_i *denotes the local state indicating the end of the game.*

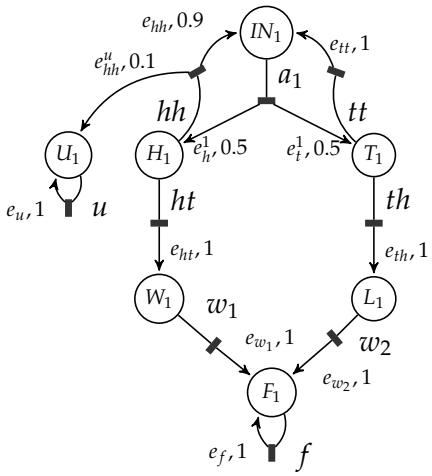
The states in the diagram are represented by circles whereas the actions are represented by black rectangular bars. The lines without arrow represent the occurrence of actions while the lines with arrows represent the events. Note that for simplicity, we numbered the agents as $0, 1, 2$*, instead of* $1, 2, 3$*.*

In the example, for $i \in \{1, 2\}$, agent i participates in action $a_i = (\{e^i_h, e^i_t\}, \pi_{a_i})$ with $loc(a_i) = \{0, i\}$ such that $e^i_h = ((avl, IN_i), (used, H_i))$, $e^i_t = ((avl, IN_i), (used, T_i))$, and $\pi_{a_i}(e^i_t) = 0.5 = \pi_{a_i}(e^i_h)$. The action a_i denotes the player i tossing the coin while events e^i_h and e^i_t denote the possible results of the toss – heads and tails respectively.

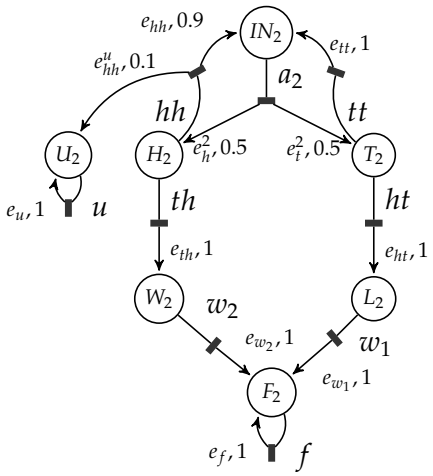
After both players toss, say a tail, the action tt is performed. Formally,



Agent 0: the coin.



Agent 1: player 1.



Agent 2: player 2.

Figure 3.1: The coin toss game with one coin – an example of a DSM.

$tt = (\{e_{tt}\}, \pi_{tt})$ is an action with $loc(tt) = \{1, 2\}$ and $\pi_{tt}(e_{tt}) = 1$. When the first player tosses heads and the second player tosses tails, agent 1 and 2 are in their local states H_1 and T_2 respectively. Then, they synchronize on the common action $ht = (\{e_{ht}\}, \pi_{ht})$ with $e_{ht} = ((H_1, T_2), (W_1, L_2))$ and $\pi_{ht}(e_{ht}) = 1$. Then, the first player wins (agent 1 in W_1) and the second player loses (agent 2 in L_2). The end of the game is depicted by the global state (F_1, F_2), where the agents stay indefinitely following the action f.

When the coin is available, which player gets the coin first is a nondeterministic choice. Thus, a_1 and a_2—the two actions representing the two players tossing the coin—are in conflict. On the other hand, when one player tosses the coin, the action c (representing the coin being available again) is independent of any other action.

Definition 9 (Conflicts and independence)**.** *Let $en(\mathbf{s})$ be the set of enabled actions at a global state $\mathbf{s}$. Two enabled actions $a, b \in en(\mathbf{s})$ are*

- in conflict *iff there exists an agent $i \in loc(a) \cap loc(b)$.*
- independent *iff $loc(a) \cap loc(b) = \varnothing$.*

The notion of conflict and independence can also be lifted to events in a natural way.

The conflict among actions introduces nondeterminism in the system, while the independence presents an opportunity for concurrency. In a global state, among the actions that are enabled, a schedulable set of actions needs to be chosen by an external scheduler. This set of actions must be mutually independent, thus allowing them to run concurrently. Since we are interested in the dynamics of the system, we do not allow the schedulable set of actions to be empty unless the set of enabled actions itself is empty.

Definition 10 (schedulable set of actions)**.** *At a global state* $\mathbf{s} \in \mathbf{S}$*, a set of enabled actions* U *is* schedulable *if each agent participates in at most one action in* U*. Formally,* $\varnothing \neq U \subseteq en(\mathbf{s})$ *is schedulable if for all* $a, b \in U$ *such that* $a \neq b$*,* $loc(a) \cap loc(b) = \varnothing$*. Let* $sch(\mathbf{s}) \subseteq 2^{en(\mathbf{s})} \setminus \varnothing$ *denote the collection of schedulable sets of actions at* $\mathbf{s}$*.*

3.3 Semantics for Distributed Stochastic Models

There are two fundamental ways to express the behavior of concurrent systems – non-interleaved and interleaved semantics. Non-interleaved semantics involves executing all independent atomic events together at one step. On the other hand, interleaved semantics reduces concurrency to a sequential computation where independent events are performed one at a time. Non-interleaved semantics is described as a modeling formalism that is closer to physical reality of concurrent systems, whereas interleaved semantics is often helpful for better understanding and simpler simulation of the system. In some concurrent models these two semantics are contextually indistinguishable, and then interleaved semantics acts as an elegant decomposition of non-interleaved.

For distributed models with nondeterminism, the semantics needs to incorporate an external scheduler that resolves nondeterminism. At any global state, the scheduler, based on the system's run so far and the current state, chooses a schedulable set of actions that the system will perform at the current state. When we restrict the scheduler to be finite memory, it may remember only a finite part of the run, and for memoryless schedulers, it may decide the next schedulable set of actions based on only the current state.

For distributed stochastic models, defining the semantics has another layer of complexity. Since some of the decisions in the dynamics of the system are also probabilistic, the set of runs in the semantics call for a well-defined probability measure. Also, the semantics should ideally be powerful enough to seamlessly accommodate both numerical and simulation based algorithms.

We start with a non-interleaved semantics for distributed stochastic models. We associate a Markov decision process along with the distributed stochastic model and a pre-defined scheduler. At any global state, if no actions are enabled, no run is possible from that global state. Otherwise, the scheduler chooses a schedulable set of actions. Since the actions in the chosen set can be executed concurrently, the next state of states is then governed by the joint probability distribution of the chosen actions.

Definition 11 (Runs of a DSM)**.** *A finite run of a DSM from a global state* $\mathbf{s} \in \mathbf{S}$ *is a sequence of the form* $\rho = \mathbf{s}_0 E_0 \mathbf{s}_1 E_1 \dots E_{k-1} \mathbf{s}_k$ *such that* $\mathbf{s}_0 = \mathbf{s}$ *and, for* $0 \leq l < k$ *the following is true:*

- *there exists a schedulable set of actions* $A_l \subseteq sch(s_l)$ *and* $e \in E_l$ *iff* $e \in E_a$ *for some* $a \in A_l$*.*
- *for each* $e = (src_e, tgt_e) \in E_l$, $(s_{l+1})_{loc(e)} = tgt_e$*.*

The last state of a finite run ρ *is defined as* $last(\rho)$*. The set of all finite runs from a state* $\mathbf{s}$ *is defined as* $runs(\mathbf{s})$*. Infinite runs are defined as usual.*

Note that the runs depend on an external scheduler's choice of schedulable set of actions at each state. We now define a scheduler that takes the current run as input and decides a schedulable set of actions. In general,

since the length of the run can be arbitrarily long, the scheduler is equipped with infinite memory.

Definition 12 (Schedulers for DSM)**.** *A scheduler for a DSM $\mathcal{D}$ is a function $\mathbb{S} : \cup_{\mathbf{s}\in\mathbf{S}} runs(\mathbf{s}) \to 2^A$ such that $\mathbb{S}(\rho) \in sch(last(\rho))$.*

We did not explicitly include the memory of the scheduler in its definition, but the scheduler may be infinite-memory, finite-memory or memoryless depending on the context. Whenever mentioned, we will detail the restriction on the memory of the scheduler. In general, the scheduler takes the current run so far and selects a schedulable set of actions enabled at the last state of the run.

Definition 13 (MDP associated with a DSM)**.** *Let $\mathcal{D} = (\mathcal{S}, \Sigma, \chi, A)$ be a DSM with $\mathcal{S} = (n, \{S_i\}, \{s_i^{in}\})$. We define a Markov Decision Process $M_{\mathcal{D}} = (\mathbf{S}, \mathbf{s}^{in}, Step)$ associated with $\mathcal{D}$ as follows:*

- *$\mathbf{S}$ is the set of global states,*
- *$\mathbf{s}^{in} = (\mathbf{s}_1^{in}, \ldots, \mathbf{s}_n^{in})$ is the initial state,*
- *$Step : \mathbf{S} \to 2^A$ defines a transition function such that, for each $\mathbf{s} \in \mathbf{S}$, the collection of schedulable set of actions that can be performed after $\mathbf{S}$ from the collection $sch(\mathbf{s})$, i.e. $Step(\mathbf{s}) = sch(\mathbf{s})$,*
- *$P : \mathbf{S} \times 2^A \to dist(\mathbf{S})$ be the probability distribution function associated with a global state $\mathbf{s}$ and a transition $U \in step(\mathbf{s})$ such that $P(\mathbf{s}, U) = \prod_{a\in U} \pi_a$, the joint probability distribution over all actions in the schedulable set of actions.*

The Markov decision process associated with the DSM depicting the coin toss game is presented in Fig 3.2. Only a part of the MDP is shown

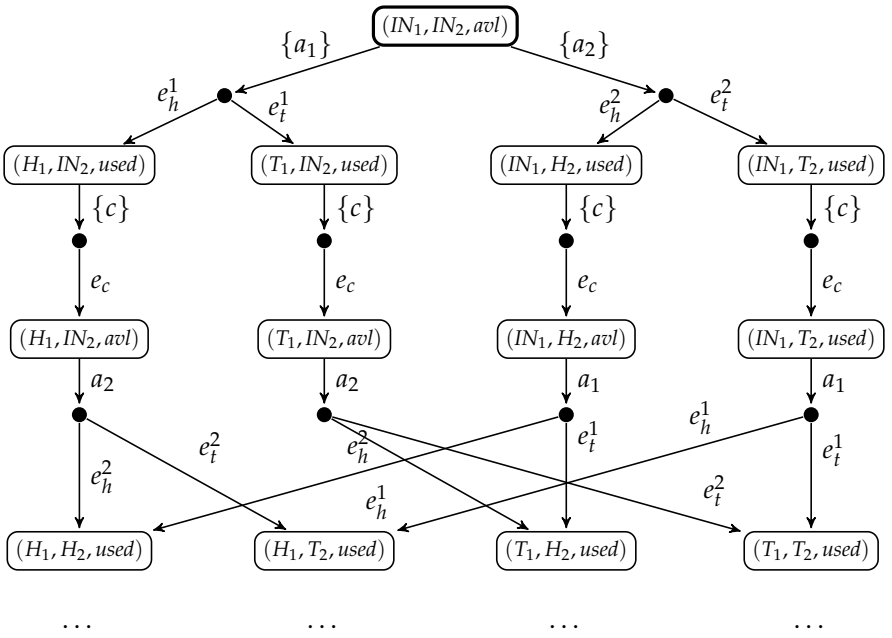


Figure 3.2: (Part of) Markov decision process associated with the coin toss game.

to facilitate ease of understanding. We can see that from the initial state (IN_1, IN_2, avl), two schedulable set of actions are available—$\{a_1\}$ and $\{a_2\}$—representing the two players tossing the coin. When a player uses the coin, the coin needs to be available again (thus the action c) before the other player can use it.

The scheduler chooses among the collection of schedulable set of actions at a given state. This choice governs the dynamics of the MDP. We show an example in Figure 3.3, where the scheduler has to choose among three transitions—$\{hh\}, \{c\}, \{hh, c\}$—at state $(H_1, H_2, used)$. In this particular example, the ordering of the steps $\{hh\}$ and $\{c\}$ does not matter since they coincide with effect of the step $\{hh, c\}$. This observation triggers two conjectures:

- (C1) the ordering of independent steps do not affect the overall dy-

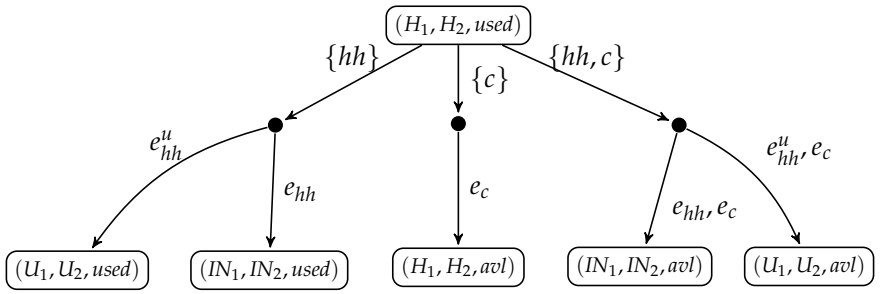


Figure 3.3: Part of Markov decision process associated with the coin toss game starting from the state $(H_1, H_2, used)$.

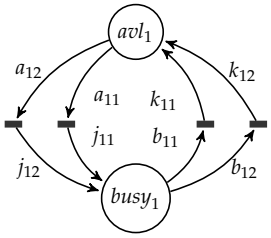
namics of the system,

- (C2) it is contextually equivalent to choose the step with maximal parallel set of actions.

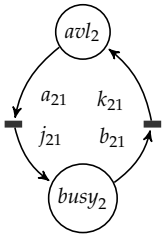
As it turns out, both the conjectures are false for a general DSM.

Example 2. *Let us assume there are two tasks T_1 and T_2, and two resources R_1 and R_2. The resource R_1 can perform both T_1 and T_2, whereas R_2 can only perform task T_1. The task T_i for $i = 1, 2$ has three states—$start_i$, $ready_i$, and $done_i$—representing the start of a task, when a task is ready with an assigned resource, and when the task is finished by the resource respectively. The resource R_i for $i = 1, 2$ has two states—avl_i and $busy_i$—representing the resource being available to perform a task and busy being assigned to a task. The corresponding DSM is depicted in Figure 3.4.*

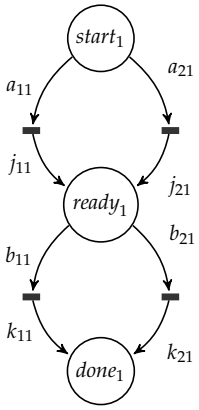
When both the resources are available and the tasks are at their corresponding start states, the resource R_1 can be assigned to both tasks T_1 and T_2, but R_2 can only be assigned to T_1. An external scheduler takes the decision of scheduling the tasks to their resources. The scheduler can schedule R_1 to perform T_2 (the action a_{12}) and R_2 to T_1 (the action a_{21}) at the state $(avl_1, avl_2, start_1, start_2)$, since a_{12} and a_{21} are independent. However, that



(i) Resource 1 (R_1).



(ii) Resource 2 (R_2).



(iii) Task 1 (T_1).

(iv) Task 2 (T_2).

Figure 3.4: A DSM depicting two tasks and two resources from Example 2.

may not be in the best interest of the scheduler, which for example, may try to optimize the cost of operations. We can imagine a situation where resource R_1 is much more cost-efficient than R_2 so that, it is better to not assign R_2 at all, and make R_1 do both the tasks sequentially. Thus, it is evident that depending on the context, choosing the step with maximal parallel actions may not be contextually equivalent or even desirable - invalidating conjecture C2.

The actions a_{12} and a_{11} are in conflict, whereas a_{11} and a_{21} are in conflict at the state $(avl_1, avl_2, start_1, start_2)$. Since the actions a_{12} and a_{21} are independent, the conjecture C1 infers that executing one does not affect the scheduler's choice over the other. However, if the scheduler chooses to execute a_{21}, the set of actions conflicted with a_{12}—an action in principle unrelated to the decision—does not include a_{11} anymore. In other words, the independent actions and the actions in conflict "interfere". Thus, the scheduler resolving concurrency not only determines the order in which independent actions occur, but can also influence whether an action can appear at all in a run. Concurrency theory has coined a term for this: *confusion* [RT86, DE95, VVW04]. As evident in this example and further detailed in Chapter 5 where we model operational processes in terms of DSM, confusion can play a crucial role in certain applications.

The non-interleaved semantics we defined for DSM gives the scheduler ample power. It is not restricted to triggering maximal parallelism of the independent actions, and the scheduler is allowed to choose any schedulable set of actions. We note that however, in general, it is hard to define independence-respecting schedulers for distributed systems. This is, for example, related to defining local winning strategies in distributed

games [GLZ04]. The main complication is that a sequentially defined scheduler must behave consistently across different linearizations that correspond to the same concurrent execution. Also, due to both concurrency and probability present in DSM, it is hard to define an interleaved semantics in which we could define a probability measure that takes into account the equivalence of runs due to concurrency. However, in a variation of DSM with fixed-duration events, presented in Chapter 5, the durations associated with the events fix a canonical linearization, so there is no need to reconcile decisions of the scheduler across different interleavings. Also, in Chapter 4, we present another variation where we restrict the nondeterminism and then the need of a scheduler is non-existent.

Chapter 4

Distributed Markov Chains

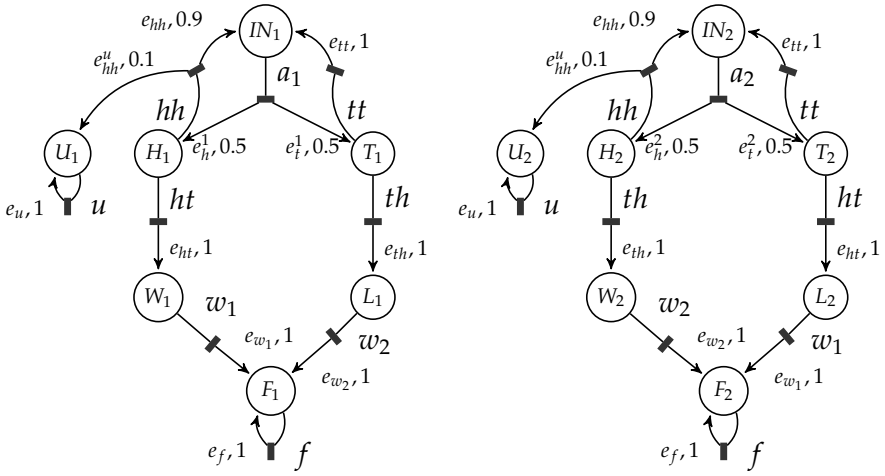
In the previous chapter, we have discussed a framework that involve nondeterminism in its dynamics. In distributed algorithms and concurrent protocols, we also often see systems that include stochastic behavior and concurrency, but not nondeterminism. Later in this chapter, we show a couple of case studies from the PRISM model checker benchmark [HKNP06] as examples. Formal verification of such systems is often performed by assuming the system as a large probabilistic model such as a Markov chain. However, exploiting the concurrency present in such a model is one way to simplify the analysis.

In this chapter, we restrict the Distributed Stochastic Model (DSM) from Chapter 3 to entail only *deterministic actions* such that no two enabled actions involve a common agent. In other words, for all global states, there exists no agent with a nondeterministic choice between two enabled actions at that global state. This condition is enforced syntactically as follows: two actions involve either (i) two disjoint set of agents, in which case no nondeterminism arises when both the actions are enabled, or (ii) some common agent, and there exists a (possibly same) common agent which is not in the same

local state when participating in the two actions. This subclass of DSM is called Distributed Markov Chains (DMC). In many distributed probabilistic systems, the communication protocols are naturally deterministic in this sense, or can be designed to be so.

The approach towards building the DMC model is in line with partial order based methods for Markov Decision Processes (MDPs) [GB06] where, typically, a partial commutation structure is imposed on the actions of a *global* MDP. For instance, in [BFHH11], partial order reduction is used to identify "spurious" nondeterminism arising out of the interleaving of concurrent actions, in order to determine when the underlying behavior corresponds to a Markov chain. In contrast, in a DMC, deterministic communication ensures that local behaviors always generate a global Markov chain. The independence of actions is directly given by the local state spaces of the components. This also makes it easier to model how components influence each other through communications.

The interplay between concurrency and stochasticity has also been explored in the setting of event structures [AB06, VVW04]. In these approaches, the global behavior — which is not a Markov chain — is endowed with a probability measure. Further, probabilistic verification problems are not formulated and studied. Markov nets, studied in [AB08], can be easily modeled as DMCs. However, in [AB08], the focus is on working out a probabilistic event structure semantics rather than on developing a model checking procedure based on the interleaved semantics, as we do here.



Agent 1: player 1. Agent 2: player 2.

Figure 4.1: The coin toss game - An example of a DMC.

4.1 Formal Definition

Definition 14 (Distributed Markov Chains). *Let $\mathcal{D} = (\mathcal{S}, \Sigma, \chi, A)$ be a Distributed Stochastic Model. We define $\mathcal{D}$ to be a Distributed Markov Chain (DMC) if the following condition is satisfied: for any two actions $a, b \in A$,*

- *either $loc(a) \cap loc(b) = \varnothing$,*
- *or $loc(a) \cap loc(b) \neq \varnothing$ implies that there exists $i \in loc(a) \cap loc(b)$ such that $(src(a))_i \neq (src(b))_i$.*

Example 3. *In the example in the coin toss game from Figure 3.1 of Chapter 3 (page 26), two players were sharing a single coin. The source of nondeterminism was the fact that the availability of the coin to the players was arbitrary. We now modify the game to include two coins, one for each player. Then, the coins need not be represented as agents anymore, since each coin toss can be represented as an internal action of an agent. We model this modified coin toss game as a DMC, illustrated in Figure 4.1.*

The coin toss game with two coins now adheres to the DMC condition. For example, for actions a_1 and a_2, $loc(a_1) \cap loc(a_2) = \varnothing$. Also, for instance, for actions hh and ht, $\{1, 2\} \in loc(hh) \cap loc(ht)$, but $(src(hh))_2 \neq (src(ht))_2$.

We claim that in a DMC, no two actions can be in conflict, i.e. at any global state no two enabled actions can have a common agent participating (formal definition at Definition 9, page 27). In other words, the local state of an agent uniquely determines the action it can participate in.

Claim 1. *For each global state of a DMC $\mathcal{D} = (\mathcal{S}, \Sigma, \chi, A)$, if a set of enabled actions is non-empty, it is schedulable. Formally, for any global state $\mathbf{s} \in \mathbf{S}$, $en(\mathbf{s}) \neq \varnothing$ implies $en(\mathbf{s}) \in sch(\mathbf{s})$.*

Proof. For a given global state $\mathbf{s} \in \mathbf{S}$, since $en(\mathbf{s}) \neq \varnothing$, there exists some actions that are enabled. Hence a subset of enabled actions is at least schedulable, i.e. there exists some $U \subseteq en(\mathbf{s})$ such that $U \neq \varnothing$ and $U \in sch(\mathbf{s})$. Let $a \in en(\mathbf{s}) \setminus U$ be an action such that $U \cup \{a\} \notin sch(\mathbf{s})$. This implies that there exists some $a' \in U$ such that $loc(a) \cap loc(a') \neq \varnothing$. Then according to the necessary condition of DMC, there exists an agent $i \in loc(a) \cap loc(a')$ such that $(src(a))_i \neq (src(a'))_i$. However, since $\{a, a'\} \in en(\mathbf{s})$, $(src(a))_i = \mathbf{s}_i = (src(a'))_i$, a contradiction. Hence $U \cup \{a\} \in sch(\mathbf{s})$. Proceeding with the same argument now with $U \cup \{a\}$ in place of U, we show that $en(\mathbf{s}) \in sch(\mathbf{s})$. □

The trace alphabet Using the necessary condition for a DMC $\mathcal{D} = (\mathcal{S}, \Sigma, \chi, A)$, we define an independence relation over the set of events Σ as follows. The independence relation $I \subseteq \Sigma \times \Sigma$ is given by $e \, I \, e'$ iff the following is true: (i) either $loc(e) \cap loc(e') = \varnothing$ or (ii) $loc(e) \cap loc(e') \neq \varnothing$ implies that there exists $i \in loc(e) \cap loc(e')$ such that $(src(e))_i \neq (src(e'))_i$. Since I is irreflexive

and symmetric, (Σ, I) is a Mazurkiewicz trace alphabet [DR95]. We note that this independence is different than that presented in Chapter 3. The independence relation present in the context of DMC is more universal in the sense that the it is defined over the set of all events.

4.2 The Non-interleaved Semantics

Due to determinacy of synchronizations, all enabled actions can be performed at any global state. Hence, there is no need for a scheduler in a DMC. Consequently, we can define a non-interleaved semantics for a DMC by executing all the enabled actions simultaneously, followed by probabilistic moves (events) by all the agents involved. We show that this turns out to be a finite state Markov chain that captures the global behavior of the DMC under this "maximally parallel" execution semantics. From a different perspective, Markov chains with a considerable degree of concurrency can be seen and dealt with as a DMC.

A nonempty set of events $u \subseteq en(\mathbf{s})$ is a *step* at $\mathbf{s}$ if for every pair of distinct events $e, e' \in u$, $e \; I \; e'$. We say u is a *maximal* step at $\mathbf{s}$ if u is a step at $\mathbf{s}$ and $u \cup \{e\}$ is not a step at $\mathbf{s}$ for any $e \notin u$. In the running example, two actions a_1 and a_2 are enabled at the initial state (IN_1, IN_2). Hence, for instance in the running example (see Fig. 4.2), there are 4 maximal steps at (IN_1, IN_2) — $\{e_h^1, e_h^2\}$, $\{e_h^1, e_t^2\}$, $\{e_t^1, e_h^2\}$ and $\{e_t^1, e_t^2\}$.

Let u be a maximal step at $\mathbf{s}$. Then $\mathbf{s}'$ is the u-successor of $\mathbf{s}$ if the following conditions are satisfied: (i) For each $e = (src_e, tgt_e) \in u$, if $i \in loc(e)$ then $\mathbf{s}'_i = (tgt_e)_i$, and (ii) $\mathbf{s}_j = \mathbf{s}'_j$ if $j \notin loc(u)$, where $loc(u) = \bigcup_{e \in u} loc(e)$.

Suppose u is a maximal step at $\mathbf{s}$ and $i \in loc(u)$. Then, because events in a step are independent, it follows that there exists a unique $e \in u$ such that

$i \in loc(e)$, so the u-successor of $\mathbf{s}$ is unique. We say $\mathbf{s}'$ is a *successor* of $\mathbf{s}$ if there exists a maximal step u at $\mathbf{s}$ such that $\mathbf{s}'$ is the u-successor of $\mathbf{s}$. From the definition of a DMC, it is easy to see that if $\mathbf{s}'$ is a successor of $\mathbf{s}$ then there exists a unique maximal step u at $\mathbf{s}$ such that $\mathbf{s}'$ is the u-successor of $\mathbf{s}$. Finally, we say that $\mathbf{s}$ is a *deadlock* if no action is enabled at $\mathbf{s}$.

Definition 15. *The Markov chain $\mathcal{M} : \mathbf{S} \times \mathbf{S} \to [0,1]$ generated by DMC $\mathcal{D}$ is given as follows:*

- *The initial state of $\mathcal{M}$ is $\mathbf{s}^{in} = (s_1^{in}, s_2^{in}, \ldots, s_n^{in})$.*
- *If $\mathbf{s} \in \mathbf{S}$ is a deadlock then $\mathcal{M}(\mathbf{s}, \mathbf{s}) = 1$ and $\mathcal{M}(\mathbf{s}, \mathbf{s}') = 0$ for $\mathbf{s}' \neq \mathbf{s}$.*
- *Suppose $\mathbf{s} \in \mathbf{S}$ is not a deadlock. Then $\mathcal{M}(\mathbf{s}, \mathbf{s}') = p$ if there exists a maximal step u at $\mathbf{s}$ such that $\mathbf{s}'$ is the u-successor of $\mathbf{s}$ and $p = \prod_{e \in u \cap E_a} \pi_a(e)$.*
- *If $\mathbf{s}$ is not a deadlock and $\mathbf{s}'$ is not a successor of $\mathbf{s}$ then $\mathcal{M}(\mathbf{s}, \mathbf{s}') = 0$.*

Claim 2. *$\mathcal{M}$ is a finite state Markov chain.*

Proof. Since there are finite number of agents each with finite set of local states, the set of global states $\mathbf{S}$ is finite. Also, since product of probability values is always between 0 and 1, $\mathcal{M}(\mathbf{s}, \mathbf{s}') \in [0,1]$ for every $\mathbf{s}, \mathbf{s}' \in \mathbf{S}$. We now need to show that

$$\sum_{\mathbf{s}' \in \mathbf{S}} \mathcal{M}(\mathbf{s}, \mathbf{s}') = 1 \; \forall \; \mathbf{s} \in \mathbf{S}.$$

If $\mathbf{s}$ is a deadlock the result follows at once. So we assume that $\mathbf{s}$ is not a deadlock. Since $\mathbf{s}$ is not a deadlock, the set of enabled actions at $\mathbf{s}$ is non-empty, i.e. $en(\mathbf{s}) \neq \varnothing$.

Let u be a maximal step at $\mathbf{s}$. If $e = (src_e, tgt_e) \in u$ and $e \in E_a$ for some action $a \in A$, then $a \in en(\mathbf{s})$ by definition of en. On the other hand, if

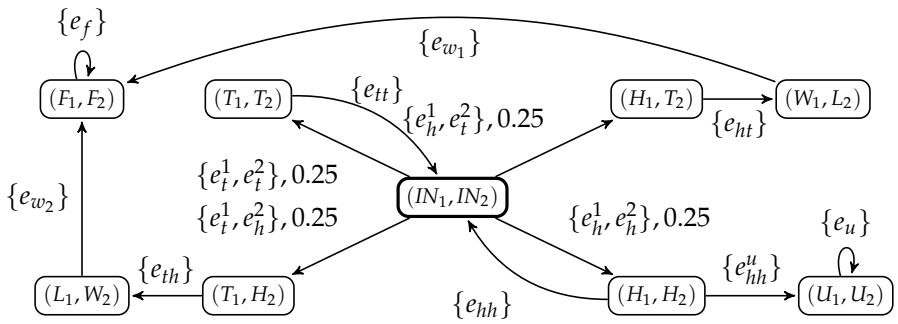


Figure 4.2: Markov chain for the DMC in Example 3.

$a \in en(\mathbf{s})$ then there must exist an event of the form $e \in u$ and $e \in E_a$ since u is a maximal step and according to Claim 1, events in a can not be in conflict with any events in u. For the same reason if $e, e' \in u$ then $loc(e) \cap loc(e') \neq \varnothing$ iff $e = e'$. Thus $|u| = |en(\mathbf{s})|$ and for each $a \in en(\mathbf{s})$ there exists a unique event e in u such that $e \in E_a$.

Clearly $\mathbf{s}_a = src_e$ for each $e \in E_a$ and $\sum_{e \in E_a} \pi^a(\mathbf{s}_a)(tgt_e) = 1$ by definition. The required result now follows from the fact that u is a maximal step at $\mathbf{s}$ iff $|u \cap E_a| = 1$ for each $a \in en(\mathbf{s})$. □

In Fig. 4.2 we show the Markov chain associated with the DMC shown in Fig. 4.1. The initial state (IN_1, IN_2) has a thicker outline. Each edge is labeled with the maximal step and the corresponding probability. Unlabeled transitions have probability 1.

4.2.1 The Path Space of Non-interleaved Semantics

Let $\mathcal{M}$ be the Markov chain associated with a DMC $\mathcal{D}$. The path space and a probability measure over this space is obtained in the usual way. A finite path in $\mathcal{M}$ from $\mathbf{s}$ is a sequence $\tau = \mathbf{s}_0\mathbf{s}_1 \ldots \mathbf{s}_m$ such that $\mathbf{s}_0 = \mathbf{s}$ and $\mathcal{M}(\mathbf{s}_\ell, \mathbf{s}_{\ell+1}) > 0$, for $0 \leq \ell < m$. The notion of an infinite path starting from

$\mathbf{s}$ is defined as usual. $Path_{\mathbf{s}}$ and $Path_{\mathbf{s}}^{fin}$ denote the set of infinite and finite paths starting from $\mathbf{s}$, respectively.

For $\tau \in Paths_{\mathbf{s}}^{fin}$, $\uparrow\tau \subseteq Paths_{\mathbf{s}}$ is the set of infinite paths that have τ as a prefix. $Y \subseteq Path_{\mathbf{s}}$ is a basic cylinder at $\mathbf{s}$ if $Y = \uparrow\tau$ for some $\tau \in Paths_{\mathbf{s}}^{fin}$. The σ-algebra over $Path_{\mathbf{s}}$, denoted $SA(\mathbf{s})$, is the least family that contains the basic cylinders at $\mathbf{s}$ and is closed under countable unions and complementation (relative to $Path_{\mathbf{s}}$). $P_{\mathbf{s}} : SA(\mathbf{s}) \to [0,1]$ is the usual probability measure that assigns to each basic cylinder $\uparrow\tau$, with $\tau = \mathbf{s}_0\mathbf{s}_1 \ldots \mathbf{s}_m$, the probability $p = p_0 \cdot p_1 \cdots p_{m-1}$, where $\mathcal{M}(\mathbf{s}_\ell, \mathbf{s}_{\ell+1}) = p_\ell$, for $0 \leq \ell < m$.

4.3 The Interleaved Semantics

As we have discussed in Chapter 3, it is not trivial to define an interleaved semantics for Distributed Stochastic Models (DSM), let alone construct it as contextually equivalent to the non-interleaved semantics. However, we show that one can define an interleaved semantics for a DMC (which is a restricted version of DSM) where one synchronization action is executed at a time, followed by a probabilistic move by the participating agents. Except in the trivial case where there is no concurrency, the resulting transition system will *not* be a Markov chain. Hence, defining a probability measure over interleaved runs, called *trajectories*, is a technical challenge. We now associate a global transition system with the DMC $\mathcal{D}$ based on event occurrences.

Definition 16 (The transition system)**.** *Let $\mathcal{D} = (\mathcal{S}, \Sigma, \chi, A)$ be a DMC such that $\mathcal{S} = (n, \{S_i\}, \{s_i^{in}\})$. We define the transition system associated with $\mathcal{D}$ to be $TS = (\mathbf{S}, \mathbf{s}^{in}, \to)$ such that*

- *$\mathbf{S}$ is the set of global states and $\mathbf{s}^{in} = (s_1^{in}, s_2^{in}, \ldots, s_n^{in})$ is the initial global*

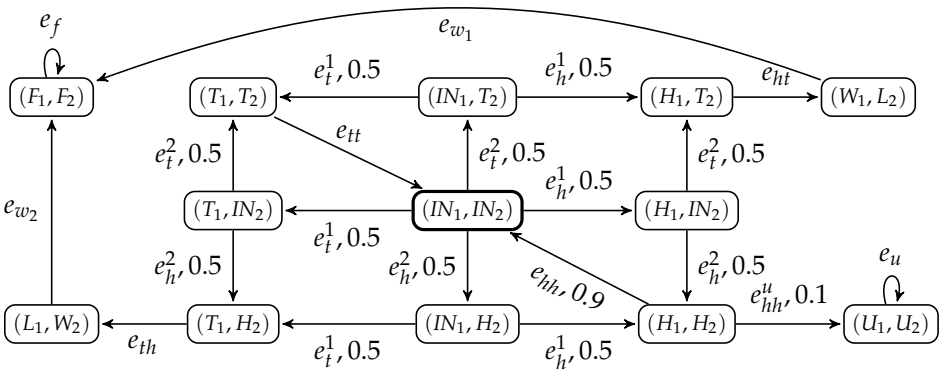


Figure 4.3: Transition system for the DMC in Example 3.

state.

- *The transition relation* $\rightarrow \subseteq \mathbf{S} \times (\Sigma \times (0,1]) \times \mathbf{S}$ *is given by* $\mathbf{s} \xrightarrow{e,p_e} \mathbf{s}'$ *iff for some* $a = (E_a, \pi_a) \in A$, *and* $e = (src_e, tgt_e) \in E_a$, *(i)* $\pi_a(e) = p_e$, *(ii)* $(\mathbf{s})_{loc(a)} = src_e$, $(\mathbf{s}')_{loc(a)} = tgt_e$, *and (iii) for* $i \in [n] \setminus loc(a)$, $\mathbf{s}_i = \mathbf{s}'_i$.

Let TS be the transition system associated with a DMC $\mathcal{D}$. To reason about the probabilistic behavior of $\mathcal{D}$ using TS, one must build a σ-algebra over the paths of this transition system and endow it with a probability measure. The major difficulty is that, due to the mix of concurrency and stochasticity, TS is not a Markov chain. In Fig. 4.3, for instance, the sum of the probabilities of the transitions originating from the initial stae (IN_1, IN_2) is 2. To get around this, we first filter out concurrency by working with equivalent classes of paths.

We shall refer to paths in TS as *trajectories*. A finite trajectory of TS from $\mathbf{s} \in \mathbf{S}$ is a sequence of the form $\mathbf{s}_0 e_0 \mathbf{s}_1 \dots \mathbf{s}_{k-1} e_{k-1} \mathbf{s}_k$ such that $\mathbf{s}_0 = \mathbf{s}$ and, for $0 \leq l < k$, $\mathbf{s}_l \xrightarrow{e_l, p_{e_l}} \mathbf{s}_{l+1}$. Infinite trajectories are defined as usual.

For the trajectory $\rho = \mathbf{s}_0 e_0 \mathbf{s}_1 \dots \mathbf{s}_{k-1} e_{k-1} \mathbf{s}_k$, we define $ev(\rho)$ to be the event sequence $e_0 e_1 \dots e_{k-1}$. Again, this notation is extended to infinite tra-

jectories in the natural way. Due to concurrency, one can have infinite trajectories that are not maximal, so we proceed as follows.

Let $\Sigma_i = \{e \mid i \in loc(e)\}$. Suppose ξ is an event sequence (finite of infinite). Then $proj_i(\xi)$ is the sequence obtained by erasing from ξ all events that are not in Σ_i. This leads to the equivalence relation $\approx$ over event sequences given by $\xi \approx \xi'$ iff $proj_i(\xi) = proj_i(\xi')$ for every i. We let $[\xi]$ denote the $\approx$-equivalence class containing ξ and call it a (Mazurkiewicz) trace. For infinite sequences, it is technically more convenient to define traces using equivalence of projections rather than permutation of independent actions.

The partial order relation $\sqsubseteq$ over traces is defined as $[\xi] \sqsubseteq [\xi']$ iff $proj_i(\xi)$ is a prefix of $proj_i(\xi')$ for every i. Finally the trace $[\xi]$ is said to be maximal if for every ξ', $[\xi] \sqsubseteq [\xi']$ implies $[\xi] = [\xi']$. The trajectory ρ is *maximal* iff $[ev(\rho)]$ is a maximal trace. In the transition system of Fig. 4.3, $(IN_1, IN_2)e_h^1(H_1, IN_2)e_T^2(H_1, T_2)e_{ht}((W_1, L_2)e_{w_1})^\omega$ is a maximal infinite trajectory. In fact, in this example all the infinite trajectories are maximal.

4.3.1 The σ-algebra of Trajectories

We denote by $Trj_{\mathbf{s}}$ the set of maximal trajectories from $\mathbf{s}$. Two trajectories can correspond to interleavings of the same partially ordered execution of events. Hence, one must work with equivalence classes of maximal trajectories to construct a probability measure. The equivalence relation $\simeq$ over $Trj_{\mathbf{s}}$ that we need is defined as $\rho \simeq \rho'$ if $ev(\rho) \approx ev(\rho')$. As usual $[\rho]$ will denote the equivalence class containing the trajectory ρ.

Let ρ be finite trajectory from $\mathbf{s}$. Then $\uparrow\rho$ is the subset of $Trj_{\mathbf{s}}$ satisfying $\rho' \in \uparrow\rho$ iff ρ is a prefix of ρ'. We now define $BC(\rho)$, the basic *trj*-cylinder at $\mathbf{s}$ generated by ρ, to be the least subset of $Trj_{\mathbf{s}}$ that contains $\uparrow\rho$ and satisfies

the closure property that if $\rho' \in BC(\rho)$ and $\rho' \simeq \rho''$ then $\rho'' \in BC(\rho)$. In other words, $BC(\rho) = \{[\rho'] \mid \rho' \in Trj_{\mathbf{s}}, [ev(\rho)] \sqsubseteq [ev(\rho')]\}$.

It is worth noting that we could have $BC(\rho) \cap BC(\rho') \neq \emptyset$ without having $\rho \simeq \rho'$. For instance, in Fig. 4.3, let $\rho = (IN_1, IN_2)e_h^1(H_1, IN_2)$ and $\rho' = (IN_1, IN_2)e_t^2(IN_1, T_2)$. Then $BC(\rho)$ and $BC(\rho')$ will have common maximal trajectories of the form $(IN_1, IN_2)e_h^1(H_1, IN_2)e_t^2(H_1, T_2)\ldots$.

We now define $\widehat{SA}(\mathbf{s})$ to be the least σ-algebra that contains the basic *trj*-cylinders at $\mathbf{s}$ and is closed under countable unions and complementation (relative to $Trj_{\mathbf{s}}$).

To construct the probability measure $\widehat{P} : \widehat{SA}(\mathbf{s}) \to [0,1]$ we are after, a natural idea would be to assign a probability to each basic *trj*-cylinder as follows. Let $BC(\rho)$ be a basic *trj*-cylinder with $\rho = \mathbf{s}_0 e_0 \mathbf{s}_1 \ldots \mathbf{s}_{k-1} e_{k-1} \mathbf{s}_k$. Then $\widehat{P}(BC(\rho)) = p_0 \cdot p_1 \cdot \ldots \cdot p_{k-1}$, where $p_\ell = p_{e_\ell}$, for $0 \leq \ell < k$. This is inspired by the Markov chain case in which the probability of a basic cylinder is defined to be the product of the probabilities of the events encountered along the common finite prefix of the basic cylinder. However, showing directly that this extends uniquely to a probability measure over $\widehat{SA}_{\mathbf{s}}$ is very difficult.

We get around this by using the Markov chain $\mathcal{M}$ associated with $\mathcal{D}$ and then embed $\widehat{SA}_{\mathbf{s}}$ into $SA_{\mathbf{s}}$, the σ-algebra generated by the infinite paths in $\mathcal{M}$ starting from $\mathbf{s}$. The standard probability measure over $SA_{\mathbf{s}}$ will then induce a probability measure over $\widehat{SA}_{\mathbf{s}}$.

4.3.2 The Probability Measure for the Trajectory Space

To construct a probability measure over the trajectory space we shall associate infinite paths in $\mathcal{M}$ with maximal trajectories in TS. The Foata normal

form from Mazurkiewicz trace theory will help achieve this. Let $\xi \in \Sigma^\star$. A standard fact is that $[\xi]$ can be canonically represented as a "step" sequence of the form $u_1 u_2 \dots u_k$. More precisely, the Foata normal form of the finite trace $[\xi]$, denoted $FN([\xi])$, is defined as follows [DR95].

- $FN([\epsilon]) = \epsilon$.
- Suppose $\xi = \xi' e$ and $FN([\xi']) = u_1 u_2 \dots u_k$. If there exists $e' \in u_k$ such that $(e', e) \notin I$ then $FN([\xi]) = u_1 u_2 \dots u_k \{e\}$. If not, let ℓ be the least integer in $\{1, 2, \dots, k\}$ such that $e \; I \; e'$ for every $e' \in \bigcup_{\ell \leq m \leq k} u_m$. Then $FN([\xi]) = u_1 \dots u_{\ell-1} (u_\ell \cup \{e\}) u_{\ell+1} \dots u_m$.

For the example shown in Fig. 4.3, $FN(e_h^1 \, e_t^2 \, e_{ht} \, e_{w_1} \, e_{w_1}) = \{e_h^1, e_t^2\} \, \{e_{ht}\} \, \{e_{w_1}\} \, \{e_{w_1}\}$.

This notion is extended to infinite traces in the obvious way. Note that $\xi \approx \xi'$ iff $FN(\xi) = FN(\xi')$.

Conversely, we can extract a (maximal) step sequence from a path in $\mathcal{M}$. Suppose $\mathbf{s}_0 \mathbf{s}_1 \dots$ is a path in $Paths_{\mathbf{s}}$. There exists a unique sequence $u_1 \, u_2 \dots$ such that u_ℓ is a maximal step at $\mathbf{s}_{\ell-1}$ and $\mathbf{s}_\ell$ is the u_ℓ-successor of $\mathbf{s}_{\ell-1}$ for every $\ell > 0$. We let $st(\tau) = u_1 \, u_2 \dots$ and call it the step sequence induced by τ.

This leads to the map $tp : Trj_{\mathbf{s}} \to Paths_{\mathbf{s}}$ given by $tp(\rho) = \tau$ iff $FN(ev(\rho)) = st(\tau)$.

Lemma 1. *tp is well defined.*

Proof. Let $\rho \in Trj_{\mathbf{s}}$ such that $\xi = ev(\rho)$ and $FN([\xi]) = u_1 u_2 \cdot \cdots$. $FN([\xi])$ is an infinite sequence since we have assumed $\mathcal{M}$ is deadlock-free. Since ρ is a maximal trajectory, by definition, $[\xi]$ is a maximal event trace.

We construct a path $\tau = \mathbf{s}_0\mathbf{s}_1 \ldots \mathbf{s}_k \cdots \in Trj_{\mathbf{s}}$ such that $\mathbf{s}_l \xrightarrow{u_l} \mathbf{s}_{l+1}$ for $l \geq 0$, which implies $FN([ev(\rho)]) = st(\tau)$. We construct τ inductively as follows:

- u_1 is a maximal step at $\mathbf{s}$. Hence, $\mathbf{s}_0 = \mathbf{s}$ and there exists s_1 such that $\mathbf{s}_0 \xrightarrow{u_1} \mathbf{s}_1$.

  All events $e \in u_1$ are pairwise independent by the definition of Foata normal form. Also, u_1 is maximal because if not, then there exists an event $e \notin u_1$ which is enabled at $\mathbf{s}$ and is pairwise independent with all events in u_1. However, since $[\xi]$ is a maximal event (and hence Mazurkiewicz) trace, there exists u_t for some $t > 1$ such that $e \in u_t$ which contradicts that $u_1u_2 \cdots$ is a Foata normal form. Finally, there exists at least one event in u_1 that is enabled at $\mathbf{s}$ since $\rho \in Trj_{\mathbf{s}}$. Hence, by the definition of DMC, all events in u_1 are enabled at $\mathbf{s}$ since they are pairwise independent concluding u_1 is a maximal step at $\mathbf{s}$.

- Let us assume we constructed $\mathbf{s}_0\mathbf{s}_1 \cdots \mathbf{s}_{k-1}$ such that $\mathbf{s}_l \xrightarrow{u_l} \mathbf{s}_{l+1}$ for $0 \leq l < k-1$. By similar argument given above, u_{k-1} is a maximal step enabled at $\mathbf{s}_{k-1}$ and hence there exists $\mathbf{s}_k$ such that $\mathbf{s}_{k-1} \xrightarrow{u_{k-1}} \mathbf{s}_k$, thus completing the proof. □

As usual, for $X \subseteq Trj_{\mathbf{s}}$ we define $tp(X) = \{tp(\rho) \mid \rho \in X\}$. It turns out that tp maps each basic cylinder in the trajectory space to a finite union of basic cylinders in the path space. As a result, tp maps every measurable set of trajectories to a measurable set of paths. Consequently, one can define the probability of a measurable set of trajectories X to be the probability of the measurable set of paths $tp(X)$.

To understand how tp acts on the basic cylinder $BC(\rho)$, let $FN(ev(\rho)) = u_1u_2\dots u_k$. We associate with ρ the set of finite paths $paths(\rho) = \{\pi \mid st(\pi) = U_1U_2\dots U_k$ and $u_\ell \subseteq U_\ell$, for $1 \leq \ell \leq k\}$. In other words $\pi \in paths(\rho)$ if it extends each step in $FN(ev(\rho))$ to a maximal step. Then, tp maps $BC(\rho)$ to the (finite) union of the basic cylinders generated by the finite paths in $paths(\rho)$. These observations and their main consequence, namely, the construction of a probability measure over the trajectory space, can be summarized as the following Lemma 2, 3 and 4:

Lemma 2. *Let $\rho = \mathbf{s}_0\mathbf{s}_1\cdots\mathbf{s}_k$ be a finite trajectory at $\mathbf{s}$. We assume $B = BC(\rho)$ to be a basic trj-cylinder from $\mathbf{s}$ and $FN([ev(\rho)]) = u_1u_2\cdots u_k$.*

1. *Suppose $\tau \in Path_{\mathbf{s}}$. Then $\tau \in tp(BC(\rho))$ iff $u_l \subseteq st(\tau)_l$ for $1 \leq l \leq k$. We define $st(\tau)_l$ to be the maximal step appearing in position l of the sequence $st(\tau)$.*

2. *tp(B) is a finite union of basic cylinder sets in $SA_{\mathbf{s}}$ and hence is a member of $SA_{\mathbf{s}}$. Furthermore $P_{\mathbf{s}}(tp(B)) = \prod_{1\leq l\leq k} p_l$ where $p_l = \prod_{e\in u_l} p_e$ for $1 \leq l \leq k$.*

Proof. 1. We need to show that $tp(BC(\rho)) = paths(\rho)$.

Suppose $\rho' \in BC(\rho)$. Then, by definition of basic cylinders, there exists $\rho'' \in BC(\rho)$ such that $\rho' \simeq \rho''$ and ρ is a finite prefix of ρ''. Hence

$$\begin{aligned}
&\rho' \simeq \rho'' \\
&\implies ev(\rho') \approx ev(\rho'') && \text{(definition of } \simeq) \\
&\implies FN([ev(\rho')]) = FN([ev(\rho'')]) && \text{(uniqueness of } FN) \\
&\implies tp(\rho') = tp(\rho'') && \text{(definition of } tp)
\end{aligned}$$

Since $FN([ev(\rho)]) = u_1 u_2 \cdots u_k$ and ρ is a finite prefix of ρ'', $tp(\rho'') \in paths(\rho)$ and hence $tp(\rho') \in paths(\rho)$. Thus $tp(BC(\rho)) \subseteq paths(\rho)$.

To prove the converse, let $\tau \in paths(\rho)$ with $st(\tau) = U_1 U_2 \cdots$. We need to prove $\tau \in tp(BC(\rho))$. By definition of tp and BC, it is sufficent to prove that there exists maximal trajectory ρ' such that $\rho' \in BC(\rho)$ and $FN([ev(\rho')]) = st(\tau)$.

We proceed by induction on k. Suppose $FN(ev(\rho)) = u_1$ with $st(\tau)_1 = U_1$ and $u_1 \subseteq U_1$. Let $U_1 \backslash u_1 = \{e_1, e_2, \ldots, e_m\}$. Then it is easy to see that there exists a trajectory of the form $\rho' = \rho\rho_1\rho_2 \in Trj_s$ such that $ev(\rho_1) = e_1 e_2 \ldots e_m$. Clearly $\rho' \in BC(\rho)$ such that $FN(ev(\rho')) = U_1 = st(\tau)$ and this proves the basis step. The induction step follows from the induction hypothesis applied at the U-successor of $\mathbf{s}$ for each U defined as above.

2. We again proceed by induction on k. Suppose $k = 1$. Let us assume $\{U_1, U_2, \ldots, U_m\}$ to be the set of maximal steps at $\mathbf{s}$ such that $u_l \subseteq U_l$ for $1 \leq l \leq m$. Then, from the previous part, it follows that $tp(BC(\rho)) = \bigcup_{1 \leq l \leq m} \uparrow(\mathbf{s}\mathbf{s}_l)$ where $\mathbf{s}_l$ is the U_l-successor of $\mathbf{s}$ for $1 \leq l \leq m$. This establishes the basis step. The induction step now follows by appealing to the induction hypothesis at $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m$. This proves that $tp(B)$ is a member of $SA_{\mathbf{s}}$. Then using the previous part of the lemma and inducing on k, it is easy to show that $P_{\mathbf{s}}(tp(B)) = \prod_{1 \leq l \leq k} p_l$ where $p_l = \prod_{e \in u_l} p_e$ for $1 \leq l \leq k$.

□

Lemma 3. *Let $B \in \widehat{SA}_{\mathbf{s}}$.*

1. *B is $\simeq$-closed.*

2. $tp(Trj_{\mathbf{s}} - B) = Path_{\mathbf{s}} - tp(B)$.

3. $tp(B) \in SA_{\mathbf{s}}$.

Proof. 1. For $\rho \in B$ and $\rho' \simeq \rho$, by definition of $B \in \widehat{SA}_{\mathbf{s}}$ it follows that $\rho' \in B$. Hence B is $\simeq$-closed.

2. Let $\rho \in Trj_{\mathbf{s}} - B$. If $tp \in tp(B)$ then there exists $\rho' \in B$ such that $tp(\rho') = tp(\rho)$. But this will imply that $FN([ev(\rho)]) = FN([ev(\rho')])$ which in turn would imply that $ev(\rho) \approx ev(\rho')$. But this implies that $\rho \simeq \rho'$ leading to the contradiction that $\rho \in B$ since B is $\simeq$-closed. This shows that $tp(Trj_{\mathbf{s}} - B) \subseteq Path_{\mathbf{s}} - tp(B)$.

   Now assume $\tau \in Path_{\mathbf{s}} - tp(B)$. It is easy to show that there exists a $\rho \in Trj_{\mathbf{s}}$ such that $tp(\rho) = \tau$. Clearly $\rho \in Trj_{\mathbf{s}} - B$. Hence $Path_{\mathbf{s}} - tp(B) \subseteq tp(Trj_{\mathbf{s}} - B)$ which concludes the proof.

3. If B is a basic cylinder at $\mathbf{s}$, then the result follows from part *(iii)* of Lemma 2. By definition, $\widehat{SA}_{\mathbf{s}}$ is closed under complementation and countable union. Since $Path_{\mathbf{s}} - tp(B) \in SA_{\mathbf{s}}$, from the previous part we now have $tp(Trj_{\mathbf{s}} - B) \in SA_{\mathbf{s}}$. Finally, if $\{B_k\}$ is a countable collection such that $B_k \in \widehat{SA}_{\mathbf{s}}$ and $tp(B_k) \in SA_{\mathbf{s}}$ for each k, $tp(\bigcup_k B_k) = \bigcup_k tp(B_k) \in SA_{\mathbf{s}}$.

   Hence for any $B \in \widehat{SA}_{\mathbf{s}}$, $tp(B) \in SA_{\mathbf{s}}$. □

Lemma 4. *$\widehat{P}_{\mathbf{s}}$ is well defined.*

Proof. Since tp is well-defined and $P_{\mathbf{s}}$ is a probability measure, $\widehat{P}_{\mathbf{s}}$ is a well-defined function. To prove that $\widehat{P}_{\mathbf{s}}$ is a well-defined measure on $\widehat{SA}_{\mathbf{s}}$, we prove the following:

- $\widehat{P}_{\mathbf{s}}(\widehat{SA}_{\mathbf{s}}) = P_{\mathbf{s}}(tp(\widehat{SA}_{\mathbf{s}})) = P_{\mathbf{s}}(SA_{\mathbf{s}}) = 1$, since $P_{\mathbf{s}}$ is a probability measure.

- Let $\{B_k\}$ be a countable collection of pairwise disjoint sets in $\widehat{SA}_{\mathbf{s}}$. Clearly $\{tp(B_k)\}$ is a countable collection of pairwise disjoint sets in $SA_{\mathbf{s}}$. We now have:

$$\begin{aligned}
\widehat{P}_{\mathbf{s}}(\bigcup_k B_k) &= P_{\mathbf{s}}(tp(\bigcup_k B_k)) && \text{(definition of } \widehat{P}_{\mathbf{s}}) \\
&= P_{\mathbf{s}}(\bigcup_k tp(B_k)) && (\textstyle\bigcup_k \text{ is countable union}) \\
&= \sum_k P_{\mathbf{s}}(tp(B_k)) && \text{(part } \textit{(iii)} \text{ of Lemma 3)} \\
&= \sum_k \widehat{P}_{\mathbf{s}}(B_k) && \text{(definition of } \widehat{P}_{\mathbf{s}})
\end{aligned}$$

Hence, $\widehat{P}_{\mathbf{s}}$ is a probability measure. □

Note that while a finite path in $\mathcal{M}$ always induces a maximal step sequence, a finite trajectory, in general, does not have this structure. Some components can get ahead of others by an arbitrary amount. The lemma above states that, despite this, any finite trajectory defines a finite set of basic cylinders whose overall probability can be easily computed, by taking the product of the probabilities of the events encountered along the trajectory. This helps considerably when verifying the properties of $\mathcal{M}$. In particular, local reachability properties can be checked by exercising only those components that are relevant.

Going back to our running example, let $\rho_t = (IN_1, IN_2)e_t^1(T_1, IN_2)$, and $X_t = \uparrow\rho_t$. Let $\rho'_t = (IN_1, IN_2)e_t^2(IN_1, T_2)$, and $X'_t = \uparrow\rho_t$. Assume ρ_h, X_h, ρ'_h and X'_h are defined similarly. Then $\widehat{P}(X_t) = \widehat{P}(X'_h) = 0.5$, while $\widehat{P}(X_h \cup X_t) = 1$. On the other hand, due to the fact that e_h^1 and e_h^2 are independent, we have $\widehat{P}(X_h \cup X'_h) = 0.75$.

4.4 The Logic and Model Checking Technique

To bring out the applicability of the DMC formalism and its interleaved semantics, we formulate a statistical model checking procedure. The specification logic $PBLTL^{\otimes}$ (product *PBLTL*) is a simple generalization of probabilistic bounded linear time temporal logic (*PBLTL*) [JCL+09] that captures Boolean combinations of local properties of the components. The logic can express interesting global reachability properties as well since the Boolean connectives can capture—in a limited fashion—the way the components influence each other.

We assume a collection of pairwise disjoint sets of atomic propositions $\{AP_i\}$. As a first step, the formulas of $BLTL^{\otimes}$ are given as follows.

- $ap \in AP_i$ is a $BLTL^{\otimes}$ formula and $type(ap) = \{i\}$,
- If φ and φ' are $BLTL^{\otimes}$ formulas with $type(\varphi) = type(\varphi') = \{i\}$ then so is $\varphi \mathbf{U}_i^t \varphi'$ where t is a positive integer. Further, $type(\varphi \mathbf{U}_i^t \varphi') = \{i\}$. As usual, $F^t\varphi$ abbreviates $(true\mathbf{U}^t\varphi)$ and $G^t\varphi$ is defined as $\neg F^t \neg\varphi$,
- If φ and φ' are $BLTL^{\otimes}$ formulas then so are $\neg\varphi$ and $\varphi \vee \varphi'$ with $type(\neg\varphi) = type(\varphi)$ and $type(\varphi \vee \varphi') = type(\varphi) \cup type(\varphi')$.

The formulas of $PBLTL^{\otimes}$ are given by:

- Suppose φ is a $BLTL^{\otimes}$ formula and γ a rational number in the open interval $(0,1)$. Then $Pr_{\geq\gamma}(\varphi)$ is a $PBLTL^{\otimes}$ formula,
- If ψ and ψ' are $PBLTL^{\otimes}$ formulas then so are $\neg\psi$ and $\psi \vee \psi'$.

To define the semantics, we project each trajectory to its components. For $\mathbf{s} \in \mathbf{s}$ and $i \in [n]$ we define $Proj_i : Trj^{fin}_{\mathbf{s}} \to S_i^+$ inductively.

- $Proj_i(\mathbf{s}) = \mathbf{s}_i$,
- Suppose $\rho = \mathbf{s}_0 e_o \mathbf{s}_1 \dots \mathbf{s}_m e_m \mathbf{s}_{m+1}$ is in $Trj^{fin}_{\mathbf{s}}$ and $\rho' = \mathbf{s}_0 e_0 \mathbf{s}_1 \dots \mathbf{s}_m$. If $i \in loc(e_m)$ then $Proj_i(\rho) = Proj_i(\rho')(\mathbf{s}_{m+1})_i$. Otherwise $Proj_i(\rho) = Proj_i(\rho')$.

We lift $Proj_i$ to infinite trajectories in the obvious way—note that $Proj_i(\rho)$ can be a finite sequence for the infinite trajectory ρ. We assume a set of local valuation functions $\{V_i\}$, where $V_i : S_i \to 2^{AP_i}$. Let φ be a $BLTL^{\otimes}$ formula with $type(\varphi) = \{i\}$. We begin by interpreting such formulas over sequences generated by the alphabet S_i. For $\varrho \in S_i^+ \cup S_i^\omega$, the satisfaction relation $\varrho, k \models_i \varphi$, with $0 \leq k \leq |\varrho|$, is defined as follows.

- $\varrho, k \models_i ap$ for $ap \in AP_i$ iff $ap \in V_i(\varrho(k)(i))$, where $\varrho(k)(i)$ is the S_i-state at position k of the sequence ϱ,
- $\neg$ and $\vee$ are interpreted in the usual way,
- $\varrho, k \models_i \varphi_1 \mathbf{U}^t_i \varphi_2$ iff there exists ℓ such that $k \leq \ell \leq max(k+t, |\varrho|)$ with $\varrho, \ell \models_i \varphi_2$, and $\varrho, m \models_i \varphi_1$, for $k \leq m < \ell$.

As usual, $\varrho \models_i \varphi$ iff $\varrho, 0 \models_i \varphi$. Next, suppose φ is a $BLTL^{\otimes}$ formula and $\rho \in Path_{\mathbf{s}}$. Then the relation $\rho \models_{\mathbf{s}} \varphi$ is defined as follows.

- If $type(\varphi) = \{i\}$ then $\rho \models_{\mathbf{s}} \varphi$ iff $Proj_i(\rho) \models_i \varphi$,
- Again, $\neg$ and $\vee$ are interpreted in the standard way.

Given a formula φ in $BLTL^{\otimes}$ and a global state $\mathbf{s}$, we define $Trj_{\mathbf{s}}(\varphi)$ to be the set of trajectories $\{\rho \in Trj_{\mathbf{s}} \mid \rho \models_s \varphi\}$.

Lemma 5. *For every formula φ, $Trj_{\mathbf{s}}(\varphi)$ is a member of $\widehat{SA}(\mathbf{s})$.*

Proof. Let $Path_{\mathbf{s}}(\varphi)$ be the set of paths in the Markov chain $\mathcal{M}$ that satisfies the formula φ. It is easy to see that $Path_{\mathbf{s}}(\varphi)$ is a member of $SA_{\mathbf{s}}$. We then use Lemma 3 to obtain the result. □

The semantics of $PBLTL^{\otimes}$ is now given by the relation $\mathcal{D} \models^{trj}_{\mathbf{s}} \psi$, defined as:

1. Suppose $\psi = Pr_{\geq\gamma}(\varphi)$. Then $\mathcal{D} \models^{trj}_{\mathbf{s}} \psi$ iff $\widehat{P}(Path_{\mathbf{s}}(\varphi)) \geq \gamma$,
2. Again, the interpretations of $\neg$ and $\vee$ are the standard ones.

For the example in Fig. 4.1, one can assert $Pr_{\geq 0.99}((F^7(L_1) \wedge F^7(W_2)) \vee (F^7(W_1) \wedge F^7(L_2)))$. Here, the local states also serve as the atomic propositions. Hence, the formula says that with probability ≥ 0.99, a winner will be decided within 7 rounds.

We write $\mathcal{D} \models^{trj} \psi$ for $\mathcal{D} \models^{trj}_{\mathbf{s}^{in}} \psi$. The model checking problem is to determine whether $\mathcal{D} \models^{trj} \psi$. We shall adapt the SMC procedure developed in [You04] to solve this problem approximately.

4.4.1 The Statistical Model Checking Procedure

We note that in the Markov chain setting, given a BLTL formula and a path in the chain, there is a bound k that depends only on the formula such that

we can decide whether the path is a model of the formula by examining just a prefix of the path of length k [JCL+09]. By the same reasoning, for a $BLTL^{\otimes}$ formula φ, we can compute a vector of bounds $(k_1, k_2, \ldots, k_n)$ that depends only on φ such that for any trajectory ρ starting from $\mathbf{s}^{in}$, we only need to examine a finite prefix ρ' of ρ that satisfies $|Proj_i(\rho')| \geq k_i$, for $1 \leq i \leq n$. The complication in our setting is that it is not guaranteed that one can generate such a prefix with bounded effort. This is due to the mix of concurrency and stochasticity in DMCs. More precisely, at a global state $\mathbf{s}$, one may need to advance the history of the agent i but this may require first executing an event $e = (src_e, tgt_e) \in E_a$ at $\mathbf{s}$ that does not involve the agent i. However, there could also be another event $e' = (src_{e'}, tgt_{e'}) \in E_a$ enabled at $\mathbf{s}$ such that $src_e = src_{e'}$. Since one must randomly choose one of the enabled events according to the underlying probabilities, one may repeatedly fail to steer the computation towards a global state in which the history of i can be advanced. A second complication is that starting from the current global state it may be impossible to reach a global state at which some event involving i is enabled.

To cope with this, we maintain a count vector $(c_1, c_2, \ldots, c_n)$ that records how many times each component has moved along the trajectory ρ that has been generated so far. A simple reachability analysis will reveal whether a component is *dead* in the current global state — that is, starting from the current state, there is no possibility of reaching a state in which an event involving this agent can be executed. If this is the case for the agent i or the c_i is already the required bound then remove it from the current set of active agents. If the current set of active agents is not empty, we execute, one by one, all the enabled actions—using a fixed linear order over the

set of actions—followed by one move by each of the participating agents, according to the underlying probabilities. Recall that action a is enabled at $\mathbf{s}$ iff $\mathbf{s}_a \in en_a$. Due to the determinacy of synchronizations, the global state thus reached will depend only on the probabilistic moves chosen by the participating agents. We then update the count vector to $(c'_1, c'_2, \ldots, c'_n)$ and mark the new dead components.

We show in Theorem 1 that, continuing in this manner, with probability 1 we will eventually generate a finite trajectory $\widehat{\rho}$ and reach a global state $\mathbf{s}$ with no active agents. We then check if $\widehat{\rho}$ satisfies φ and update the score associated with the statistical test described as follows.

Theorem 1. *With probability* 1*, the method of generating a finite trajectory described in section 4.4.1 will terminate.*

Proof. The core task is to check whether $\mathcal{D} \models^{trj}_{\mathbf{s}} \widehat{P}_{\geq\gamma}(\varphi)$ where φ is a $BLTL^{\otimes}$ formula. Let $(k_1, k_2, \ldots, k_n)$ be the bound vector obtained from φ as follows. If $i \notin type(\varphi)$ then $k_i = 0$. If $i \in type(\varphi)$ then k_i is the maximum of the bounds associated (in the usual way) with the i-type formulas appearing in φ.

Let ρ be a finite trajectory such that the corresponding count vector is $(c_1, c_2, \ldots, c_n)$. In other words the agent i has executed c_i events so far along ρ. Let $\mathbf{s}^{in} \xrightarrow{\rho} \mathbf{s}$. Then, ρ is *terminal* if for every i either $c_i \geq k_i$ or there exists i such that $c_i < k_i$ and the component i is dead at $\mathbf{s}$. It is easy to show that one can effectively determine if a finite trajectory is terminal. Let TER be the set of terminal finite trajectories and $\mathrm{Y} = \cup_{\rho \in TER} BC(\rho)$. Clearly TER is a countable set and hence Y is in $\widehat{SA}$. It is now enough to prove that $\widehat{P}(\mathrm{Y}) = 1$.

It will be more convenient to carry out the proof in the Markov chain

setting. We first blow up $\mathcal{M}$ into the larger $\widehat{\mathcal{M}}$ as follows. Let $\widehat{\mathbf{S}} = \mathbf{S} \times (C_1 \times C_2 \times \cdots \times C_n)$ such that $C_i = \{0, 1, \ldots, k_i\} \cup \{\omega_i\}$ for every i. Here, ω_i denotes "large" and satisfies $k_i < \omega_i$ and $\omega_i + 1 = \omega_i$ for every i.

We define $\widehat{\mathcal{M}}$ via: $\widehat{\mathcal{M}}((\mathbf{s}, \mathbf{c})(\mathbf{s}', \mathbf{c}')) = p$ iff $\mathcal{M}(\mathbf{s}, \mathbf{s}') = p$. Morever, if $\mathbf{s}'$ is the u-successor of $\mathbf{s}$ then for each i the following conditions hold. Suppose $i \in loc(u)$. Then $\mathbf{c}'(i) = \mathbf{c} + 1$ if $\mathbf{c}(i) < k_i$ and $\mathbf{c}'(i) = \omega_i$ if $\mathbf{c}(i) = k_i$ or $\mathbf{c}(i) = \omega_i$. On the other hand, $\mathbf{c}'(i) = \mathbf{c}(i)$ if $i \notin loc(u)$.

It is easy to verify that $\widehat{\mathcal{M}}$ is a finite state Markov chain. The initial state is $(\mathbf{s}^{in}, \mathbf{0})$, where $\mathbf{0}(i) = 0$ for each i. In what follows we will often suppress the mention of $(\mathbf{s}^{in}, \mathbf{0})$.

Let τ be a finite path in $\widehat{\mathcal{M}}$ that ends at $(\mathbf{s}, \mathbf{c})$. We say $(\mathbf{s}, \mathbf{c})$ is *final* iff $k_i \leq \mathbf{c}_i$ for every i or there exists i such that $\mathbf{c}(i) < k_i$ and i is dead at $(\mathbf{s}, \mathbf{c})$. Then τ is *terminal* iff $(\mathbf{s}, \mathbf{c})$ is final. Similar to TER, we now define $\widehat{TER}$ to be the set of terminal finite paths in $\widehat{\mathcal{M}}$. From the definitions it follows that $tp(TER) = \widehat{TER}$ and hence it suffices to prove that $P(\cup_{\tau \in \widehat{TER}} BC(\tau)) = 1$. Here, P denotes the usual probability measure over $SA_{(\mathbf{s}^{in}, \mathbf{0})}$ in $\widehat{\mathcal{M}}$. We also note that $\widehat{TER}$ is a countable set and hence $\bigcup_{\tau \in \widehat{TER}} BC(\tau)$ will be a member of $SA_{(\mathbf{s}^{in}, \mathbf{0})}$.

Next, let $(\mathbf{s}, \mathbf{c}) \in \widehat{\mathcal{S}}$. Let $\widehat{\mathbf{W}}$ be the set of final states. Now as shown in [BKo8], $P(\bigcup_{\tau \in \widehat{TER}} BC(\tau)) = 1$ follows if we can show that $(\mathbf{s}^{in}, \mathbf{0}) \in \widehat{\mathbf{S}} - Pre^*(\widehat{\mathbf{S}} - Pre^*(\widehat{\mathbf{W}}))$ where $Pre^*(\widehat{\mathbf{x}})$ for $\widehat{\mathbf{S}} \subseteq \widehat{\mathbf{S}}$ is the least subset of $\widehat{\mathbf{S}}$ that contains $\widehat{\mathbf{X}}$ and satisfies: If $\widehat{\mathbf{s}} \in \widehat{\mathbf{X}}$ and $\widehat{\mathcal{M}}(\widehat{\mathbf{s}'}, \widehat{\mathbf{s}}) > 0$ then $\widehat{\mathbf{s}'} \in Pre^*(\widehat{\mathbf{X}})$.

We now claim that $Pre^*(\widehat{\mathbf{W}}) = \widehat{\mathbf{S}}$. It is easy to see that proving this is sufficient since this leads to $\widehat{\mathbf{S}} - Pre^*(\widehat{\mathbf{S}} - Pre^*(\widehat{\mathbf{W}})) = \widehat{\mathbf{S}}$ and trivially $(\mathbf{s}^{in}, \mathbf{0}) \in \widehat{\mathbf{S}}$. We recall $\widehat{\mathbf{Y}} \subseteq \widehat{\mathbf{S}}$ is a strongly connected component of $\mathcal{M}$ iff there is a path from $\widehat{\mathbf{s}}$ to $\widehat{\mathbf{u}}$ for every $\widehat{\mathbf{s}}, \widehat{\mathbf{u}} \in \widehat{\mathbf{Y}}$. Let $\{SC_1, SC_2, \ldots, SC_r\}$

be the set of strongly connected components (SCCs) of $\widehat{\mathbf{M}}$. The relation $\preceq$ over SCCs is given by: $SC \preceq SC'$ iff there exists a state $\widehat{\mathbf{s}} \in SC$, a state $\widehat{\mathbf{u}} \in SC'$ and a path from $\widehat{\mathbf{s}}$ to $\widehat{\mathbf{u}}$ in $\widehat{\mathbf{M}}$. Clearly, $\preceq$ is a partial order relation and the maximal elements under *preceq* will be called the bottom strongly connected components (BSCCs).

Now let $\widehat{\mathbf{Y}}$ be a BSCC and $(\mathbf{s}, \mathbf{c}), (\mathbf{s}', \mathbf{c}') \in \widehat{\mathbf{Y}}$. Then it must be the case that $\mathbf{c} = \mathbf{c}'$. Next, suppose $\mathbf{c}(i) < k_i$ for some i. Then i must be dead at $(\mathbf{s}, \mathbf{c})$ which implies that i is dead at every state at $\widehat{\mathbf{Y}}$. Consequently, every state in $\widehat{\mathbf{Y}}$ is final. By definition of BSCCs, we now have $Pre^*(\widehat{\mathbf{W}}) = \widehat{\mathbf{S}}$, thus concluding the proof. □

Statistical test

The parameters for the test are δ, α, β, where δ is the size of the indifference region and (α, β) is the strength of the test, with α bounding the Type I errors (false positives) and β bounding the Type II errors (false negatives). These parameters are to be chosen by the user. We generate finite i.i.d. sample trajectories sequentially. We associate a Bernoulli random variable x_ℓ with the sample $(\rho)_\ell$ and set $x_\ell = 1$ if $(\rho)_\ell \in Trj_{\mathbf{s}^{in}}(\varphi)$ and set $x_\ell = 0$ otherwise. We let $c_m = \sum_\ell x_\ell$ and compute the score $SPRT$ via

$$SPRT = \frac{(\gamma^-)^{c_m}(1-\gamma^-)^{n-c_m}}{(\gamma^+)^{c_m}(1-\gamma^+)^{n-c_m}}.$$

Here $\gamma^+ = \gamma + \delta$ and $\gamma^- = \gamma - \delta$. If $SPRT \leq \frac{\beta}{1-\alpha}$, we declare $\mathcal{D} \models^{trj} \widehat{P}_{\geq r}\varphi$. If $SPRT \geq \frac{1-\beta}{\alpha}$, we declare $\mathcal{D} \not\models^{trj} \widehat{P}_{\geq \gamma}\varphi$. Otherwise, we draw one more sample and repeat.

This test is then extended to handle formulas of the form $\neg\psi$ and $\psi_1 \vee \psi_2$ in the usual way [JCL+09]. It is easy to establish the correctness of this

statistical model checking procedure.

4.5 Experimental Evaluation

We have tested our SMC procedure on two probabilistic distributed algorithms: (i) a leader election protocol for a unidirectional ring of anonymous processes by Itai and Rodeh [IR90, Fok06] and (ii) a randomized solution to the dining philosophers problem [PZ86]. Both these algorithms—for large instances—exhibit a considerable degree of concurrency since any two agents that do not share a neighbor can execute independently. We focused on approximately verifying termination properties of these algorithms to bring out the scalability and the performance features of our SMC technique. We also compared our results with the corresponding ones obtained using the statistical model checking tool PLASMA [BCLS13].

In the leader election protocol, each process randomly chooses an identity from $\{1, 2, \ldots, N\}$, and passes it on to its neighbor. If a process receives an identity lower than its own, the message is dropped. If the identity is higher than its own, the process drops out of the election and forwards the message. Finally, if the identity is the same as its own, the process forwards the message, noting the identity clash. If an identity clash is recorded, all processes with the highest identity choose a fresh identity and start another round.

Since the initial choice of identities for the N processes can be done concurrently, in the global Markov chain, there will be N^N possible moves. However, correspondingly in the interleaved semantics, there will be N^2 transitions from the initial state.

We have built a DMC model of this system in which each process and

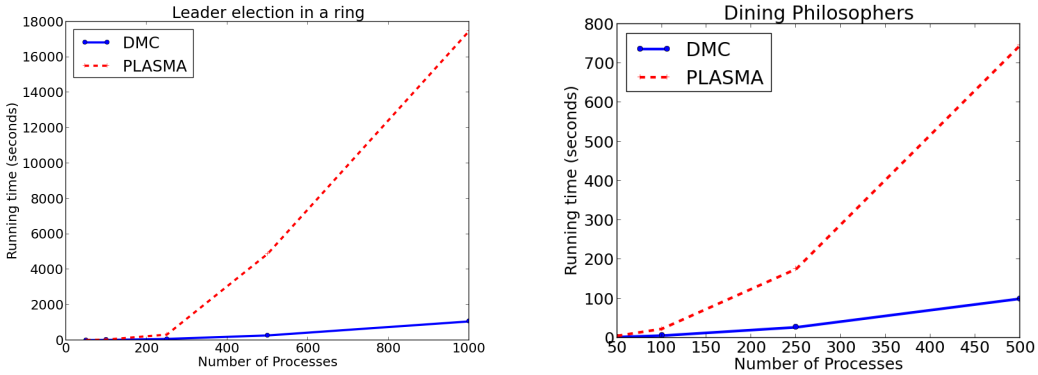


Figure 4.4: Comparison of simulation times in DMC and PLASMA.

channel is an agent. Messages are transferred between processes and channels via synchronizations while ensuring this is done using a deterministic protocol. For simplicity, all channels in our implementation have capacity 1. We can easily construct higher capacity channels by cascading channels of capacity 1 while staying within the DMC formalism.

The challenge in modeling the dining philosophers problem as a DMC is to represent the forks between philosophers, which are typically modeled as shared variables. We use a deterministic round robin protocol to simulate these shared variables. The same technique can be used for a variety of other randomized distributed algorithms presented as case studies for PLASMA.

We ran our trajectories based SPRT procedure written in Python programming language on a Linux server (Intel Xeon 2.30 GHz, 16 core, 72GB RAM). For the first example, we verified that a leader is elected with probability above 0.99 within K rounds, for a ring of N processes for various values of N up to 1000. For each N, it turned out that $K = N$ is a good choice for K as explained below.

The termination property was specified as follows. Let L_i denote the boolean variable which evaluates to true iff in the current global state, node

i has been elected as the leader. Then for N processes with $K = N$, the specification is:

$$Pr_{\geq 0.99}(\bigvee_{i=1}^{N}[F^N(\neg L_1) \wedge \cdots \wedge F^N(\neg L_{i-1}) \wedge F^N(L_i) \wedge F^N(\neg L_{i+1}) \wedge \cdots \wedge F^N(\neg L_n)]).$$

For the dining philosophers, we verified that with probability above 0.95 every philosopher eventually eats, up to $N = 500$ philosophers. In both the experiments, we set the bound on both the Type-I and Type-II errors to be 0.01 and the indifference region to be 0.01. We tested the same properties with the same statistical parameters using the PLASMA model. PLASMA supports parallel execution and multithreading. Since our DMC implementation is currently sequential, we restricted PLASMA to single-threaded sequential execution for a meaningful comparison.

In Fig. 4.4, we have compared the running time for SPRT model-checking in the DMC model with that for PLASMA. The x-axis is the number of processes in the system and the y-axis is the running time, in seconds. We have not been able to determine from the literature how PLASMA translates the model into a DTMC. Consequently we could only compare the simulation times at the system level while treating the PLASMA tool as a black box. In case of PLASMA, the specifications use time bounds which we took it to imply the number of time steps for which the model is simulated. We found that for $N = 1000$, PLASMA verifies the termination property to be true if the time bound is set to be $10,000$. Further, increasing the bound does not cause the simulation times to change. Hence we fixed the time bound to be $10,000$ for all choices of N in the PLASMA setting. In the DMC setting we found that setting $K = N$ caused our implementation to verify

the termination property to be true. Again, increasing this to larger number of rounds does not change the simulation times. For this reason we fixed $K = N$ for each N.

The experiments show that as the model size increases, the running time increase for the DMC approach is almost linear whereas for PLASMA it is more rapid. The results also show the significant performance and scalability advantages of using the interleaved semantics approach based on DMC models. We expect further improvements to be easily achieved via a parallel implementation.

Chapter 5

Quantitative Analysis of Operational Processes

An operational process is an interlinked set of tasks that produces a specific service or product. Organizations from diverse industry sectors—such as health care, finance services, and supply chain management—have large processes consisting of many different components that are dependent on each other. The atomic building block of such processes are tasks. No matter how large and diverse, organizations have a finite amount of resources at their disposal. These resources are capable of performing the tasks in the process, but all resources may not be able to perform all tasks, or be efficient at in terms of cost and time. Organizations are always trying to improve upon the performance of their processes in terms of various *Key Performance Indicators (KPIs)*.

The need for an underlying formal model for operational processes has been long felt. Workflow nets (WF nets, a class of Petri nets), equipped with clean graphical notation and abundance of analysis techniques, have served as a solid framework for modeling process activities [Aal97, AH04, Aal14].

The design and analysis of processes has been consolidated into a field named Business Process Management (BPM) and workflow nets pioneered as the modeling formalism of BPM systems [Aal15, vdAvHtH+11].

Traditionally, workflow nets are designed as tasks intertwined with decision gates. The frequently occurring design patterns are provided as templates called workflow patterns [RHM06]. However, operational processes often incorporate stochastic decision points due to the increasing complexity of such systems as well as the inclusion of uncontrollable components in the processes. On top of that, the processes are hyper-scaled and involve a large number of instances running in parallel. For example, a pizza company may have hundreds of orders being processed in parallel, or a financial institution handling many loan applications at the same time. Such processes are nonetheless resource-constrained, and the company policy or a dedicated managerial body schedules how the resources get assigned to the tasks spanning the parallel instances.

Simulation is a common practice in the process management industry. Many process analysis tools natively support simulation as well. The state-of-the-art approach for analyzing stochastic operational processes is to simulate the system an arbitrary number of times. If, for example, p% of the simulations satisfy the desired property, one claims that the property is true with probability at least p, often with confidence interval estimates of certain parameters. However, the deductions are often difficult to justify and can be arbitrarily far from truth – Wil van der Aalst correctly points out that "simulation does not provide any proof" [Aal15].

In a nutshell, we are interested to model resource-constrained operational processes (i) that are stochastic in nature, (ii) incorporate nondetermin-

ism in terms of resource allocation, (iii) allow hyper-scalability, (iv) provide modeling freedom, and (v) can accommodate simulation techniques with *provable error bounds*. Even when the probability distributions can be measured or approximated from domain knowledge or historical data [Aal11], model-based analysis [Aal13] of such systems is hard due to the interplay between stochastic behavior, concurrency, time taken to perform tasks, and resource contention.

The most prominent modeling formalism for stochastic analysis of business processes is (generalized) stochastic WF nets [Rei03, CYFM02]. These models use timed transitions with exponentially distributed firing delay. A few recent works [SW09, MM07, BFV12] also demonstrate a generic Markovian analysis that is mostly applicable to rigidly structured WF systems. The work of [Her14] focuses on modeling BPM systems as Markov decision processes in the language of PRISM [HKNP06]. These models are either very simplistic in nature, where block-like patterns are chained together, or hard-to-tackle specifications involving arbitrary nondeterminism that cannot be readily adapted for sound simulation techniques. Most importantly, extending these approaches to model processes with shared resources across multiple cases is not obvious.

Performance evaluation of resource-constrained processes has seen a number of approaches in last two decades (see [OLRR12] for a detailed discussion). A serious drawback that is often present in such modeling formalisms is the ability to model finite resources shared across multiple instances of the process model – for example, [Rei03, LWC+02] assume infinite resources and a single case at a time, while [AHK+02] use queueing theory and do not allow concurrency. A more comprehensive approach

is to use (colored) generalized stochastic Petri nets (GSPN) and (colored) Petri nets for evaluating the performance of operational processes [OLRR12, Fer94, SR95, NAR05, DFZ00]. Some of these approaches support only a few performance metrics such as throughput. The approach in [OLRR12] to use GSPN when modeling processes is the most widely covered. It supports multiple cases and finite resources but does not allow nondeterminism since the underlined semantics is a continuous-time Markov chain.

We formally introduce resource-constrained processes and then use a timed extension of the Distributed Stochastic Model (DSM) from Chapter 3 to model the behavior of the processes. Our work is the first to (i) provably bound the error of analysis of operational processes with finite resources shared across multiple cases arriving at different time points and (ii) provide a sound analysis of the sample size of simulation.

5.1 Resource-constrained Processes (RCP)

A *resource-constrained process (RCP)* has a finite set of resources allocated across replicated instances of a stochastic process model. A *process model* is a set of tasks with logical and temporal dependencies among each other. Each task is mapped to one of the available resources that can perform the task. Each task performed by a resource has an execution time, ideally drawn from a probability distribution. For simplicity, we assume that the time taken by a resource to complete a task is fixed – say the mean value of the distribution. Resources are assigned to the tasks following a predetermined allocation strategy.

A *case* is an instance of the process model. Multiple cases run in parallel, sharing the same set of finite resources. We study systems with a fixed

number of cases arriving within a given period of time. The cases need not start simultaneously and may follow a process arrival rate (such as the Poisson process).

We observe that the tasks and resources can be considered as individual agents that behave independently and communicate among each other. There are two types of communications among them: (i) task-task interaction, where a completed task passes the thread of control to some other tasks, and (ii) task-resource interaction, which describes the allocation and execution of a task to a resource.

Hence, an RCP consists of two main components: (i) the process model, instantiated as a fixed number of cases, and (ii) a finite set of resources. An RCP is then governed by a resource allocation strategy.

5.1.1 The Process Model

A process model in an RCP consists of a start state, a finite set of tasks and an end state. The tasks in the process model are combined in sequence or parallel. We assume that the control flow is probabilistic: a discrete probability distribution is attached to each set of outgoing choices while tasks with a unique choice have outgoing probability 1. Each case is an instantiated copy of the process.

Example 4. *An example of an RCP is the loan/overdraft process in a financial institution, where the cases correspond to applications from different customers. Tasks in the process may include stages such as 'submitting', 'reviewing', 'accepting' or 'declining' the application.*

We use a process model that has been mined from a real-life event log of loan/overdraft applications of a large financial institution [BA13]. Along with

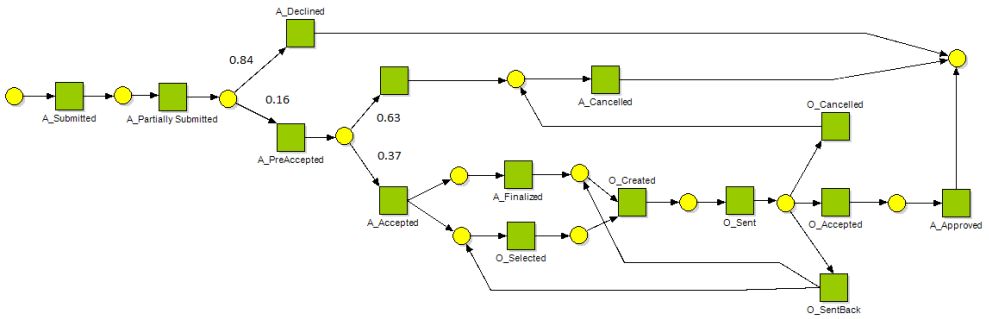


Figure 5.1: An example process of an RCP system (in Petri net notation).

the process, the average time for each resource to perform a particular task has been mined. The event log is obtained from BPI Challenge 2012 [vD12].

Fig. 5.1 demonstrates, in Petri net notation, a process for loan/overdraft applications of a large financial institution. We consider two sub-processes—namely the application and offer sub-processes—of the overall loan/overdraft application process. The tasks of the application and offer sub-processes are prefixed with "`A_`*" and "*`O_`*" respectively. From historical data, we estimate the probability values of outgoing edges from places in the Petri net. Unmarked edges have probability* 1*.*

The process model starts with the submission of an application (`A_Submitted`*). An application can be declined (*`A_Declined`*) if it does not pass any checks. The probability of an application being declined outright is estimated to be* 0.84*. Applications that pass the checks are pre-accepted (*`A_PreAccepted`*) with probability* 0.16*. Often additional information is obtained by contacting the customer by phone. Based on this information, an application can be cancelled (*`A_Cancelled`*) with probability* 0.63 *or accepted (*`A_Accepted`*) with probability* 0.37*. Once an application is accepted, it is finalized (*`A_Finalized`*) and, in parallel, an offer is selected for the customer (*`O_Selected`*). An offer is then created (*`O_Created`*) and sent to eligible applicants and their responses are assessed (*`O_Sent`*). A customer then may (i) accept the offer (*`O_Accepted`*) with probability* 0.01*, (ii) cancel the offer*

Table 5.1: Resource-task matrix (truncated) with average time taken by the resource.

	10228	**10629**	**10779**	**10859**	**10861**	**10862**
A_Accepted	2521			30435	47184	53086
A_Approved						
A_Cancelled			51106			
A_Declined						
A_Finalized	0			35	62	0
A_PartlySubmitted						
A_Preaccepted	31897	2186		5775	9823	1061
A_Submitted						
O_Cancelled				173747	140840	32
O_Created	47			1	1	2
O_Selected	130			0	48	38587
O_Sent	0			0	0	0
O_Sent_Back						
Dummy	0	0	0	0	0	0
O_Accepted	0	0	0	0	0	0

*(*O_Cancelled*) with probability* 0.95*, or (iii) send the offer back (*O_SentBack*) with probability* 0.04*. If an offer is sent back, a new offer is created for the customer. If the offer is accepted, the application is finally approved (*A_Approved*). Declining, cancellation or approval signals the end of the application.*

5.1.2 Resources

A finite set of resources is assumed, each capable of performing a subset of tasks in the process. Different resources can take different amount of time and cost for performing the same task. We may further group resources with the same behavior and capability into roles.

In the running example, we profiled the top 46 of the busiest resources from the event log. The resource-task matrix with cell values representing

the average time taken by the resource on the particular task is shown in Table 5.1 (truncated to 6 resources). The columns indicate the resources and the rows indicate the tasks. If a cell is empty, it indicates that the task is not assigned to the particular resource. If the value in a cell is 0, the resource performs the task instantly. The unit of time is irrelevant to our analysis. For this example, we only consider the time of events and do not discuss the cost separately.

5.1.3 The Resource Allocation Strategy

At any moment, multiple tasks compete for a number of available resources and, conversely, multiple resources may be available for a single task. The resource allocation strategy is responsible for assigning each task ready to be executed to a single available resource. A strategy is said to be randomized if at a given configuration of the system, the strategy chooses the resource-task assignment according to some probability distribution. A strategy is defined to be deterministic if, given the same configuration of the system, the strategy always picks the same resource allocation strategy for tasks.

In most resource allocation strategies for operational processes, a set of best practices are followed based on heuristics. Thus, it is desirable to provide maximum expressiveness to the modeling formalism to allow industry best practices [NAR05, HRVZ, RM05]. For example, resources are often classified as *generalist* and *specialist* for certain set of tasks. A generalist is a resource suitable for performing a greater number of tasks, whereas a specialist performs a smaller number of tasks efficiently (in terms of time and/or cost). A best practice suggests *flexible assignment policy*, where at any moment, the task ready to be executed is provided with the most specialist

(available) resource so that maximum flexibility is preserved for near future. Such a policy is deterministic in nature.

In the running example, we assume that a priority-based scheduler is available. This scheduler assumes that there is an ordering among incoming cases – one can think of it as different tiers (platinum/gold/silver) of customers. We also assume that given a case, there is an ordering among its tasks. Hence, there is a total ordering among tasks across all cases. For simplicity, we also assume a total order among the resources. Hence at any configuration of the system, the scheduler goes through the resources in ascending order. For each resource, if a set of assigned tasks are available for that particular resource, it schedules the lowest ranked task among them. We note that such a scheduling policy may not be optimal, but only serves to explain our approach and demonstrate some experimental results.

5.2 Properties of Interest

On top of analyzing qualitative properties such as soundness [vdAvHtH$^+$11], operational processes also demand rigorous performance analysis. It is desirable to measure how key performance indicators (KPIs) relate among each other in resource-constrained processes. A number of quantitative aspects such as probability, cost and time come into play – on top of the concurrency and nondeterminism that are present in the system. Given a RCP and a pre-defined scheduler, the goal is to understand the relationship among these quantitative aspects and how they behave over scalability.

A fixed number of cases of a process model arrive following an arrival process. Each resource performing a task has its own fixed cost and time. We analyze linear temporal properties such as: when C cases arrive at a rate of

λ cases per second, with probability at least p, at least x fraction of cases are completed within time t. We would like to examine the relationship among these parameters C, λ, p, x and t – depending on the KPI of interest. Note that we can also reach closer to the optimal parameter values using binary search. If we fix a property and all parameters in that property except one, we can then run simulations for each value of that parameter. The limits of the value for which the property is satisfied can be found by running set of simulations for values chosen by a binary search. For example, if we are interested in optimizing x, we fix the other values p, t, λ and C and do a binary search for the optimal value of x in the range $[0, 1]$.

The state-of-the-art industry practice for quantitative analysis operational processes is to simulate the system [NAR05, OLRR12, HRVZ]. However, rigorous statistical analysis has often been limited to computing analysis of variance and confidence intervals with respect to arbitrary number of simulations. It is often not easy to justify the correctness of the analysis and the relationship between the number of simulations and the result is often not well-established. We use statistical model checking techniques and pose the property of interest as a hypothesis testing question. We then apply sequential probability ratio test, which formally connects the sample size to the desired margin for error. The conclusions we draw from our experiments come with guaranteed bounds on the probability of error in terms of false positives and false negatives. For a detailed discussion on statistical model checking, we refer to Chapter 2. We now describe the modeling formalism and the simulation analysis strategy for operational processes.

5.3 Timed Distributed Stochastic Model (tDSM)

We introduce Timed Distributed Stochastic Model (tDSM), an extension of DSM introduced in Chapter 3 with real-time associated with each event. These timestamps are real-valued and fixed-duration, representing the amount of time the event takes to complete.

Definition 17 (Timed DSM). *Let $\mathcal{D} = (\mathcal{S}, \Sigma, \chi, A)$ be a DSM. We define a* Timed DSM (tDSM) *to be $\mathcal{H} = (\mathcal{D}, \delta)$ where $\delta : \Sigma \to \mathbb{R}_{\geq 0}$ is the* duration function *over the set of events Σ.*

5.3.1 Snapshots and Schedulers

A tDSM $\mathcal{H}$ evolves as follows. All agents start at their initial state, so the initial global state at time 0 is $\mathbf{s}^{in} = (s_1^{in}, s_2^{in}, \ldots, s_n^{in})$. Suppose $\mathcal{H}$ is at a global state $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ at time t. A set of schedulable actions $U \in sch(\mathbf{s})$ is chosen from the set of enabled actions. We assume the existence of a scheduler that guides this choice. We can assume the set U to be non-empty, since not selecting any action is same as keep the system moving. The actions in U start concurrently and independently. For each action $a \in U$, an event $e_a = (src_e, tgt_e) \in E_a$ is chosen according to the probability distribution π_a with an associated duration $\delta(e_a)$ and cost $\chi(e_a)$. The durations $\{\delta(e_a)\}_{a \in U}$ fix a sequentialization of the events $\{e_a\}_{a \in U}$. In general, there will also be a pending list of partially executed events currently in progress, with their own completion times. Among the list of pending events, old and new, the events with the earliest time to completion finish first. For each completed event e, the local states of agents in $loc(e)$ are changed to tgt_e, while the states of the agents $[n] \setminus loc(e)$ are unchanged. This gives rise to a new global state

where potentially a new set of actions are scheduled, and we repeat the process of choosing a set of actions to schedule. We have to ensure that the scheduler respects the decisions made earlier, so that all pending events remain compatible with the new choice. Thus, the global configuration of a tDSM is not solely captured using the current global state. We denote a global configuration of a tDSM as a *snapshot*, consisting of a global state and a list of partially executed events, with their time to completion from the current time point.

Definition 18 (Snapshots of a tDSM)**.** *A* snapshot *of an tDSM $\mathcal{H}$ is a tuple $(\mathbf{s}, U, X)$ where $U \subseteq en(\mathbf{s})$ and $X = \{(a, e, t) \mid a \in U, e \in E_a, t \in \mathbb{R}_{\geq 0}\}$ such that:*

- $\mathbf{s}$ *is the current global state,*
- $U \in en(\mathbf{s})$ *is the set of actions that are currently being performed, which may not have started together,*
- *For each $a \in U$, there is exactly one entry $(a, e, t) \in X$ denoting that event $e \in E_a$ is in progress with time $t \leq \delta(e)$ till completion.*

Let $\mathcal{Y}$ be the set of snapshots. Though $\mathcal{Y}$ is uncountable, a tDSM will give rise to a discrete set of reachable snapshots, determined by the duration function δ.

5.3.2 Defining Schedulers

Nondeterministic choices between actions are resolved by a scheduler. We note that the snapshots in a tDSM can encode, using finite memory, the current state of the system as well as the events that are currently in progress.

At each snapshot, the scheduler can pick the next schedulable set of actions which, for consistency, should include all the actions already in progress but not necessarily new ones. We assume the scheduler to be randomized in general, i.e. it chooses one such schedulable set of actions according to a probability distribution associated with the snapshot. A deterministic scheduler can easily be emulated with the randomized scheduler by fixing all the probability distributions associated with the snapshots to have probability 1 for the next schedulable set of actions, and 0 otherwise.

Definition 19 (Schedulers)**.** *A* scheduler $\mathcal{G}$ *is defined over snapshots as follows. Let* $y = (\mathbf{s}, U, X) \in \mathcal{Y}$ *be a snapshot. Let the next possible set of schedulable actions at* y *is defined as* $next(y) = \{W_y \in sch(\mathbf{s}) \mid U \subseteq W_y \subseteq en(\mathbf{s})\}$*. Then, the scheduler is* $\mathcal{G} : \mathcal{Y} \to \{\Phi_\cdot : (2^A \to [0,1])\}$ *such that for all* $y \in \mathcal{Y}$*,* $\mathcal{G}(y)$ *is a probability distribution* $\Phi_y : next(y) \to [0,1]$*.*

We also note that, in general, it is hard to define independence-respecting schedulers for distributed systems. This is closely related to defining local winning strategies in distributed games [GLZ04]. The main complication is that a sequentially defined scheduler must behave consistently across different linearizations that correspond to the same concurrent execution. However, in a tDSM, the durations associated with the events fix a canonical linearization, so there is no need to reconcile decisions of the scheduler across different interleavings.

5.3.3 The Dynamics

Once we fix a scheduler, we can associate a transition system associated with a tDSM whose states are snapshots. The initial snapshot consists of the initial global state and no actions being performed. At any global snapshot,

the scheduler probabilistically chooses the next schedulable set of actions, which includes the actions associated with events that are that are already being performed. An event is chosen from each of the newly chosen actions, if any, according to the probability distribution associated with the corresponding action. Note that these new events have time to completion same as their duration. Among all the events—old and new—we pick the subset with minimal time to completion. These are the events that will complete before the rest. For each of these events, we then change the local states of the participating agents. Finally, we update the snapshot by removing the completed events and updating the time to completion for the rest of the events.

Definition 20 (Transition System of a tDSM)**.** *Given a tDSM $\mathcal{H}$ and a scheduler $\mathcal{G}$, we construct the transition system $TS = (\mathcal{S}, y^{in}, \rightarrow)$ as follows. Let $\mathcal{S} \subseteq \mathcal{Y}$ be a set of snapshots, with the initial snapshot is given by $y^{in} = (\mathbf{s}^{in}, \varnothing, \varnothing) \in \mathcal{S}$. Given a snapshot $y = (\mathbf{s}, U, X) \in \mathcal{S}$, where $U = \{a_1, a_2, \ldots, a_m\}$ and $X = \{(a_1, e_1, t_1), \ldots, (a_m, e_m, t_m)\}$, we define the next snapshots from y as follows.*

- *Let W_y be the set of actions scheduled according to the probability distribution Φ_y, i.e. $\Phi_y(W_y) > 0$. Recall that $U \subseteq W_y$. Let $V = W_y \setminus U = \{b_1, b_2, \ldots, b_k\}$.*

- *For each action $b = (E_b, \pi_b) \in V$, we pick an event $e_b \in E_b$ according to π_b, with duration $\delta(e_b)$ and cost $\chi(e_b)$. This generates a list $X_{new} = \{(b, e_b, \delta(e_b)) \mid b \in V, e_b \in E_b\}$. Note that all the events in $X \cup X_{new}$ have pairwise disjoint locations.*

- *From $X \cup X_{new}$, let $t_{min} = min(\{t \mid (a, e, t) \in X \cup X_{new}\})$ be the minimum across all durations left. We pick the subset $E_{\min} = \{(a, e, t) \in X \cup X_{new} \mid$*

$t = t_{\min}\}$. *We then update the snapshot as follows:*

1. *For each* $(a, e, t_{\min})$ *in* $E_{\min}$, *set* $\mathbf{s}_{loc(e)}$ *to* tgt_e, *and remove a from* $U \cup V$ *and* $(a, e, t_{\min})$ *from* $X \cup X_{new}$.
2. *For each* (a, e, t') *in* $X \cup X_{new}$, *update* t' *to* $t' - t_{\min}$, *thus reducing the time to completion of e by* $t_{\min}$.

This results in a new snapshot $(\mathbf{s}', U', X')$, *where* $U' = U \cup V$ *and* $X' = X \cup X_{new}$.

We denote the transition $y \xrightarrow{X_{new}, E_{\min}, prob(y,y'), time(y,y'), cost(y,y')} y'$, defined as follows.

- X_{new} encodes the events that are newly chosen at y by the scheduler,
- $E_{\min}$ encodes the events that has completed in this step,
- The probability associated with the transition is $prob(y, y') = \Phi_y(W_y) \cdot \prod_{(b,e_b,\delta(e_b))\in X_{new}} \pi_b(e_b)$. If $X_{new} = \varnothing$, $p = \Phi_y(W_y)$.
- The time duration associated with the transition is $time(y, y') = t_{\min}$ attached to each $(a, e, t_{\min}) \in E_{\min}$.
- The cost associated with the transition is $cost(y, y')$, the sum of $\chi(e)$, for all $(a, e, t_{\min}) \in E_{\min}$.

Theorem 2. *The transition system* $TS = (\mathcal{S}, y^{in}, \rightarrow)$ *associated with a tDSM* $\mathcal{D}$ *along with* δ *and a scheduler* $\mathcal{G}$ *is a Markov chain.*

Proof. Let $y = (\mathbf{s}, U, X)$ be a snapshot in the transition system TS and N_y be the set of next snapshots. We claim that $\sum_{y'\in N_y} prob(y, y') = 1$. We know

that the next snapshots can be grouped according to the next schedulable actions (probabilistically) chosen by the scheduler. Hence,

$$\sum_{y' \in N_y} prob(y, y') = \sum_{W_y \in next(y)} \left(\Phi_y(W_y) \cdot \sum_{\substack{X_{new} \text{ generated} \\ \text{from } W_y}} \prod_{(b, e_b, \delta(e_b)) \in X_{new}} \pi_b(e_b) \right)$$

Suppose $V = \{b_1, b_2, \ldots, b_k\}$ is the new set of actions chosen by the scheduler at a snapshot y. As observed earlier, each combination of events $\{e_1, e_2, \ldots, e_k\}$ generated by applying π_{b_i} to E_{b_i}, $i \in \{1, 2, \ldots, k\}$, results in a unique next snapshot. Hence fixing V, the following is true:

$$\sum_{\substack{X_{new} \text{ generated} \\ \text{from } W_y}} \prod_{(b, e_b, \delta(e_b)) \in X_{new}} \pi_b(e_b) = 1.$$

If $V = \varnothing$, it trivially applies. Morever, since Φ_y is a probability distribution over $next(y)$, $\sum_{W_y \in next(y)} \Phi_y(W_y) = 1$, proving the required result. □

5.3.4 Simulation Techniques for tDSM

We introduce a Statistical Model Checking (SMC) technique for tDSM. The algorithm is illustrated in Algorithm 1. We intend to analyze a property bounded by probability, time, and cost of the form $Pr_{\geq \theta} T_{\leq t} C_{\leq c} \phi$, a shorthand of "with probability at least θ, property ϕ is satisfied within time t and cost c". In this discussion, we assume that ϕ is a reachability property specified in linear time temporal logic.

We are provided with a tDSM $\mathcal{H}$ and a scheduler $\mathcal{G}$, along with a property $Pr_{\geq \theta} T_{\leq t} C_{\leq c} \phi$. Since we are interested in simulation method with prov-

Algorithm 1 Statistical Model Checking for tDSM

INPUT:

a tDSM $\mathcal{H}$ and a scheduler $\mathcal{G}$
a property $Pr_{\geq\theta}T_{\leq t}C_{\leq c}\phi$
error bounds: α (Type-I) and β (Type-II), threshold of indifference: δ

OUTPUT:

YES or NO

procedure SIMULATE-TDSM
  $m \leftarrow 0$ ▷ the number of simulations so far
  $\theta^+ \leftarrow \theta + \delta$ and $\theta^- \leftarrow \theta - \delta$ ▷ indifference region
  while True **do**
    $t_{spent} \leftarrow 0$ ▷ time spent so far
    $c_{spent} \leftarrow 0$ ▷ cost accumulated so far
    $y^{in} = (\mathbf{s}^{in}, \varnothing, \varnothing)$ ▷ the initial snapshot
    $y = (\mathbf{s}, U, X) := y^{in}$ ▷ the current snapshot
    $\rho := y$ ▷ the current execution
    $b \leftarrow 0$ ▷ the outcome of the Bernoulli random variable
    $d_m \leftarrow 0$ ▷ accumulator of outcome of the Bernoulli variable
    while $(t_{spent} \leq t)$ and $(c_{spent} \leq c)$ **do**
      $W_y \leftarrow$ *scheduled actions according to* $\mathcal{G}(y)$
      $V \leftarrow W_y \setminus U$
      $X_{new} \leftarrow \{(b, e_b, \delta(e_b)) \mid b \in V, e_b \in E_b\}$
      $t_{min} \leftarrow min(\{t \mid (a, e, t) \in X \cup X_{new}\})$
      $E_{\min} = \{(a, e, t) \in X \cup X_{new} \mid t = t_{\min}\}$
      for all $(a, e, t_{\min}) \in E_{\min}$ **do**
        $\mathbf{s}_{loc(e)} \leftarrow tgt_e$
        $U \cup V \leftarrow (U \cup V) \setminus \{a\}$
        $X \cup X_{new} \leftarrow (X \cup X_{new}) \setminus \{(a, e, t_{min})\}$
      for all $(a, e, t') \in X \cup X_{new}$ **do**
        $t' \leftarrow t' - t_{\min}$ ▷ update time bound
      $y = (\mathbf{s}', U \cup V, X \cup X_{new})$ ▷ the new snapshot
      $t_{spent} \leftarrow t_{spent} + t_{\min}, c_{spent} \leftarrow c_{spent} + \sum_{(a,e,t)\in E_{\min}} \chi(e), \rho \leftarrow \rho y$
      if ρ satisfies ϕ **then**
        $b = 1$
        break
    $m \leftarrow m + 1$ and $d_m \leftarrow d_m + b$
    $quo = \frac{(\theta^-)^{d_m}(1-\theta^-)^{m-d_m}}{(\theta^+)^{d_m}(1-\theta^+)^{m-d_m}}$
    if $(quo \geq \frac{(1-\beta)}{\alpha})$ **then** ▷ property is not satisfied
      return NO
    else if $(quo \leq \frac{\beta}{1-\alpha})$ **then** ▷ property is satisfied
      return YES

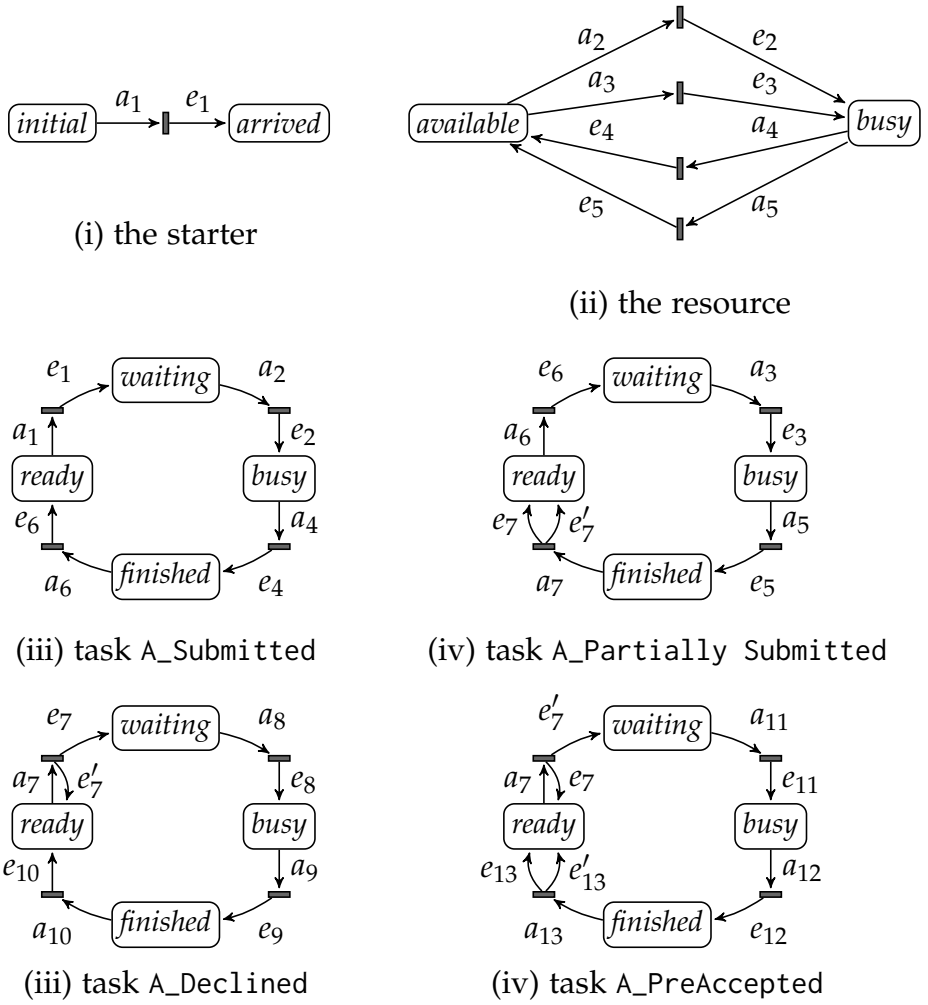
able error bound, we also take the bound of Type-I error (false positive) α, Type-II error (false negative) β, and threshold of indifference δ as inputs. The output would be a binary *yes* or *no*, indicating if the tDSM along with the scheduler satisfies the given property. The correctness of our result is quantified with the error bounds.

We keep simulating the system and collecting samples, keeping track of the number of simulations so far. To generate one sample of the system, we keep running the system generating new snapshots, keeping track of the quantitative bounds. While the time and cost spent so far is less than the time and cost bound provided with the property, we run the system unless the run satisfies the property in hand. Since we have established in Theorem 2 that the transition system of the tDSM is a Markov chain, the set of runs has a well-defined probability measure. Following standard SMC technique, we then guarantee that the simulation will end with probability 1 and we can report if the property is satisfied or not with error of report bounded by the provided error bounds.

5.4 Modeling Resource-constrained Processes

We model resource-constrained processes as tDSM. All tasks across cases and resources are individual tasks that interact with each other and then move probabilistically. The resource-allocation strategy can be interpreted as the scheduler in a tDSM.

Each task is an individual agent incorporating the states of a task, such as 'ready to perform', 'waiting for a resource', 'busy executing', 'finished' etc. When a task finishes, it triggers other task agents in the control flow that are ready to perform. Each resource is also a simple agent, looping



(i) the starter

(ii) the resource

(iii) task `A_Submitted`

(iv) task `A_Partially Submitted`

(iii) task `A_Declined`

(iv) task `A_PreAccepted`

Figure 5.2: Modeling (part of) the running RCP system example as tDSM agents. The probability values and the duration attached to the events are not depicted.

between being 'available' and 'busy'. The scheduler maps each task waiting for a resource to an available resource. This results in a synchronization that generates an event whose duration captures the time taken to perform the task. We also model the start and end states of a process as agents, to model the arrival and completion of a case.

We recall that an RCP system consists of a business process instantiated as a number of cases and a finite set of resources. Let us assume that in an

RCP system $\mathcal{B}$, there are C cases numbered $\{1, 2, \ldots, C\}$ each with k_T tasks labelled as follows: T_{ij} denotes the j^{th} task of the i^{th} case, for $1 \leq i \leq C$ and $1 \leq j \leq k_T$. We assume the rate of the arrival process is λ cases per second. Let us assume there are r resources denoted $\{R_1, R_2, \ldots, R_r\}$. In the running example of loan/overdraft application, $k_T = 15$ and $r = 46$ (see Figure 5.1). For each resource R_i, let $tasks(R_i) \in \{T_{ij} \mid 1 \leq i \leq C, 1 \leq j \leq k_T\}$ denote the set of tasks resource R_i is able to perform. Since resources can be shared among tasks, for any $1 \leq i, j \leq r$, $tasks(R_i) \cap tasks(R_j)$ can possibly be non-empty.

Given an RCP system, we transform it to a tDSM as follows. We model tasks and resources as agents. To facilitate the arrival process and clearly mark the case completion, we also model the start and end states as agents. Hence, the RCP system $\mathcal{B}$ can be modeled using $r + C \times (k_T + 2)$ agents.

Each task agent consists of 4 states (i) *ready* to perform, (ii) *waiting* for a resource, (iii) *busy* being executed, and (iv) *finished*. Each resource agent consists of 2 states (i) *available* and (ii) *busy*. An agent modeling the start state is called a starter agent, and has two states *waiting* and *arrived*. Similarly, the agent modeling the end state is named finisher agent with states *pending* and *done*.

We illustrate a part of tDSM modeling the running RCP example in Figure 5.2. For brevity, we did not illustrate the rest of the tDSM corresponding to the loan/draft application, but it can be easily extended. We show the agents corresponding to the start state and 4 tasks: `A_Submitted`, `A_Partially Submitted`, `A_Declined` and `A_PreAccepted`. We also demonstrate a resource that can perform tasks `A_Submitted` and `A_Partially Submitted`. The states are depicted in rounded cornered rectangles and the edges be-

tween the states are defined as follows: $s \xrightarrow{\text{a} \quad \text{e}} s'$ denotes the action at local state s, $e \in E_a$ and s' be the next local state after event e. We note that each event e also has a probability value and a duration attached to it, but for simplicity in showcasing the example, we only included the event names in the diagram. We would also like to point out that the initial states are not marked, but one can freely assume any starting state as the modeling seems fit.

The starter agent mimics the wait for a case. At state *initial*, it synchronizes with the starting task agent at *ready* state followed by an event with probability 1. The duration of the event is equal to the total time before arrival for the particular case. The starter then moves to *arrived* state and the starting task moves to *waiting* state, where it waits for a resource to be scheduled. For example, Figure 5.2(i) shows the action a_1 between the starter agent and the task `A_Submitted` followed by event e_1 such that $\pi_a(e_a) = 1$ and $\delta(e_1) = 1/\lambda$.

Once a task is in *waiting* state and the scheduler assigns a resource that is in *available* state, the task and the resource synchronize and they both move to *busy* state with probability 1. In Figure 5.2, examples of such actions are a_2 and a_3. There they synchronizes again, performs an event with possibly non-zero time duration, and the task moves to *finished* with probability 1. In Figure 5.2, examples of such actions are a_4 and a_5.

Once a task is finished, it signals the next task(s) in the control flow via synchronization. For example, in Figure 5.2, after task `A_Partially Submitted` is at *finished*, it sychronizes with tasks `A_Declined` and `A_Preaccepted` via action a_7 such that $E_{a_7} = \{e_7, e_7'\}$ with $\pi_{a_7}(e_7) = 0.84$ and $\pi_{a_7}(e_7') = 0.16$. Hence, with probability 0.84, event e_7 is chosen and only task `A_Declined`

moves to *waiting* state. Otherwise, with probability 0.16, event e'_7 is chosen and only task `A_Preaccepted` moves to *waiting* state. In both cases, `A_Partially Submitted` moves to *ready* state. A task is ready again after finishing since due to loops in control flow, a task may be executed multiple times in the same case.

When a case finishes, the last task in *finished* state synchronizes with the finisher agent in *pending* state. The task then moves to *ready* as usual and the finisher agent moves to *done*, indicating the completion of the corresponding task. The finisher agent then stays at the state *done* with probability 1.

5.4.1 Modeling Flexibility and Predictive Analysis

We would like to point out that the described methodology to model RCP systems as tDSM is only one example among many possibilities. One may easily extend this approach and incorporate an even more complex state space for tasks or resources modeling different scenarios. For example, we can easily model probabilistic error in task execution. Let us assume that when the resource depicted in Figure 5.2 performs task `A_Submitted`, there is a small probability of 0.1 that such an execution fails. This phenomenon can be easily modeled by adding another event e to E_{a_2} such that $\pi_{a_2}(e_2) = 0.9$ and $\pi_{a_2}(e) = 0.1$ where after e, they move back to *available* and *waiting* respectively. Also, for simplicity, we assumed that the business processes under consideration are sound, but tDSM can be easily used to model unsound RCP systems as well.

This methodology also supports verifying a variety of properties, depending on the focus of optimization for the business. For example, one may investigate the performance of resources by verifying the following

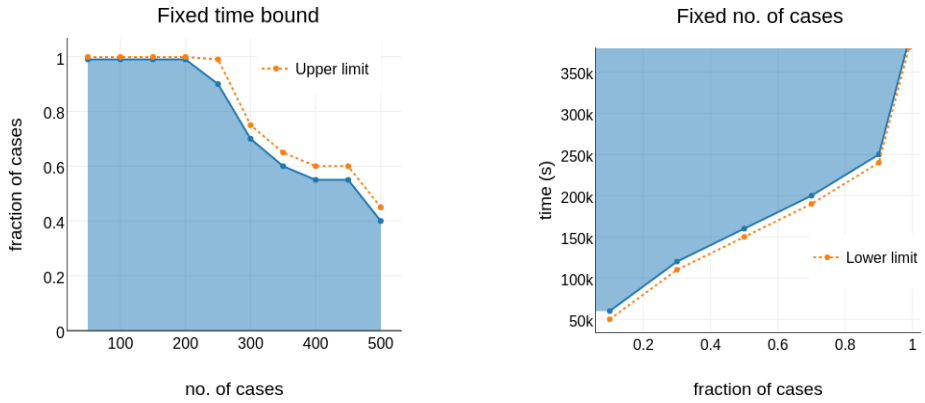


Figure 5.3: (left) Fraction of cases completed vs total no. of cases when the time bound is fixed, (right) Minimum total time vs fraction of cases completed when the total number of cases is fixed.

property: given a resource, in what fraction of cases was it used?

Our analysis techniques also open the door for strong predictive analysis for businesses. Instead of looking at a business in terms of a set of historical data, we propose to model business operations as a living and breathing process. Such a process is also built on strong modeling formalism, which allows us to tweak the model parameters and provide meaningful predictions over the relationship among KPIs. Armed with such predictive analysis, a business can not only understand their current operations, but also improve on the desired KPIs by focusing on the correct parts of the model.

5.5 Experimental Evaluation

We have tested our SMC procedure on the running example. The property we are interested in is follows:

With probability at least 0.99, *when cases arrive at a fixed rate of* 1 *case per* 10 *seconds, x fraction of cases are completed among C cases within t seconds.*

We ensure that the probability of Type-I error and Type-II error of verifying this property is less than 0.01 with indifference region (0.99 ± 0.005).

We first investigate the relationship between the number of total cases and the fraction of cases completed within fixed maximum time $t = 500,000$ seconds. The shaded area in Figure 5.3 (left) represents the values (C, x) that satisfies the property when $t = 500,000$ seconds. The dotted line represents an upper limit for the values (C, x) satisfying the property. As expected, as more cases arrive within the same time bound, the fraction of cases that are completed goes down.

Then, we illustrate the relationship between the minimum total time and the fraction of cases completed within that total time when the total number of cases $C = 100$ is fixed. The shaded area in Figure 5.3 (right) represents the values (x, t) that satisfies the property when $C = 100$. The dotted line represents a lower limit for the values (x, t) satisfying the property. Again as expected, the more time is allotted, the more fraction of cases gets completed given the total number of cases are fixed.

We show the relationship between multiple quantitative variables in these two graphs. The goal is not to calculate traditional KPIs such as throughput directly for a specific set of parameters, but rather shed light on the relations among the parameters using simulation techniques with provable error bounds. The stress test for the performance of the system can be viewed in terms of a number of parameters such as the number of cases, the total time budget and many more (such as cost, which is not detailed in this case study).

We note that the experimental results presented is a succinct representation of many simulation results. Each data point in the border of the covered

area represent a simulation with a particular set of quantitative data points. We have also performed binary search over parameters to find a upper (or, lower) limit for the data points. Understandably, the limits are not the tightest, but can be made arbitrary closer to the real limit with more simulations and binary search in the space between the limit the covered area. Though our experiment is only a proof-of-concept, we would like point out that we can scale the size of the model comfortably to accommodate 500 cases. The experiments were run in Python on a standard core-i5 processor laptop with 12 GB of RAM.

Chapter 6

The Conclusion and Future Work

The goal of this thesis was to provide a solid framework for modeling distributed probabilistic systems with an application on quantitative analysis. We presented Distributed Stochastic Model (DSM), extending along the line of asynchronous transition systems. In its most general form, the model can be incorporated with both probability and nondeterminism in a distinct fashion. One can also attach non-negative real-valued cost and time along with the atomic events.

We then discussed the behavior of the model when nondeterminism is restricted, which turns the model into a Markov chain. We have also defined both interleaved and non-interleaved semantics on this restricted version of the model, which paves the way for clean simulation-based model checking procedures. Two case studies of distributed probabilistic algorithms from the PRISM benchmark [HKNP06] have been used for experiments and comparison with the performance of the statistical model checking tool PLASMA [BCLS13].

In the next part, we defined resource-constrained operational processes, a demanding area of research under business process management. The

challenge was to improve the state-of-the-art simulation techniques that does not provide any formal guarantee on the outcome of the analysis. We first defined resource-constrained processes, with the help of an example from a Dutch bank. We then developed timed distributed stochastic model, a variation on top of the framework, where atomic events are equipped with fixed-duration time. This allows us to define a finite-memory scheduler that respects concurrency, and under such a scheduler, a countably-infinite state Markov chain can be associated with the model.

We then use this approach to model resource-constrained processes and showed how a statistical model checking technique can be developed. This provides a better alternative to the state-of-the-art by ensuring provable error bounds in terms of false negative and false positive rates for the result of the simulation. The proof-of-concept is presented with the process from a Dutch bank and we have provided predictive analysis on relating some of the important business KPIs.

We believe that the goal of the thesis—to establish a framework to transcend beyond correctness and use simulation-based model checking techniques for performance evaluation of real-life systems [BHHK10, AHV15]—was reasonably met. We have applied our framework to a novel application area improving upon the state-of-the-art techniques. Nonetheless, there is a significant scope of future work that can be built on our framework.

The framework can be restricted to only consider goal-oriented properties. For example, similar to the Probabilistic Workflow Nets [EHS16], one can define a DSM such that each agent has a unique final state and there is a final action indicating that the global system continues to remain in the final state. With suitable restrictions on the model, one may be able to

compute the expected reward in such a model.

There is an interesting line of work on statistical model checking for unbounded properties for Markov chains [DHKP16]. The key idea is to detect strongly connected components in the Markov chain with statistical guarantees. One may explore how the Distributed Markov Chain (DMC) model presented in this thesis may take advantage of the concurrency that is often present in a Markov chain to improve the complexity of finding strongly connected components. The well-defined interleaved semantics associated with the model can aid to find the components locally in term of agents.

The main partial order concept we have used is to group trajectories into equivalence classes. One can also explore how ample sets [JGP99] and related notions can be used to model check properties specified in logics such as PCTL [BK08]. Another possibility is to see if the notion of finite unfoldings from Petri net theory can be applied in the setting of DMCs [EH08, MP95].

In our two case studies, the specification has a global character in that it mentions every agent in the system. In many specifications, only a few agents will be mentioned. If the system is loosely coupled, we can check whether the required property is fulfilled without having to exercise all the agents. This will lead to additional computational gains.

In many of the benchmark examples in [HKNP06], the probabilistic moves are local. On the other hand, DMCs allow synchronous probabilistic moves where the probability distribution is influenced by information obtained through communication. It will be interesting to exploit this feature to model and analyze applications arising in embedded control systems.

We currently allow agents to gain complete information about the state of the agents they synchronize with. In practice, only a part of this state may be exposed.

One may also extend the Timed Distributed Stochastic Model (tDSM) to shed light on different types of scheduling policies and their impact on business processes. We would also like to incorporate stochastic durations for events, which will take us to a Continuous Time Markov Chain (CTMC) setting. Finally, one may explore how approximate verification techniques can also enrich process mining of business processes [Aal11].

Bibliography

[Aal97] W. M. P. Aalst. Verification of workflow nets. In Pierre Azéma and Gianfranco Balbo, editors, *Application and Theory of Petri Nets 1997*, number 1248 in Lecture Notes in Computer Science, pages 407–426. Springer Berlin Heidelberg, 1997.

[Aal11] W. M. P. Aalst. *Process mining: discovery, conformance and enhancement of business processes*. Springer Science & Business Media, 2011.

[Aal13] W. M. P. Aalst. Business process management: a comprehensive survey. ISRN Softw. Eng. 1-37. *ISRN Software Engineering*, (1), 2013.

[Aal14] W. M. P. Aalst. Business process management as the "Killer App" for Petri nets. *Software & Systems Modeling*, 14(2):685–691, June 2014.

[Aal15] W. M. P. Aalst. *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems*, chapter Business Process Simulation Survival Guide, pages 337–370. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[AB06] Samy Abbes and Albert Benveniste. True-concurrency probabilistic models: Branching cells and distributed probabilities for event structures. *Info. and Comp.*, 204(2):231–274, 2006.

[AB08] Samy Abbes and Albert Benveniste. True-concurrency probabilistic models: Markov nets and a law of large numbers. *Theor. Comput. Sci.*, 390(2-3):129–170, 2008.

[AH04] W. M. P. Aalst and Kees Max van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2004.

[AHK+02] Wil Van Der Aalst, Kees Van Hee, Prof. Dr. Kees, Max Hee, Remmert Remmerts De Vries, Jaap Rigter, Eric Verbeek, and Marc Voorhoeve. Workflow management: Models, methods, and systems, 2002.

[AHV15] Rajeev Alur, Thomas A Henzinger, and Moshe Y Vardi. Theory in practice for system design and verification. *ACM Siglog News*, 2(1):46–51, 2015.

[BA13] R. P. Jagadeesh Chandra Bose and W. M. P. Aalst. *Business Process Management Workshops: BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers*, pages 221–222. Springer Berlin Heidelberg, 2013.

[BCHG+97] Christel Baier, Edmund M. Clarke, Vasiliki Hartonas-Garmhausen, Marta Kwiatkowska, and Mark Ryan. *Symbolic model checking for probabilistic processes*, pages 430–440. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[BCLS13] Benoît Boyer, Kevin Corre, Axel Legay, and Sean Sedwards. Plasma-lab: A flexible, distributable statistical model checking library. In *Proc. QEST 2013*, volume 8054 of *Lecture Notes in Computer Science*, pages 160–164, 2013.

[Bed87] Marek Antoni Bednarczyk. *Categories of Asynchronous Systems*. PhD thesis, Sussex, UK, UK, 1987. AAIDX83002.

[BFHH11] Jonathan Bogdoll, Luis Maria Ferrer Fioriti, Arnd Hartmanns, and Holger Hermanns. Partial order methods for statistical model checking and simulation. In *Formal Techniques for Distributed Systems*, Lecture Notes in Computer Science, pages 59–74. 2011.

[BFV12] Kelly R Braghetto, João E Ferreira, and Jean-Marc Vincent. Performance evaluation of resource-aware business processes using stochastic automata networks. *International Journal of Innovative Computing, Information and Control*, 8(7B):5295–5316, 2012.

[BHHK10] Christel Baier, Boudewijn R Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Performance evaluation and model checking join forces. *Communications of the ACM*, 53(9):76–85, 2010.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, June 1984.

[BJK+05] Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner. *Model-based testing of reactive systems: advanced lectures*, volume 3472. Springer, 2005.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[BL03] Benedikt Bollig and Martin Leucker. *Model Checking Probabilistic Distributed Systems*, pages 291–304. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[Box79] George EP Box. Robustness in the strategy of scientific model building. *Robustness in statistics*, 1:201–236, 1979.

[CDL10] Edmund Clarke, Alexandre Donzé, and Axel Legay. On simulation-based probabilistic model checking of mixed-analog circuits. *Formal Methods in System Design*, 36(2):97–113, 2010.

[CFL+08] Edmund M. Clarke, James R. Faeder, Christopher J. Langmead, Leonard A. Harris, Sumit Kumar Jha, and Axel Legay. *Statistical Model Checking in BioLab: Applications to the Automated Analysis of T-Cell Receptor Signaling Pathway*, pages 231–250. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[CGP99] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[Clao8] Edmund M Clarke. The birth of model checking. In *25 Years of Model Checking*, pages 1–26. Springer, 2008.

[CYFMo2] L. I. N. Chuang, Q. U. Yang, R. E. N. Fengyuan, and Dan C. Marinescu. Performance Equivalent Analysis of Workflow Systems Based on Stochastic Petri Net Models. In *Engineering and Deployment of Cooperative Information Systems*, number 2480 in LNCS, pages 64–79. Springer Berlin Heidelberg, 2002.

[CZ11] Edmund M. Clarke and Paolo Zuliani. *Statistical Model Checking for Cyber-Physical Systems*, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[DE95] Jörg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press, New York, NY, USA, 1995.

[DFZoo] Juliane Dehnert, JÃűrn Freiheit, and Armin Zimmermann. Modeling and performance evaluation of workflow systems, 2000.

[DHKP16] Przemysław Daca, Thomas A. Henzinger, Jan Křetínský, and Tatjana Petrov. *Faster Statistical Model Checking for Unbounded Temporal Properties*, pages 112–129. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[DR95] V. Diekert and G. Rozenberg. The book of traces. World Scientific, 1995.

[EH08] J. Esparza and K. Heljanko. *Unfoldings: A Partial-Order Approach to Model Checking*. Springer Publishing Company, 1 edition, 2008.

[EHS16] Javier Esparza, Philipp Hoffmann, and Ratul Saha. Polynomial analysis algorithms for free choice probabilistic workflow nets. In *International Conference on Quantitative Evaluation of Systems*, pages 89–104. Springer International Publishing, 2016.

[Fel08] Willliam Feller. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, 2008.

[Fer94] A. Ferscha. Qualitative and quantitative analysis of business workflows using generalized stochastic petri nets. In *Proceedings of the Ninth Austrian-informatics Conference on Workflow Management : Challenges, Paradigms and Products: Challenges, Paradigms and Products*, CON '94, pages 222–234, Munich, Germany, Germany, 1994. R. Oldenbourg Verlag GmbH.

[Fok06] Wan Fokkink. Variations on Itai-Rodeh leader election for anonymous rings and their analysis in PRISM. *J. UCS*, 12, 2006.

[GB06] Marcus Groesser and Christel Baier. Partial order reduction for Markov decision processes: A survey. In *Formal Methods for Components and Objects*, pages 408–427. 2006.

[GLZ04] Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games and distributed control for asynchronous systems. In *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings*, pages 455–465, 2004.

[GV08] Orna Grumberg and Helmut Veith. *25 years of model checking: history, achievements, perspectives*, volume 5000. Springer, 2008.

[Her14] Luke Thomas Herbert. *Specification, Verification and Optimisation of Business Processes*. A Unified Framework. Technical University of Denmark, 2014.

[HKNP06] Andrew Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, number 3920 in Lecture Notes in Computer Science, pages 441–444. Springer Berlin Heidelberg, January 2006.

[HRVZ] Kees Van Hee, Hajo Reijers, Eric Verbeek, and Loucif Zerguini. On the optimal allocation of resources in stochastic workflow nets.

[IR90] Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Info. and Comp.*, 88(1):60–87, September 1990.

[JCL+09] Sumit K. Jha, Edmund M. Clarke, Christopher J. Langmead, Axel Legay, AndrÃl' Platzer, and Paolo Zuliani. A Bayesian approach to model checking biological systems. In *Computational Methods in Systems Biology*, pages 218–234. 2009.

[JGP99] Edmund M. Clarke Jr, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Mass, 1999.

[JPS96] S. Jesi, G. Pighizzini, and N. Sabadini. Probabilistic asynchronous automata. *Mathematical systems theory*, 29(1):5–31, 1996.

[Kat16] Joost-Pieter Katoen. The probabilistic model checking landscape. 2016.

[Kel76] Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, July 1976.

[LCJ11] Zhi Liu, Gene Cheung, and Yusheng Ji. Distributed markov decision process in cooperative peer-to-peer repair for wwan video broadcast. In *Proceedings of the 2011 IEEE International Conference on Multimedia and Expo*, ICME '11, pages 1–6, Washington, DC, USA, 2011. IEEE Computer Society.

[LDB10] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *Runtime Verification*, pages 122–135. Springer, 2010.

[LWC+02] Dongsheng Liu, Jianmin Wang, Stephen C.F. Chan, Jiaguang Sun, and Li Zhang. Modeling workflow processes

with colored petri nets. *Computers in Industry*, 49(3):267 – 281, 2002.

[Mil82] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[MM07] Matteo Magnani and Danilo Montesi. BPMN: How Much Does It Cost? An Incremental Approach. In *Business Process Management*, number 4714 in Lecture Notes in Computer Science, pages 80–87. Springer Berlin, September 2007.

[MP95] K. L. McMillan and D. K. Probst. A technique of state space search based on unfolding. *Form Method Syst Des*, 6(1):45–65, January 1995.

[NAR05] Mariska Netjes, W. M. P. Aalst, and Hajo A Reijers. Analysis of resource-constrained processes with colored petri nets. In *Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, volume 576, pages 251–266. series DAIMI, 2005.

[Nor98] James R Norris. *Markov chains*. Number 2. Cambridge university press, 1998.

[NPS13] Gethin Norman, David Parker, and Jeremy Sproston. Model checking for probabilistic timed automata. *Formal Methods in System Design*, 43(2):164–190, 2013.

[OLRR12] C.A.L. Oliveira, R.M.F. Lima, H.A. Reijers, and J.T.S. Ribeiro. Quantitative analysis of resource-constrained business pro-

cesses. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 42(3):669–684, May 2012.

[PGL+13] Sucheendra K. Palaniappan, Benjamin M. Gyori, Bing Liu, David Hsu, and P. S. Thiagarajan. *Statistical Model Checking Based Calibration and Analysis of Bio-pathway Models*, pages 120–134. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[PZ86] Amir Pnueli and Lenore D. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.

[Rei03] Hajo Reijers. *Design and Control of Workflow Processes: Business Process Management for the Service Industry*. Springer, July 2003.

[RHM06] Nick Russell, Arthur H. M. Ter Hofstede, and Nataliya Mulyar. Workflow ControlFlow Patterns: A Revised View. Technical report, 2006.

[RM05] H.A. Reijers and S. Liman Mansar. Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. *Omega*, 33(4):283 – 306, 2005.

[RT86] G. Rozenberg and P. S. Thiagarajan. *Petri nets: Basic notions, structure, behaviour*, pages 585–668. Springer Berlin Heidelberg, Berlin, Heidelberg, 1986.

[Seg95] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-time Systems*. PhD thesis, Cambridge, MA, USA, 1995. Not available from Univ. Microfilms Int.

[SEJ+15] Ratul Saha, Javier Esparza, Sumit Kumar Jha, Madhavan Mukund, and P. S. Thiagarajan. Distributed Markov chains. In *In Proceedings of 16th Verification, Model Checking, and Abstract Interpretation - 16th International Conference, (VMCAI 2015), Mumbai, India, January 12–14, 2015*, pages 117–134, 2015.

[Shi85] M. W. Shields. Concurrent machines. *The Computer Journal*, 28(5):449–465, 1985.

[SMB16] Ratul Saha, Madhavan Mukund, and R. P. Jagadeesh Chandra Bose. *Time-Bounded Statistical Analysis of Resource-Constrained Business Processes with Distributed Probabilistic Systems*, pages 297–314. Springer International Publishing, Cham, 2016.

[SR95] A. K. Schömig and H. Rau. A petri net approach for the performance analysis of business processes. Mai, 1995.

[SSLD12] Songzheng Song, Jun Sun, Yang Liu, and Jin Song Dong. *A Model Checker for Hierarchical Probabilistic Real-Time Systems*, pages 705–711. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[SW09] Partha Sampath and Martin Wirsing. Computing the Cost of Business Processes. In *Information Systems: Modeling, Development, and Integration*, number 20 in Lecture Notes in Business Information Processing, pages 178–183. Springer Berlin Heidelberg, April 2009.

[vD12] B.F. van Dongen. BPI challenge 2012, 2012.

[vdAvHtH+11] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.

[VVW04] Daniele Varacca, Hagen Völzer, and Glynn Winskel. Probabilistic event structures and domains. In *CONCUR 2004 - Concurrency Theory*, pages 481–496. 2004.

[Wal45] A. Wald. Sequential tests of statistical hypotheses. *Ann. Math. Statist.*, pages 117–186, 1945.

[Win89] Glynn Winskel. *An introduction to event structures*, pages 364–397. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989.

[WN95] Glynn Winskel and Mogens Nielsen. Handbook of logic in computer science (vol. 4). chapter Models for Concurrency, pages 1–148. Oxford University Press, Oxford, UK, 1995.

[You04] Håkan L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon University, Pittsburgh, USA, 2004.

[YS06] Håkan L. S. Younes and Reid G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.*, 204(9):1368–1409, September 2006.

[Zie87] Wieslaw Zielonka. Notes on finite asynchronous automata. *ITA*, 21(2):99–135, 1987.