Verification and Planning for Stochastic Processes with Asynchronous Events

Håkan Lorens Samir Younes

January 2005
CMU-CS-05-105

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Reid G. Simmons, Chair
Edmund M. Clarke
Geoffrey J. Gordon
Jeff G. Schneider
David J. Musliner, Honeywell Laboratories

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

This research was sponsored by the US Army Research Office (ARO) under contract no. DAAD190110485, the National Aeronautics & Space Administration (NASA) under grant nos. NAG9-1092, NAG2-1266 and NAG9-1248, and the Royal Swedish Academy of Engineering Sciences (IVA) with grants from the Hans Werthén fund. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the sponsoring institutions, the U.S. Government or any other entity.

Keywords: Model checking, probabilistic verification, decision theoretic planning, failure analysis, stochastic processes, Markov chains, hypothesis testing, acceptance sampling, discrete event simulation, phase-type distributions, sequential analysis, temporal logic, transient analysis

Abstract

Asynchronous stochastic systems are abundant in the real world. Examples include queuing systems, telephone exchanges, and computer networks. Yet, little attention has been given to such systems in the model checking and planning literature, at least not without making limiting and often unrealistic assumptions regarding the dynamics of the systems. The most common assumption is that of history-independence: the Markov assumption. In this thesis, we consider the problems of verification and planning for stochastic processes with asynchronous events, without relying on the Markov assumption. We establish the foundation for statistical probabilistic model checking, an approach to probabilistic model checking based on hypothesis testing and simulation. We demonstrate that this approach is competitive with state-of-the-art numerical solution methods for probabilistic model checking. While the verification result can be guaranteed only with some probability of error, we can set this error bound arbitrarily low (at the cost of efficiency). Our contribution in planning consists of a formalism, the generalized semi-Markov decision process (GSMDP), for planning with asynchronous stochastic events. We consider both goal directed and decision theoretic planning. In the former case, we rely on statistical model checking to verify plans, and use the simulation traces to guide plan repair. In the latter case, we present the use of phase-type distributions to approximate a GSMDP with a continuous-time MDP, which can then be solved using existing techniques. We demonstrate that the introduction of phases permits us to take history into account when making action choices, and this can result in policies of higher quality than we would get if we ignored history dependence.

For the one closest to my heart

Acknowledgments

First and foremost, I would like to thank Reid Simmons for his great support and timely advice during my tenure as a graduate student. I thank him for his patience during my first two years in the PhD program, when I was making little progress on exploration planning for mobile robots. I am grateful for the freedom I was given to pursue heuristics for partial order planning as a research project on the side, which helped me build confidence in my own ability to conduct original research. I admire his ability to quickly get familiar with a new topic and always having something insightful to say. It has been a great pleasure to have had the opportunity to work with him.

The topic for my dissertation started to take shape during an internship at Honeywell Laboratories in the Summer of 2001, where I had the opportunity to work with David Musliner. He put me to work on a simulation based verifier for a probabilistic extension of CIRCA, which later grew into my work on statistical probabilistic model checking. He deserves credit as the originator of my dissertation topic and for making the Summer of 2001 one that I will remember fondly for the rest of my life. I returned to Honeywell in the Summer of 2002. This time I was working closely with Vu Ha on decision theoretic planning, which helped me tremendously in later formulating the GSMDP framework.

I thank Edmund Clarke, Geoffrey Gordon, and Jeff Schneider for the effort they put into reviewing my dissertation. I also want to thank Michael Littman for letting me help with the organization of the probabilistic track of the 4th International Planning Competition. It was a valuable experience for me. In addition, I thank Marta Kwiatkowska, Gethin Norman, and David Parker for a great collaboration on the assessment of algorithms for probabilistic model checking. I am truly grateful for the opportunity to work with them. I am also grateful for being invited to attend a Dagstuhl seminar on probabilistic methods in verification and planning. It was during this seminar that I was exposed to the concept of phase-type distributions.

I am indebted to Magnus Boman, my Master's thesis advisor at the Royal Institute of Technology (KTH) in Stockholm, for fostering my appetite for scientific research. Memories from the six months I spent in Boman's research group after completing my Master's thesis bring me great joy. My colleagues from that time, Johan Kummeneje and David Lybäck, are still good friends of mine. I miss the open and collaborative relationship we had, and I dearly miss eating a steady lunch every day in good company. Sadly, lunch is often eaten in a rush out of a box at one's office desk in the United States, rather than on a real plate in a lunch restaurant with a plentiful complimentary salad bar, as is traditional in Sweden.

Good friends are hard to come by. I thank my very best friends Roland Kalmkvist, Jonas Berglind, and Richard Hagel for all the good times we had together through the years. I also want to thank my fellow graduate students Sungwoo Park, Rune Jensen, and Patrick Riley for their good friendship and support. My officemate Joshua Dunfield must not pass unmentioned either, with whom I had lengthy philosophical and political arguments. For the record, I do not believe everything he says (although my dear wife thinks I do). On the other hand, I sure hope he does not pay too much attention to all the nonsense that comes out of my mouth. I also want to mention Magnus Christensson (a friend of mine in elementary school) and Odd Möller, who both spurred my interest in computer programming and computer science.

A tree is no stronger than its roots. I thank my parents, Ingrid and Samir Younes, for their liberal upbringing. They did not always understand my desires and goals, and I am still not sure they understand why anyone would go to graduate school instead of getting a "real job", but they always gave me their full support. My brother, Stefan, deserves mention as well. We had a great childhood together, went through some rough years when I was the geek and he was the cool kid, and now have a healthy relationship as two people trying to raise a new generation of Youneses on separate shores of the Atlantic ocean.

In the past few years, however, the person closest to my heart has been my beloved wife Genevieve. Her love is the inspiration for my professional achievements. I finally thank Kathleen Hower, my mother-in-law, for giving me a family in a foreign and sometimes inhospitable country thousands of miles away from my dear Sweden.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Stochastic processes with asynchronous events (and actions) are abundant in the real world. The canonical example is a simple queuing system with a single service station, for example modeling your local post office. Customers arrive at the post office, wait in line until the service station is vacant, spend time being serviced by the clerk, and finally leave. We can think of the arrival and departure (due to service completion) of a customer as two separate *events*. There is no synchronization between the arrival and departure of customers, i.e. the two events just introduced are *asynchronous*, so this is clearly an example of an asynchronous system. Other examples of asynchronous systems include telephone exchanges and computer networks.

When we talk about stochastic processes, we are primarily concerned with random variations in the timing of events, for example the duration of a phone call (timing of a "hang up" event) or the lifetime of an electronic component (timing of a "fail" event). We assume that we are given a probability distribution accurately capturing the timing of events. We do not concern ourselves with how these probability distributions are obtained, although we expect them to be based on a collection of empirical measurements to which we fit an analytic distribution function. For example, the duration of a phone call is typically modeled using an exponential distribution, while component lifetime often is found to match a Weibull distribution.

1.1 Two Problems

In this thesis, we consider two separate problems concerning stochastic processes with asynchronous events: *verification* and *planning*. For verification, we are given a system, or a model of a system, and are asked to determine whether the system satisfies some given property. The solution to a verification problem is a "yes" or "no" answer. In the case of a telephone exchange, for example, we may want to verify that the probability is at least 0.9999 that no calls are dropped in a 24-hour period. For planning purposes, we inject a decision dimension into the model, and are asked to find a course of action that will enable a goal to be attained or some expected reward to be maximized. For instance, for a network of computers, we can introduce different service actions and then try to find a service policy that will give us the best value.

Verification and planning can be seen as vital steps in the development of functional systems. Through planning, we obtain a system design, and verification is used to ensure that the system design is satisfactory.

1.1.1 Verification

Probabilistic verification of continuous-time stochastic processes has received increasing attention in the model checking community in the past five years, with a clear focus on developing numerical solution methods for model checking of continuous-time Markov processes. Numerical techniques tend to scale poorly with an increase in the size of the model (the "state space explosion problem"), however, and are feasible only for restricted classes of stochastic discrete event systems.

We present a *statistical* approach to probabilistic model checking, employing hypothesis testing and discrete event simulation. Our solution method works for any discrete event system that can be simulated, and can be used to verify systems too large for numerical analysis. Since we rely on statistical hypothesis testing, we cannot guarantee that the verification result is correct, but we can at least bound the probability of generating an incorrect answer to a verification problem. Another advantage of our model checking algorithm, as with most statistical solution methods, is that it is trivially parallelizable, so we can solve problems faster in a distributed fashion by utilizing multiple interconnected computers.

1.1.2 Planning

Planning for stochastic processes with asynchronous events and actions has received little attention in the artificial intelligence (AI) literature, although some attention has recently been given to planning with *concurrent* actions. Guestrin et al. (2002) and Mausam and Weld (2004) use discrete-time *Markov decision processes* (MDPs) to model and solve planning problems with concurrent actions, but the approach is restricted to instantaneous actions executed in synchrony. Rohanimanesh and Mahadevan (2001) consider planning problems with temporally extended actions that can be executed in parallel. By restricting the temporally extended actions to *Markov options*, the resulting planning problems can be modeled as discrete-time *semi*-Markov decision processes (SMDPs).

All three of the approaches cited above model time as a discrete quantity. This is a natural model of time for synchronous systems driven by a global clock. Asynchronous systems, on the other hand, are best represented using a dense (continuous) model of time (Alur et al. 1993). Continuous-time MDPs (Howard 1960) can be used to model asynchronous systems, but are restricted to events and actions with exponential trigger time distributions. Continuous-time SMDPs (Howard 1971b) lift the restriction on trigger time distributions, but cannot model asynchrony.

We introduce the *generalized* semi-Markov decision process (GSMDP), based on the GSMP model of discrete event systems (Glynn 1989), as a model for asynchronous stochastic decision processes. A GSMDP, unlike an SMDP, remembers if an event enabled in the current state has been continuously enabled in previous states without triggering. This is key in modeling asynchronous processes, which typically involve events that race to trigger first in a state, but the event that triggers first does not necessarily disable the competing events. For example, if a customer is currently being serviced at the post office, the fact that another customer arrives does not mean that the service of the first customer has to start over from scratch. By including a real-valued clock for each event in the description of states, we can model a GSMDP as an MDP, but this will be a *general state space, continuous-time* MDP.

We present two different solution methods for GSMDPs. First, we consider the problem of planning for goal achievement, and present a planning framework based on the Generate, Test and Debug (GTD) paradigm introduced by Simmons (1988). This work ties together our efforts in planning and verification. The second solution method is based on a decision theoretic framework, and we present the use of phase-type distributions (Neuts 1981) to approximate a GSMDP with a continuous-time MDP that then can be solved exactly (or approximately).

1.2 Summary of Research Contribution

Stochastic models with asynchronous events can be rather complex, in particular if the Markov assumption does not hold, such as if event delays are not exponentially distributed for continuous-time models. Many phenomena in nature are, in fact, best modeled with non-exponential distributions, for example, the lifetime of a product (Nelson 1985) or a computer process (Leland and Ott 1986). Yet, the Markov assumption is commonly made, and the attention in the AI planning literature, in particular, is given almost exclusively to discrete-time models, which are inappropriate for asynchronous systems. We believe, however, that the complexity of asynchronous systems is manageable. More precisely, we set out to provide evidence for the following statement:

Thesis. *Verification and planning for stochastic processes with asynchronous events can be made practical through the use of statistical hypothesis testing and phase-type distributions.*

We will support this statement by developing a set of techniques and tools for verification and planning with asynchronous events. In verification, we provide a unifying semantics for interpreting probabilistic temporal logic formulae over general stochastic discrete event systems. We have developed a statistical approach to probabilistic model checking, based on hypothesis testing and simulation. The main theoretical results are Theorems 5.4 and 5.8, which establish the verification procedure for conjunctive and nested probabilistic statements. We show, through empirical studies, that our approach compares well with state-of-the-art numerical techniques for model checking Markov processes. We also show that the use of memoization and heuristics for selecting the verification error of nested probabilistic operators can make statistical verification of properties with nested probabilistic statements work in practice. Finally, we consider the verification of so called “black-box” systems, which are systems that have already been deployed and cannot be simulated, and make explicit the assumptions required for it to produce reliable results.

In planning, we establish a framework for stochastic decision processes with asynchronous events. We consider both goal directed and decision theoretic (reward oriented) planning. For goal directed planning, we use our statistical model checking algorithm to verify plans. Plans that fail to satisfy a given goal condition are repaired, and we rely on the execution traces generated during plan verification to find reasons for failure. We show that the information obtained from the execution traces can help us understand why a plan fails, and can also be used to guide automated plan repair. For decision theoretic planning, we introduce the GSMDP model, and show how phase-type distributions can be used to approximate a GSMDP with a continuous-time MDP. We show, through experiments, that the introduction of phases can help us produce better policies (in terms of expected reward) by allowing us to take history dependence into account.

We would like to highlight two tools, in particular, that have come out of our research effort and are now available to the public. These are YMER[1], a tool for probabilistic model checking, and TEMPASTIC-DTP[2], which is our decision theoretic planner for GSMDPs.

1.3 Overview of Thesis

This thesis is divided into two parts, corresponding to the two different research problems that we address: verification and planning. The two parts are to a large extent independent of each other. We rely on the verification work when we discuss goal directed planning in Chapter 8, but only on an abstract level. The separation into two largely independent parts is made with a heterogeneous audience in mind. The target audience for the part on verification is the model checking community, while the part on planning primarily targets researchers in artificial intelligence. To accommodate readers with a cross-disciplinary inclination, we provide a comprehensive introduction in Chapter 2 to terminology, notation, and techniques that are used extensively throughout the remainder of the thesis. Chapter 3 provides the context for our research contribution with a discussion of related work in probabilistic verification and planning under uncertainty.

Part I consists of a thorough presentation and evaluation of our statistical approach to probabilistic model checking. We start in Chapter 4 by introducing the *unified temporal stochastic logic* (UTSL) for specifying properties of stochastic discrete event systems. UTSL represents a unification of Hansson and Jonsson's (1994) PCTL, which has a semantics defined for discrete-time Markov processes, and Baier et al.'s (2003) version of CSL, which has a semantics defined for continuous-time Markov processes. We provide a semantics for UTSL that is defined in terms of general stochastic discrete event systems.

Chapter 5 introduces a model checking algorithm for UTSL, based on statistical hypothesis testing. This work originated in an effort to verify plans for complex stochastic temporal domains, with a focus on probabilistic time-bounded reachability properties (Younes and Musliner 2002). Time-bounded CSL properties were later considered (Younes and Simmons 2002b), although with an unsatisfactory solution for conjunctive and nested probabilistic operators. These shortcomings have now been addressed, and a sound and practical solution to the verification of properties with nested probabilistic operators is presented for the

[1] http://www.cs.cmu.edu/~lorens/ymer.html
[2] http://www.cs.cmu.edu/~lorens/tempastic-dtp.html

first time in this thesis.

Chapter 6 provides an empirical evaluation of our model checking algorithm and a comparison with numerical solution methods. The comparative study extends a previously published (Younes et al. 2004) comparison of statistical and numerical solution methods for probabilistic model checking. The results are intended as an aid to practitioners when facing a choice between different solution techniques, or when selecting parameters for a specific solution method.

The model checking algorithm presented in Chapter 5 relies on the ability to generate sample trajectories for a stochastic discrete event system on demand. In Chapter 7, we consider a situation where this is not possible, for example, if we want to verify an already deployed system for which we have no model. We assume that we are provided with a finite set of sample trajectories, and show how to statistically verify UTSL properties based on this limited source of information about a system. This chapter, which concludes the part on verification, is based on a previously published technical report (Younes 2004).

In Part II, we consider the problem of planning with asynchronous events and actions. We describe two complementary approaches. Chapter 8 describes a goal directed approach. We present a general planning framework for generating stationary policies for controllable stochastic discrete event systems that satisfy UTSL goal conditions. The statistical model checking algorithm is used for policy verification, and policies that do not satisfy a given goal condition are repaired. We rely on the sample trajectories generated during the verification phase to guide the repair effort. This chapter is based on work reported two consecutive years at ICAPS (Younes et al. 2003; Younes and Simmons 2004a).

A decision theoretic approach to planning with asynchronous events and actions is presented in Chapter 9, where we introduce the generalized semi-Markov decision process (GSMDP). We present the use of continuous-phase type distributions to approximate a GSMDP with a continuous-time MDP, which can then be solved exactly. We extend the work of Younes and Simmons (2004c) by considering additional techniques for approximating a general distribution with a phase-type distribution. The "Bellman equation" for a GSMDP first appeared in a workshop paper (Younes and Simmons 2004b).

Finally, Chapter 10 discusses directions for future work in verification and planning. For verification, this includes statistical techniques for verifying steady-state properties and the use of symbolic data structures for faster discrete event simulation. In planning, we call for a formal analysis of optimal GSMDP planning and discuss the possibility of using value function approximation techniques to solve GSMDPs.

Chapter 2

Background

This chapter introduces terminology and techniques that will be used extensively in later chapters. Readers already familiar with concepts such as *random variable*, *probability distribution*, *acceptance sampling*, and *stochastic process* may still find it useful to read this chapter, as our notation may differ from what they are used to. In particular, this is the case for standard parametric probability distributions, and we refer the reader to Table 2.1 for a summary of our notation for important distributions.

2.1 Random Variables and Probability Distributions

Consider the chance experiment of observing the outcome of a die roll. The possible observations are the integers 1 through 6. For a regular die, we assume that each outcome is equally likely, i.e. outcome i is observed with probability $1/6$. Now, consider a chance experiment that consists of observing the duration of a phone call. The outcome of this experiment is a positive real number, rather than an integer, and there is some probability of observing a call with a duration no longer than t.

Formally, we represent a chance experiment with a *random variable* (Feller 1957; Wadsworth and Bryan 1960), also called a *variate*. A random variable X can take on any value in an outcome space Ω, and we associate a non-negative weight $f(x)$ with each possible outcome $x \in \Omega$. The outcome space, as illustrated by the two examples in the previous paragraph, can be discrete or continuous. We assume, for simplicity, that the outcome space is either the integers or the real numbers. In the former case, we call X a *discrete random variable*, while in the latter case X is referred to as a *continuous random variable*. Impossible

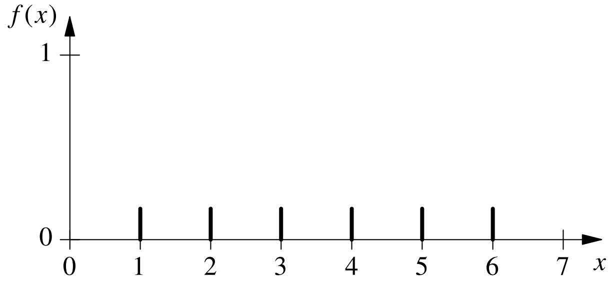


Figure 2.1: Probability density function for a discrete random variable.

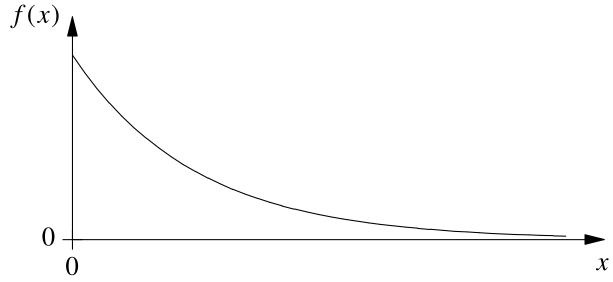


Figure 2.2: Probability density function for a continuous random variable.

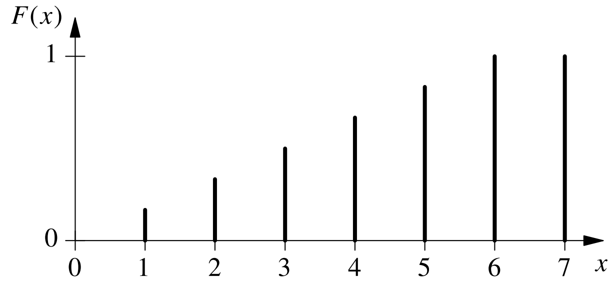


Figure 2.3: Cumulative distribution function for a discrete random variable.

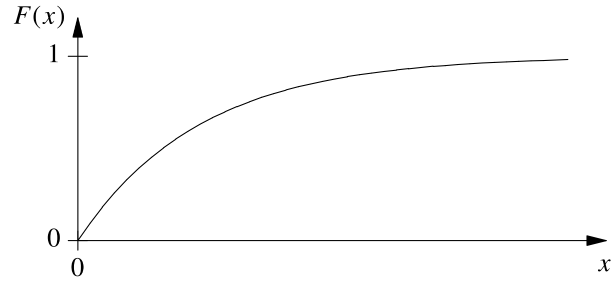


Figure 2.4: Cumulative distribution function for a continuous random variable.

outcomes, for example 7 in the die roll experiment, are assigned zero weight.

The total weight for the outcome space must equal unity. In other words, the weight function f must satisfy the condition $\int_\Omega f(x) = 1$. For discrete outcome spaces, $f(x)$ is simply the probability associated with outcome x. In the continuous case, $f(x)$ is *not* a probability, however, and $f(x)$ can be greater than 1. For example, $f(x)$ is either 0 or 2 for a continuous uniform distribution over the interval $(0, 0.5)$. The function $f(x)$ is called the *probability density function* for the random variable X. Figures 2.1 and 2.2 show the probability density function for a discrete and a continuous random variable, respectively. The *support* of a probability distribution is the subset of the outcome space Ω with positive weight. It is $\{1, 2, 3, 4, 5, 6\}$ for the distribution in Figure 2.1 and $[0, \infty)$ for the distribution in Figure 2.2.

The probability that the value of X is at most t, $\Pr[X \leq t]$, is a function $F(t)$ called the *cumulative distribution function*. We have $F(t) = \sum_{x=-\infty}^{t} f(x)$ for discrete random variables and $F(t) = \int_{-\infty}^{t} f(x)\, dx$ for continuous random variables. Since $f(x)$ is non-negative for all values of x, $F(t)$ is a non-decreasing function of t, $\lim_{t\to-\infty} F(t) = 0$, and $\lim_{t\to\infty} F(t) = 1$. A probability distribution is *positive* if $F(0) = 0$. Figures 2.3 and 2.4 show two examples of cumulative distribution functions for positive distributions.

We can obtain new random variables as functions of existing random variables. In the board game Monopoly, for example, a player rolls two dice at once and adds the outcome of the two rolls to determine the number of steps to take on the board. Let X_1 and X_2 be random variables representing the individual die rolls. The sum $X_1 + X_2$ is another random variable Y representing the chance experiment of simultaneously rolling two identical dice and summing up their outcomes. In general, a function $g(X_1, \ldots, X_n)$ of n random variables is itself a random variable Y with some probability density function and cumulative distribution function.

2.1.1 Expectation, Variance, and Moments

The probability density function or cumulative distribution function for a random variable X fully characterizes the chance experiment represented by X. It is common to present a set of summarizing statistics for the experiment instead of the whole distribution function. The most commonly used summarizing statistic is the *mean*, or *expected value*, of a random variable. The expected value of X, denoted $\mathrm{E}[x]$, is defined as $\mu = \sum_{x=-\infty}^{\infty} xf(x)$ for discrete distributions and $\mu = \int_{-\infty}^{\infty} xf(x)\,dx$ for continuous distributions. The value μ represents the "expected outcome" of a chance experiment, but does not necessarily correspond to a possible outcome. In the case of a single die throw, for example, we have $\mu = 3.5$.

While the mean is a measure of location for a random variable, the *variance* of X, denoted $\mathrm{Var}[X]$ or σ^2, is a measure of spread. It is defined as $\sigma^2 = \mathrm{E}[(X-\mu)^2]$, where μ is the mean of X. The square root of the variance, σ, is called the *standard deviation*, and is sometimes preferred as a measure of spread in practice because σ and μ are of the same unit of measurement. For example, if μ is the average length of a phone call in seconds, then σ measures the spread in seconds, while σ^2 gives a measure of spread in squared seconds. The spread can also be specified using the *coefficient of variation*, defined as $cv = \sigma/\mu$, or the squared coefficient of variation (cv^2), which gives a measure of spread that is relative to the location μ.

The mean of a random variable is a special case of a set of summarizing statistics called *moments*. The ith moment of a random variable X is defined as $\mu_i = \mathrm{E}[X^i]$. Obviously, the mean of X is μ_1. The variance, σ^2, can be expressed using the first two moments:

$$\sigma^2 = \mathrm{E}[(X-\mu_1)^2] = \mathrm{E}[X^2] - 2\mu_1\,\mathrm{E}[X] + \mu_1^2 = \mu_2 - \mu_1^2$$

The squared coefficient of variation, cv^2, is therefore equal to $(\mu_2/\mu_1^2) - 1$.

Distribution	$F(x)$	μ	σ^2
Bernoulli	$\begin{cases} 0 & \text{if } x < 0 \\ 1-p & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$	p	$p(1-p)$
Geometric, $G(p)$	$1-(1-p)^x \quad (x \geq 0)$	$\frac{1}{p}$	$\frac{1-p(1-p)}{p^2}$
Binomial, $B(n,p)$	$\sum_{i=0}^{x} \binom{n}{i} p^i (1-p)^{n-i}$	np	$np(1-p)$
Uniform, $U(a,b)$	$\begin{cases} 0 & \text{if } x < a \\ (x-a)/(b-a) & \text{if } a \leq x \leq b \\ 1 & \text{if } x > b \end{cases}$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
Exponential, $Exp(\lambda)$	$1-e^{-\lambda x} \quad (x \geq 0)$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
Weibull, $W(\eta,\beta)$	$1-e^{-(x/\eta)^\beta} \quad (x \geq 0)$	$\eta\Gamma(1+\beta^{-1})$	$\eta^2\big(\Gamma(1+2\beta^{-1}) - \Gamma^2(1+\beta^{-1})\big)$
Lognormal, $L(\mu,s)$	$\Phi(s^{-1}\log(x/\mu) - s/2) \quad (x \geq 0)$	μ	$\mu^2\big(e^{s^2}-1\big)$

Table 2.1: Common parametric probability distributions.

2.1.2 Parametric Distributions

A probability distribution can be almost arbitrarily complex, but many important phenomena in nature can be fairly accurately described using only a few parameters. We call a distribution *parametric* if the shape of its distribution function is determined by the values of a finite number of parameters. Table 2.1 shows the cumulative distribution function, mean, and variance for seven parametric distributions that will occur frequently in this thesis. Next, we describe each of these distributions in more detail.

Let the random variable X represent the chance experiment of tossing an unbiased coin. The probability distribution associated with X can be specified using the single parameter $p = 1/2$, and is an example of a *Bernoulli* distribution. The random variable X is called a *Bernoulli variate* and the chance experiment represented by X is a *Bernoulli trial*. In general, the Bernoulli distribution can be used to model any chance experiment with two distinct outcomes, typically encoded by the integers 0 and 1, and with a probability p of outcome 1 occurring.

Next, consider an experiment where we toss a coin repeatedly until we get a head. Let X be a random variable with value equal to the number of coin tosses in an experiment. In this case, X is said to have a *geometric* distribution with parameter $p = 1/2$. The probability of observing that X has value x (i.e. that

x coin tosses are required to get one head in a specific experiment) is $p(1-p)^x$, which is the probability density function for the geometric distribution. The probability of observing x tails in a row is $(1-p)^x$, so the cumulative distribution function is $F(x) = 1-(1-p)^x$.

Let $X_1, \ldots, X_n$ be n independent and identically distributed Bernoulli variates with parameter p. The random variable $Y = \sum_{i=1}^{n} X_i$ then has a *binomial* distribution with parameters n and p, denoted $B(n,p)$. If we carry out n independent coin tosses, for example, then the number of heads that we observe is binomially distributed with $p = 1/2$. The binomial distribution will play a central role in the next section, when we discuss acceptance sampling, which is the technique we will later use for statistical probabilistic model checking.

A random variable representing a die roll has a discrete *uniform* distribution with $f(x) = 1/6$ for $x \in \{1, \ldots, 6\}$ (Figures 2.1 and 2.3 plot $f(x)$ and $F(x)$, respectively, for this distribution). The uniform distribution can also be defined over a continuous interval (a,b), with $f(x) = (x-a)/(b-a)$ for $x \in (a,b)$. The uniform distribution has finite support, unlike the geometric distribution and the three continuous distributions mentioned below which all have infinite support.

The *exponential* distribution, with cumulative distribution function $F(x) = 1-e^{-\lambda x}$, is one of the most widely used continuous distributions due to its favorable analytical properties. The parameter λ is the *rate* of the distribution, for example representing the failure rate of an electrical component or the arrival rate of customers at a post office. Figures 2.2 and 2.4 plot $f(x)$ and $F(x)$, respectively, for the exponential distribution with $\lambda = 1$. The exponential distribution is *memoryless*. This means that if X is a random variable with an exponential distribution, then $\Pr[X > t+s \mid X > t] = \Pr[X > s]$. The geometric distribution, which in many ways can be seen as a discrete version of the exponential distribution, is also memoryless, and these are in fact the only memoryless distributions (Feller 1957, p. 305). The memoryless property is essential for analytical tractability in many applications.

Not all phenomena in the real world can be properly captured by a memoryless distribution. Component lifetime, for example, is often not memoryless. Failure may be more likely early on during a warm-up period than when a system has been running for a while, or it could be the case that the failure rate increases with time due to material fatigue. The *Weibull* distribution (Weibull 1951), with cumulative distribution function $F(x) = 1-e^{-(x/\eta)^\beta}$, is commonly used in reliability engineering for this purpose. The parameter η is a scale parameter, while β is a shape parameter with $0 < \beta < 1$ giving a decreasing failure rate and $\beta > 1$

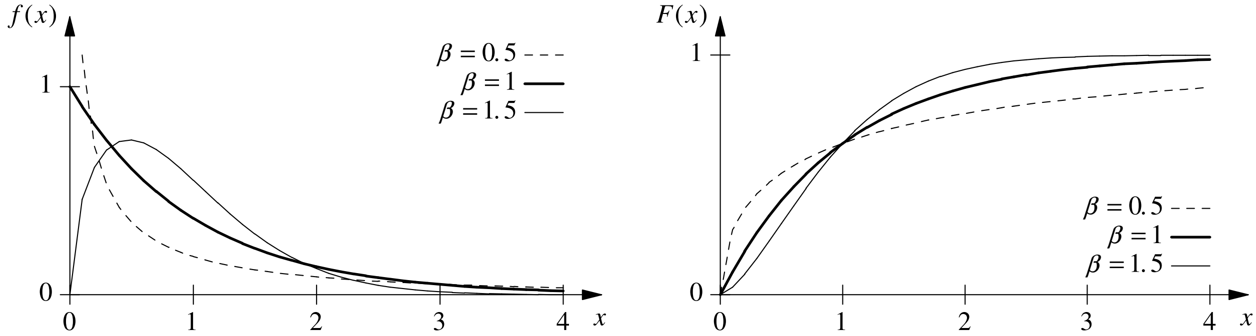


Figure 2.5: Probability density function (left) and cumulative distribution function (right) for the Weibull distribution.

giving an increasing failure rate. The mean and variance of a Weibull distribution are defined in terms of the *gamma function*, $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}\,dt$, as shown in Table 2.1. If β is equal to 1, then the Weibull distribution is simply an exponential distribution with rate $1/\eta$. Figure 2.5 shows the probability density function and cumulative distribution function for three different values of β.

The *lognormal* distribution is another probability distribution commonly used in reliability engineering. If X is a random variable with a lognormal distribution, then $Y = \log X$ is a normal variate. The cumulative distribution function for the standard normal distribution ($\mu = 1$ and $\sigma = 0$) is given by the formula

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\,dt \quad , \tag{2.1}$$

and Table 2.1 shows the distribution function for the lognormal distribution in terms of $\Phi(x)$.

2.1.3 Phase-Type Distributions and Approximation Techniques

The exponential distribution, with its memoryless property, is often used in models of stochastic systems. This results in models for which tractable solution techniques for many problems (e.g. model checking and planning) exist. *Phase-type distributions* (Neuts 1975, 1981), both discrete and continuous, generalize the exponential distribution to permit memory dependence in the form of *phases*. We will use phase-type distributions in Chapter 9 to approximate non-exponential parametric distributions for the purpose of solving decision theoretic planning problems with asynchronous events.

Erlang (1917) was the first to consider a generalization of the exponential distribution that preserves much of its analytic tractability. Let $X_1, \ldots, X_n$ be n random variables, all having an exponential distribution with rate λ. The random variable $Y = \sum_{i=1}^{n} X_i$ is then said to have an *Erlang* distribution with

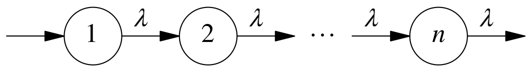


Figure 2.6: Erlang distribution.

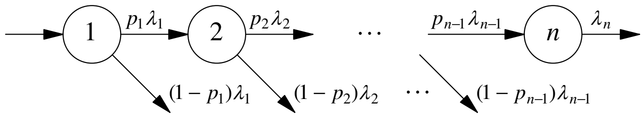


Figure 2.7: Coxian distribution.

parameters n and λ. The Erlang distribution can be thought of as a chain of n phases where the time spent in each phase before transitioning to the next phase is exponentially distributed with rate λ (Figure 2.6). The random variable Y represents the time from entry of the first phase until exit of the last phase. A *generalized Erlang* distribution includes the possibility of exiting the chain already after the first phase (there is a probability p of transitioning to the second phase).

A *Coxian* distribution (Cox 1955) is a further generalization of the Erlang distribution, permitting phase-dependent transition rates and a probability $q_i = (1 - p_i)$ of bypassing the remaining phases after exiting phase i. Figure 2.7 shows an n-phase Coxian distribution. Note that a Coxian distribution with n phases has $2n - 1$ parameters, while an n-phase Erlang distribution only has a single parameter (the rate λ).

The Erlang and Coxian distributions are special cases of the class of phase-type distributions. In general, a phase-type distribution with n phases represents the time from entry until absorption in a *Markov process* (see Section 2.3.3) with n transient states and a single absorbing state. We are primarily interested in continuous phase-type distributions, as this thesis is concerned with asynchronous systems, which are best represented using a continuous model of time. The general form of an n-phase continuous phase-type distribution is specified using $n^2 + 2n$ parameters:

- λ_i, for $1 \leq i \leq n$, representing the exit rate for phase i.
- p_{ij}, for $1 \leq i, j \leq n$, representing the probability that phase i is followed by phase j. The probability $q_i = 1 - \sum_{j=1}^{n} p_{ij}$ is the probability of absorption immediately following phase i.
- α_i, for $1 \leq i \leq n$, representing the probability that the initial phase is i.

If we define an $n \times n$ matrix Q, with elements $Q_{ii} = -\lambda_i(1 - p_{ii})$ and $Q_{ij} = \lambda_i p_{ij}$ $(i \neq j)$, and a row vector $\vec{\alpha} = [\alpha_i]$, then the cumulative distribution function for a continuous phase-type distribution is given by $F(x) = 1 - \vec{\alpha}e^{Qx}\vec{e}$, where $\vec{e}$ is a unit column vector of size n. The kth moment of the distribution is $\mu_k = k!\vec{\alpha}(-Q)^{-k}\vec{e}$. It is common to consider only *acyclic* phase-type distributions, where phase j can be reached only from phases $i < j$, because they require fewer parameters.

We can use a phase-type distribution PH to approximate a general distribution G, for example a Weibull or lognormal distribution. The most straightforward approximation technique is the *method of moments*, where the objective is to match the first k moments of G and PH. When using the method of moments, it is desirable to match as many moments of G as possible, but we will typically need more phases to match more moments, so there is a tradeoff between accuracy and complexity of the approximate model. The objective is often to find a phase-type distribution that matches a fixed number of moments and is minimal (in terms of the number of phases), or close to minimal, within a certain class of phase-type distributions (e.g. acyclic phase-type distributions).

We can easily match a single moment of a general distribution G by using an exponential distribution with rate $1/\mu_1$, but this typically yields a poor approximation of G. It is possible to match the first two moments of any positive distribution using either a generalized Erlang distribution or a two-phase Coxian distribution. If the squared coefficient of variation, cv^2, is less than 1, then we can use a generalized Erlang distribution with the following parameters (Sauer and Chandy 1975; Marie 1980):

$$n = \left\lceil \frac{1}{cv^2} \right\rceil \qquad \lambda = \frac{1 - p + np}{\mu_1}$$
$$p = 1 - \frac{2n \cdot cv^2 + n - 2 - \sqrt{n^2 + 4 - 4n \cdot cv^2}}{2(n-1)(cv^2+1)}$$

For example, a uniform distribution $U(0,1)$ ($\mu_1 = 1/2$ and $cv^2 = 1/3$) can be approximated by a three-phase (generalized) Erlang distribution with $p = 1$ and $\lambda = 6$. For distributions with $cv^2 \geq 1/2$, we match the first two moments with a two-phase Coxian distribution with parameters $\lambda_1 = 2/\mu_1$, $\lambda_2 = 1/(\mu_1 \cdot cv^2)$, and $p = 1/(2 \cdot cv^2)$ (Marie 1980). For example, a Weibull distribution $W(1, 1/2)$ has $\mu_1 = 2$ and $cv^2 = 5$, and can therefore be approximated by a two-phase Coxian distribution with $\lambda_1 = 1$, $\lambda_2 = 1/10$, and $p = 1/10$. Whitt (1982) and Altiok (1985) show how to find a phase-type distribution with only two phases that matches the first three moments of a general distribution, provided that $cv^2 > 1$ and $\mu_3 > 3\mu_2^2/(2\mu_1)$. Telek and Heindl (2002) provide bounds on μ_3, with $cv^2 > 1/2$, for which a two-phase Coxian distribution can be used to match three moments. Johnson and Taaffe (1989) use a mixture of Erlang distributions to match the first three moments of any positive distribution, but the resulting phase-type distribution is a factor two from minimal in the class of acyclic phase-type distributions. Johnson and Taaffe (1990) describe an approach for matching three moments based on nonlinear programming, which results in close to minimal acyclic phase-type distributions. An analytic solution, combining an Erlang distribution with a two-phase Coxian distribution, for matching three moments with close to minimal acyclic phase-type distributions is presented by Osogami and Harchol-Balter (2003).

It is possible to match the first few moments of a distribution without obtaining a good fit for the distribution function. For example, the first two moments do not reveal whether the distribution function has multiple modes. Instead of matching moments of a distribution, we can try to match the shape of the distribution function. The *Kullback-Leibler divergence* (KL-divergence), or *relative entropy*, is a popular similarity measure for distribution functions. Let f and g be two probability density functions. The KL-divergence of f and g is defined as follows (Kullback and Leibler 1951, p. 80):[1]

$$KL(f,g) = \int_{-\infty}^{\infty} f(x) \log \frac{f(x)}{g(x)}\, dx \tag{2.2}$$

Asmussen et al. (1996) use the EM (Expectation-Maximization) algorithm (Dempster et al. 1977) to fit a general phase-type distribution to an arbitrary continuous distribution, minimizing the KL-divergence. Bobbio and Cumani (1992) present a maximum likelihood estimation algorithm for fitting an acyclic phase-type distribution to a continuous distribution. For both fitting algorithms, the user selects the number of phases to use instead of the number of moments to match, with more phases typically resulting in a better fit. These approaches are computationally more costly than the method of moments. The number of iterations required for the EM algorithm to converge tends to grow with the number of phases. Convergence can be reached faster by imposing restrictions on the structure of the phase-type distribution, for example by matching a sum of n exponential distributions or an n-phase Coxian distribution rather than a general phase-type distribution. Figure 2.8 shows the probability density function for the uniform distribution $U(0,1)$ and five different phase-type distributions (two obtained by matching moments, and three obtained through use of the EM algorithm). We need only a single phase to match the first moment of $U(0,1)$, and we need three phases to match the first two moments (achieved by an Erlang distribution, as mentioned earlier).

A continuous distribution can also be approximated by a *discrete* phase-type distribution (Bobbio et al. 2003, 2004). An advantage of using discrete, rather than continuous, phase-type distributions is that a lower coefficient of variation can be achieved with the same number of phases. It is known that with n phases,

[1] The KL-divergence can be thought of as the distance between two probability density functions, although technically it is not a true distance measure because it is not symmetric.

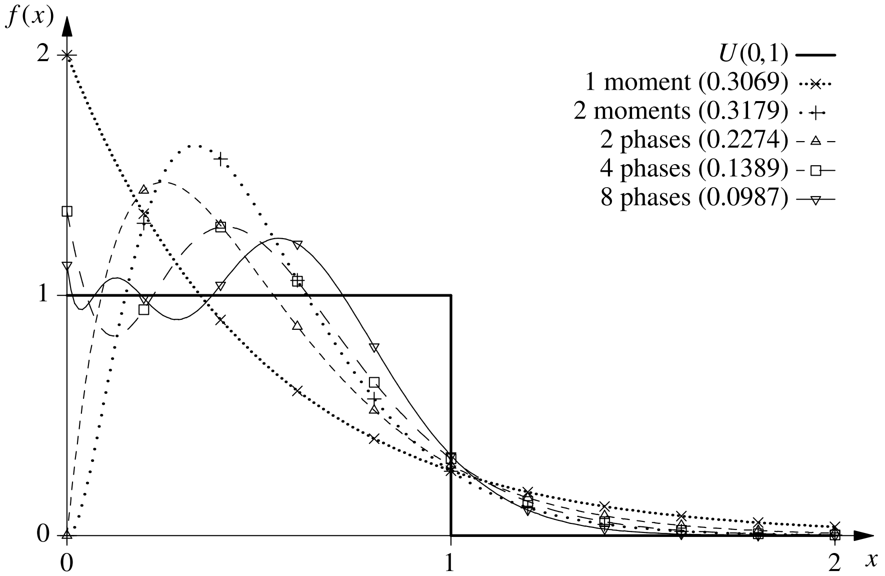


Figure 2.8: Phase-type fitting for uniform distribution. The KL-divergence for each phase-type distribution is shown in parentheses.

cv^2 is at least $1/n$ for a continuous phase-type distribution, with $1/n$ achieved exactly by an n-phase Erlang distribution (Aldous and Shepp 1987). Discrete phase-type distributions can also capture distributions with finite support and deterministic distributions, while continuous phase-type distributions always have infinite support. One clear disadvantage, however, with discrete-time approximations of continuous-time systems is that coincident events must be taken into consideration. With continuous distributions, the probability of two events occurring at the same time is zero, but if we discretize time, two events may occur in the same interval of time. This can significantly increase the complexity of any analysis of the model, and is particularly a problem for analyses of systems with asynchronous events.

2.2 Acceptance Sampling with Bernoulli Trials

A probabilistic model checking problem can be phrased as a *hypothesis testing* problem. We will take advantage of this in Chapter 5 when presenting a statistical approach to probabilistic model checking. As an example of a hypothesis testing problem, consider a manufacturing process that produces units of some product. Each manufactured unit is either functional or defective, and assume that there is some probability p, unknown to us, of the process producing a functional unit. Naturally, we want p to be high, meaning that the expected fraction of functional units, in a lot of produced units, is high. Let θ be the lowest acceptable value of p. By inspecting a limited number of manufactured units, we want to determine if the manufacturing process is acceptable (i.e. $p \geq \theta$). This section discusses how to solve problems like this statistically using a technique called *acceptance sampling*, which we will later use for probabilistic model checking.

2.2.1 Problem Formulation

Let X_i be a random variable having a Bernoulli distribution with parameter p, i.e. $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$. An observation x_i of X_i has value either 0 or 1. For the manufacturing process mentioned above, x_i is 1 if the ith unit that we observe is functional, and 0 if it is defective. Each random variable X_i, called a *Bernoulli trial*, represents the inspection of a manufactured unit and the observation x_i represents the outcome of the inspection. We are interested in testing whether the parameter p of the Bernoulli distribution is above or below some given threshold θ. More specifically, we want to test the hypothesis $H : p \geq \theta$ against the alternative hypothesis $K : p < \theta$.

We are going to consider statistical approaches for solving this hypothesis testing problem, and we generally have to tolerate that any statistical test procedure has some probability of accepting a false hypothesis, but this is tolerable so long as the probability of error is sufficiently low. In particular, the test procedure should limit the probability of accepting the hypothesis K when H holds (known as a type I error, or false negative) to α, and the probability of accepting H when K holds (a type II error, or false positive) should be at most β. We generally assume that both α and β are less than $1/2$. Figure 2.9 plots the probability of accepting H as a function of p, denoted L_p, for a hypothetical acceptance sampling test with ideal performance in the sense that the probability of a type I error is exactly α and the probability of a type II error is exactly β. The parameters α and β determine the *strength* of an acceptance sampling test.

The above problem formulation is flawed, however, as it essentially requires that we can differentiate between $p = \theta$ and $p = \theta - \epsilon$ for arbitrary $\epsilon > 0$. For $p = \theta$, we require the probability of accepting H to be at least $1 - \alpha$, but for p only infinitesimally smaller than θ, the probability of accepting H is required to be at most β. For this to work, we either need to conduct exhaustive sampling, which is impractical if the sample population is large, or we need to have $1 - \alpha = \beta$, which means that if one error probability is set low then the other is required to be high. In order to avoid exhaustive sampling and obtain the desired control over the two error probabilities, we relax the hypothesis testing problem by introducing two thresholds p_0 and

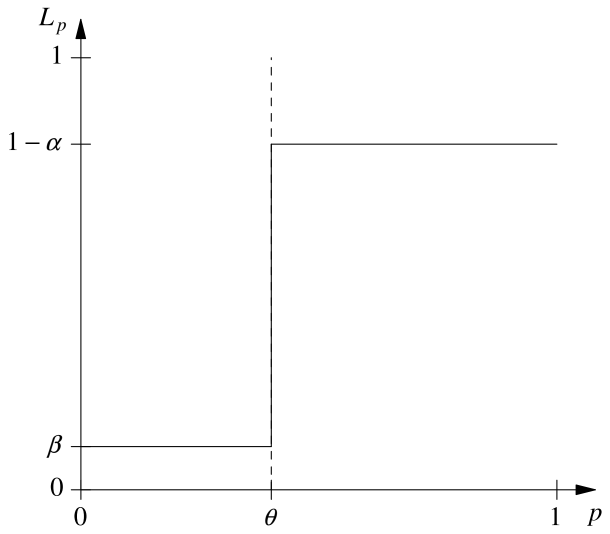


Figure 2.9: Probability, L_p, of accepting the hypothesis $H : p \geq \theta$ as a function of p for a hypothetical statistical test.

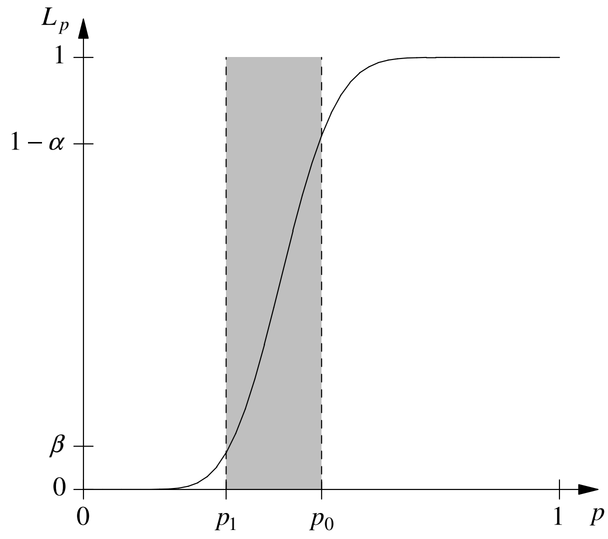


Figure 2.10: Probability, L_p, of accepting the hypothesis $H_0 : p \geq p_0$ as a function of p for a statistical test with indifference region.

p_1, with $p_0 > p_1$. Instead of testing H against K, we choose to test the hypothesis $H_0 : p \geq p_0$ against the alternative hypothesis $H_1 : p \leq p_1$. We require that the probability of accepting H_1 when H_0 holds is at most α, and the probability of accepting H_0 when H_1 holds is at most β. Figure 2.10 shows the typical performance characteristic for a realistic acceptance sampling test. If the value of p is between p_0 and p_1, we are indifferent with respect to which hypothesis is accepted, and both hypotheses are in fact false in this case. The region (p_1, p_0) is referred to as the *indifference region* and it is shown as a gray area in Figure 2.10.

We will often find it appropriate to define the two thresholds p_0 and p_1 in terms of a single threshold θ and the half-width of the indifference region δ, i.e. $p_0 = \theta + \delta$ and $p_1 = \theta - \delta$. Testing H_0 against H_1 can then be interpreted as testing the hypothesis $H : p \geq \theta$ against the alternative hypothesis $K : p < \theta$, as originally specified, where acceptance of H_0 results in acceptance of H and acceptance of H_1 results in acceptance of K. The probability of accepting H is therefore at least $1 - \alpha$ if $p \geq \theta + \delta$ and at most β if $p \leq \theta - \delta$. If $|p - \theta| < \delta$, then the test gives no bounds on the probability of accepting a false hypothesis. In this case, however, we say that p is sufficiently close to the threshold θ so that we are indifferent with respect to which of the two hypotheses, H or K, is accepted. By narrowing the indifference region, we can get arbitrarily close to the ideal performance shown in Figure 2.9.

We now turn to the problem of finding a test procedure with the desired characteristics. A set of n

observations is referred to as a *sample* from now on. We first present a test procedure that uses samples of fixed size, and then present a sequential test procedure where the sample size required for a test of a given strength is a random variable. We will see that the sequential test procedure, while giving no upper bound on the sample size for any given run, typically requires far smaller samples on average than a test procedure using samples of predetermined size.

2.2.2 Acceptance Sampling with Fixed-Size Samples

A sample of size n consists of n observations, $x_1, \ldots, x_n$, of the Bernoulli variates $X_1, \ldots, X_n$ that represent our experiment. To test the hypothesis $H_0 : p \geq p_0$ against the alternative hypothesis $H_1 : p \leq p_1$, using a single sample of size n, we specify a constant c. If $\sum_{i=1}^{n} x_i$ is greater than c, then hypothesis H_0 is accepted. Otherwise, if the given sum is at most c, then hypothesis H_1 is accepted. The problem is now to find n and c such that H_1 is accepted with probability at most α when H_0 holds, and H_0 is accepted with probability at most β when H_1 holds. The pair $\langle n, c\rangle$ represents an acceptance sampling test that uses a single fixed-size sample, and we refer to this pair as a *single sampling plan* (Grubbs 1949; Duncan 1974).

Optimal Single Sampling Plans

The probability distribution of a sum of n Bernoulli variates with parameter p is a binomial distribution with parameters n and p, denoted $B(n, p)$. The probability of $\sum_{i=1}^{n} X_i$ being at most c is therefore given by the cumulative distribution function for $B(n, p)$:

$$F(c; n, p) = \sum_{i=0}^{c} \binom{n}{i} p^i (1-p)^{n-i} \tag{2.3}$$

Thus, with probability $F(c; n, p)$ we accept hypothesis H_1 using a single sampling plan $\langle n, c\rangle$, and consequently hypothesis H_0 is accepted with probability $1 - F(c; n, p)$ by the same sampling plan.

If we can find a pair $\langle n, c\rangle$ simultaneously satisfying $F(c; n, p) \leq \alpha$ for all $p \geq p_0$ and $1-F(c; n, p) \leq \beta$ for all $p \leq p_1$, then we have a single sampling plan with strength $\langle \alpha, \beta\rangle$ for testing H_0 against H_1. For fixed c and n, $F(c; n, p)$ is a non-increasing function of p in the interval $[0, 1]$. This means that $F(c; n, p_0) \leq \alpha$ implies $F(c; n, p) \leq \alpha$ for all $p \geq p_0$, and $1 - F(c; n, p_1) \leq \beta$ implies $1 - F(c; n, p) \leq \beta$ for all $p \leq p_1$. Hence, finding a single sampling plan $\langle n, c\rangle$ with the prescribed strength amounts to solving the following system of non-linear inequalities for the integer variables n and c:

$$F(c; n, p_0) \leq \alpha \tag{2.4a}$$

$$1 - F(c; n, p_1) \leq \beta \tag{2.4b}$$

This system of inequalities typically has an infinite number of solutions. We generally prefer sampling plans that use small samples (i.e. require few observations) over those that use large samples, so we want to minimize n subject to (2.4a) and (2.4b). The stated optimization problem does not have a simple, closed-form solution, except in a few special cases discussed below. Peach and Littauer (1946) propose using a Poisson approximation to find a suitable single sampling plan. Grubbs (1949) provide tables with optimal sampling plans for $\alpha = 0.05$, $\beta = 0.10$, and $n \leq 150$. A graphical solution method is provided by Larson (1966, p. 273). With the widespread availability of fast digital computers, however, these solution methods are essentially obsolete.

Algorithm 2.1 is a procedure for finding an optimal single sampling plan given the parameters p_0, p_1, α, and β that specify the hypothesis testing problem and the desired strength of the sampling plan. The algorithm uses binary search to find a minimum sample size, n, under the assumption that c does not have to be an integer. It then searches linearly from the minimum to find a valid single sampling plan. The inverse of the function $\tilde{F}(x; n, p) = (F(\lfloor x \rfloor; n, p) + F(\lceil x \rceil; n, p))/2$, for $x \in [0, n]$, is used extensively by the algorithm. For fixed n and p, $\tilde{F}(x; n, p)$ is a non-decreasing function of x. Thus, $\tilde{F}(x_0; n, p_0) \leq \alpha$ implies $\tilde{F}(x; n, p_0) \leq \alpha$ for all $x \leq x_0$, and $1 - \tilde{F}(x_1; n, p_1) \leq \beta$ implies $1 - \tilde{F}(x; n, p_1) \leq \beta$ for all $x \geq x_1$. As a consequence, if $x_0 \geq x_1$, then any x in the interval $[x_1, x_0]$ can be used to simultaneously satisfy (2.4a) and (2.4b) for the given n. If, on the other hand, $x_0 < x_1$, then we need to use a sample larger than n in order to obtain a test with the desired strength.

Example 2.1. For probability thresholds $p_0 = 0.5$ and $p_1 = 0.3$, and error bounds $\alpha = 0.2$ and $\beta = 0.1$, the optimal single sampling plan found by Algorithm 2.1 is $\langle 30, 12 \rangle$. This means that we need a sample of size 30, and we accept the hypothesis $p \geq 0.5$ if and only if the sum of the 30 observations exceeds 12. Figure 2.10 (p. 18) plots the probability $L_p = 1 - F(12; 30, p)$ of accepting the hypothesis $H_0 : p \geq 0.5$ as a function of p. We can see that for values of p far away from the indifference region, the probability of accepting a false hypothesis is virtually zero. Note also that $1 - F(12; 30, p_1) \approx 0.084 < \beta$ and $F(12; 30, p_0) \approx 0.181 < \alpha$, so the actual strength of the test is better than $\langle \alpha, \beta \rangle$.

```
SINGLE-SAMPLING-PLAN(p0, p1, α, β)
  n_min ⇐ 1, n_max ⇐ −1
  n ⇐ n_min
  while n_max < 0 ∨ n_min < n_max do
    x0 ⇐ F̃^{-1}(α; n, p0)
    x1 ⇐ F̃^{-1}(1 − β; n, p1)
    if x0 ≥ x1 ∧ x0 ≥ 0 then
      n_max ⇐ n
    else
      n_min ⇐ n + 1
    if n_max < 0 then
      n ⇐ 2 · n
    else
      n ⇐ ⌊(n_min + n_max)/2⌋
  n ⇐ n_max − 1
  repeat
    n ⇐ n + 1
    c0 ⇐ ⌊F̃^{-1}(α; n, p0)⌋
    c1 ⇐ ⌈F̃^{-1}(1 − β; n, p1)⌉
  until c0 ≥ c1
  return ⟨n, ⌊(c0 + c1)/2⌋⟩
```

Algorithm 2.1: Procedure for finding an optimal single sampling plan using binary search. $\tilde{F}^{-1}(y; n, p)$ can be computed by adding the terms of (2.3) until the sum equals or exceeds y.

Sample Sizes

How large a sample is required to obtain a single sampling plan of strength $\langle \alpha, \beta \rangle$ for testing $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$? In general, we can give only an approximate answer, but there are two special cases for which n can be expressed precisely as a formula of the test parameters.

First, consider the case when $p_1 = 0$ and $p_0 < 1$. From (2.3) it follows that $F(c; n, 0) = 1$ for all choices of n and c, so (2.4b) is trivially satisfied. The reasoning behind Algorithm 2.1 tells us that choosing c as low as possible makes it easier to satisfy (2.4a). We therefore set $c = 0$, which gives us $F(c; n, p_0) = (1 - p_0)^n$. We can now derive a lower bound for n from (2.4a):

$$(1 - p_0)^n \leq \alpha \implies n \log(1 - p_0) \leq \log \alpha \implies n \geq \frac{\log \alpha}{\log(1 - p_0)} \tag{2.5}$$

The minimum sample size for $p_1 = 0$ and $p_0 < 1$ is thus $n = \lceil \log \alpha / \log(1 - p_0) \rceil$. Note that n is independent of β, which makes perfect sense because the given sampling plan will always guarantee a zero

thresholds		**optimal single sampling plan**	
$p_1 = 0$	$p_0 = 1$	$n = 1$	$c = 0$
$p_1 = 0$	$p_0 < 1$	$n = \left\lceil \dfrac{\log \alpha}{\log(1 - p_0)} \right\rceil$	$c = 0$
$p_1 > 0$	$p_0 = 1$	$n = \left\lceil \dfrac{\log \beta}{\log p_1} \right\rceil$	$c = n - 1$

Table 2.2: Optimal single sampling plans for different choices of p_1 and p_0.

probability of accepting H_0 when H_1 is true.

The second special case is essentially a mirror image of the first: $p_1 > 0$ and $p_0 = 1$. We can see from (2.3) that $F(c; n, 1) = 0$ so long as $c < n$, meaning that (2.4a) is trivial to satisfy. Choosing c as large as possible makes it easier to satisfy (2.4b), so we choose $c = n - 1$. This gives us $1 - F(c; n, p_1) = p_1^n$ and we can now derive a lower bound for n from (2.4b):

$$p_1^n \leq \beta \implies n \log p_1 \leq \log \beta \implies n \geq \frac{\log \beta}{\log p_1} \tag{2.6}$$

The optimal sample size is therefore $n = \lceil \log \beta / \log p_1 \rceil$ for $p_1 > 0$ and $p_0 = 1$. As in the previous case, n depends only on one of the error bounds: the probability of accepting H_1 when H_0 holds is always zero. Table 2.2 summarizes the two cases when we can express n exactly, and also shows the optimal single sampling plan for the degenerate case when the indifference region is $(0, 1)$.

Example 2.2 ("five nines"). Imagine that we are testing a critical system, and we want to be almost certain that the system almost never fails. Let $p_0 = 1$, $p_1 = 1 - 10^{-5} = 0.99999$ and $\beta = 10^{-10}$. Table 2.2 gives us the single sampling plan $\langle 2302574, 2302573 \rangle$ for the specified parameters of the test. This implies that to guarantee a probability of at most 10^{-10} of accepting the system as functional when its failure probability is at least 10^{-5}, we should make over two million observations and accept the system only if we observe no failures.

We can derive an approximation formula for n when $p_1 > 0$ and $p_0 < 1$. A binomial distribution $B(n, p)$ has mean np and variance $np(1 - p)$. Let $Y = \left(\sum_{i=1}^{n} X_i - np\right) / \sqrt{np(1 - p)}$, where each X_i is a Bernoulli variate with parameter p as before. Then Y is approximately normal with mean 0 and variance 1 for large n, as first shown by De Moivre (1738).[2] In other words, $\Pr[Y \leq x] \approx \Phi(x)$, with $\Phi(x)$ being the

[2] Pearson (1924) aids the modern statistician in understanding the contribution of De Moivre.

standard normal cumulative distribution function given by (2.1). We accept hypothesis H_1 if $\sum_{i=1}^{n} x_i \le c$, for some constant c, so the probability of accepting H_1 is approximately $\Phi\big((c - np)/\sqrt{np(1-p)}\big)$. The optimal single sampling plan should accept H_1 with probability α if $p = p_0$ and probability $1-\beta$ if $p = p_1$. Using the inverse of $\Phi(x)$ and the fact that $\Phi(x) = 1 - \Phi(-x)$, we can express these constraints as follows:

$$\frac{c - np_0}{\sqrt{np_0(1-p_0)}} = \Phi^{-1}(\alpha) \tag{2.7a}$$

$$-\frac{c - np_1}{\sqrt{np_1(1-p_1)}} = \Phi^{-1}(\beta) \tag{2.7b}$$

By adding (2.7a) and (2.7b), we can derive an approximation formula for n:

$$\begin{aligned}(c - np_0) - (c - np_1) &= \Phi^{-1}(\alpha)\sqrt{np_0(1-p_0)} + \Phi^{-1}(\beta)\sqrt{np_1(1-p_1)} \\ \implies \sqrt{n}(p_1 - p_0) &= \Phi^{-1}(\alpha)\sqrt{p_0(1-p_0)} + \Phi^{-1}(\beta)\sqrt{p_1(1-p_1)} \\ \implies n &= \frac{\big(\Phi^{-1}(\alpha)\sqrt{p_0(1-p_0)} + \Phi^{-1}(\beta)\sqrt{p_1(1-p_1)}\big)^2}{(p_0 - p_1)^2}\end{aligned} \tag{2.8}$$

Thus, the sample size for a single sampling plan is approximately inversely proportional to the squared width of the indifference region. The presence of the factors $\sqrt{p_i(1-p_i)}$ in the numerator indicates that the sample size also depends on the placement of the indifference region. For a fixed width, the sample size is largest if the indifference region is centered around $p = 1/2$, and it decreases if the indifference region is shifted towards $p = 0$ or $p = 1$.

To get an idea of how the sample size depends on α and β, we can use the following approximation formula for the inverse normal cumulative distribution function with $\eta = \sqrt{-\log \alpha^2}$ (Hastings 1955, p. 191):

$$\Phi^{-1}(\alpha) \approx \tilde{\Phi}^{-1}(\alpha) = -\eta + \frac{a_0 + a_1\eta}{1 + b_1\eta + b_2\eta^2}\ , \qquad \big|\Phi^{-1}(\alpha) - \tilde{\Phi}^{-1}(\alpha)\big| < 3 \cdot 10^{-3} \tag{2.9}$$

$$a_0 = 2.30753 \quad b_1 = 0.99229$$

$$a_1 = 0.27061 \quad b_2 = 0.04481$$

This means that n is roughly proportional to the logarithm of α and β. Consequently, decreasing α or β tends to be less costly than narrowing the indifference region.

Example 2.3. For probability thresholds $p_0 = 0.505$ and $p_1 = 0.495$, and error bounds $\alpha = \beta = 10^{-2}$, the approximation formulae (2.8) and (2.9) give us $n \approx 54174$. The true value for n, computed by Algorithm 2.1, is 54117. If we keep the same error bounds, but shift the indifference region by setting $p_0 = 0.905$ and $p_1 = 0.895$, we get 19490 as the approximate sample size and 19481 as the exact.

```
SIMPLE-SEQUENTIAL-TEST(p_0, p_1, α, β)
  ⟨n, c⟩ ⇐ SINGLE-SAMPLING-PLAN(p_0, p_1, α, β)
  m ⇐ 0, d_m ⇐ 0
  while d_m ≤ c ∧ d_m + n − m > c do
    m ⇐ m + 1
    d_m ⇐ d_{m−1} + x_m
  if d_m > c then
    return H_0
  else
    return H_1
```

Algorithm 2.2: Sequential acceptance sampling procedure based on a single sampling plan.

2.2.3 Sequential Acceptance Sampling

The sample size for a single sampling plan is fixed and therefore independent of the actual observations made. It is often possible, however, to reduce the expected number of observations required to achieve a desired test strength by taking the observations into account as they are made.

Sequential Modification of Single Sampling Plan

If we use a single sampling plan $\langle n, c\rangle$ and the sum of the first m observations ($m < n$) is already greater than c, then we can accept H_0 without making further observations. Conversely, if the sum of the first m observations is d_m, and $d_m + n - m \leq c$ so that regardless of the outcome of the remaining $n - m$ observations we already know that the sum of n observations will not exceed c, then we can safely accept H_1 after making only m observations. The modified test procedure, summarized in Algorithm 2.2, is a simple example of a *sequential* sampling plan: after each observation, we decide whether sufficient information is available to accept either of the two hypotheses or additional observations are required.

The Sequential Probability Ratio Test

The idea of reducing the expected sample size by taking observations into account as they are made was first explored by Dodge and Romig (1929), who constructed double sampling plans where a second sample is drawn only if the observations constituting the first sample do not give sufficient support for accepting a hypothesis. A general theory of sequential hypothesis testing was later developed in a seminal paper by Wald (1945), where the *sequential probability ratio test* is defined. This test is provably optimal in the sense that it minimizes the expected sample size if $p = p_0$ or $p = p_1$ (Wald and Wolfowitz 1948), and the expected savings in the number of required observations compared to a single sampling plan is often substantial even if we use the sequential modification of the latter.

The sequential probability ratio test is carried out as follows. At the mth stage of the test, i.e. after making m observations $x_1, \ldots, x_m$, we calculate the quantity

(2.10) $$\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{\Pr[X_i = x_i | p = p_1]}{\Pr[X_i = x_i | p = p_0]} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}} \ ,$$

where $d_m = \sum_{i=1}^{m} x_i$. The quantity p_{jm} is simply the probability of the observation sequence $x_1, \ldots, x_m$, given that $\Pr[X_i = 1] = p_j$. This makes the computed quantity a ratio of two probabilities, hence the phrase *probability ratio* in the name of the test. Hypothesis H_0 is accepted if

(2.11) $$\frac{p_{1m}}{p_{0m}} \leq B \ ,$$

and hypothesis H_1 is accepted if

(2.12) $$\frac{p_{1m}}{p_{0m}} \geq A \ .$$

Otherwise, additional observations are made until either (2.11) or (2.12) is satisfied. A and B, with $A > B$, are chosen so that the probability is at most α of accepting H_1 when H_0 holds, and at most β of accepting H_0 when H_1 holds.

Finding A and B that gives strength $\langle \alpha, \beta \rangle$ is non-trivial. In practice we choose $A = (1-\beta)/\alpha$ and $B = \beta/(1-\alpha)$, which results in a test that very closely matches the prescribed strength. Let the actual strength of this test be $\langle \alpha', \beta' \rangle$. Wald (1945, p. 131) shows that the following inequalities hold:

(2.13) $$\alpha' \leq \frac{\alpha}{1-\beta}$$

(2.14) $$\beta' \leq \frac{\beta}{1-\alpha}$$

This means that if α and β are small, which typically is the case in practical applications, then α' and β' can only narrowly exceed the target values. Wald (1945, p. 132) also shows that $\alpha' + \beta' \leq \alpha + \beta$, so at least one of the inequalities $\alpha' \leq \alpha$ and $\beta' \leq \beta$ must hold, and in practice we often find that both inequalities hold.

Example 2.4. Let $p_0 = 0.5$, $p_1 = 0.3$, $\alpha = 0.2$ and $\beta = 0.1$ as in Example 2.1. If we use $A = (1-\beta)/\alpha$ and $B = \beta/(1-\alpha)$, then we are guaranteed that $\alpha' \leq 0.2/0.9 \approx 0.222$ and $\beta' \leq 0.1/0.8 = 0.125$ by the inequalities (2.13) and (2.14). Through computer simulation we obtain the estimates $\alpha' \approx 0.175 < \alpha$ and $\beta' \approx 0.082 < \beta$, so the strength of the test is in reality better than $\langle \alpha, \beta \rangle$.

If $p_0 = 1$ or $p_1 = 0$, then the sequential probability ratio test is equivalent to the test procedure encoded by Algorithm 2.2, provided that we choose $A = \alpha^{-1}$ and $B = \beta$. For $p_0 = 1$ and $x_i = 1$ for all i up to and including m, the probability ratio (2.10) equals p_1^m. We therefore accept H_0 if $p_1^m \leq \beta$, which is identical to the condition in (2.6). If, on the other hand, we observe a single zero before condition (2.11) is satisfied, the probability ratio becomes ∞ and we immediately accept H_1, corresponding to choosing $c = n - 1$ for a single sampling plan. For $p_1 = 0$, the probability ratio equals $(1 - p_0)^{-m}$ if the first m observations are zeros. We accept H_1 if $(1 - p_0)^{-m} \geq \alpha^{-1}$, which is equivalent to the condition in (2.5). In this case, we accept H_0 if we observe a single one before condition (2.12) is satisfied, corresponding to $c = 0$ for a single sampling plan. Anderson and Friedman (1960) call sampling plans of this kind *curtailed single sampling plans* and they prove that such plans are *strongly optimal*. This means that any other sampling plan with at least the same strength *always* requires at least as many observations for all values of p. In general, as mentioned above, the sequential probability ratio test only guarantees *expected* optimality for $p \in \{p_0, p_1\}$.

When implementing the sequential probability ratio test, it is typically computationally more practical to work with the logarithm of p_{1m}/p_{0m}. At stage m, we therefore compute

$$f_m = \log \frac{p_{1m}}{p_{0m}} = d_m \log \frac{p_1}{p_0} + (m - d_m) \log \frac{1 - p_1}{1 - p_0} \ .$$

We accept H_0 if $f_m \leq \log \frac{\beta}{1-\alpha}$, accept H_1 if $f_m \geq \log \frac{1-\beta}{\alpha}$, and make at least one more observation otherwise. Pseudocode for the sequential probability ratio test is given as Algorithm 2.3.

Geometric Interpretation of Sequential Tests

To gain a better understanding of how sequential tests work, it is intuitively appealing to give a geometric interpretation of such tests. At stage m of a sequential test, we summarize the m observations made so far with the statistic d_m. The pair $\langle m, d_m \rangle$ can be considered as the current state of the test, where m and d_m are non-negative integers with $d_m \leq m$. The two-dimensional space $S = \{\langle m, d_m \rangle \in \mathbb{Z}^* \times \mathbb{Z}^* \mid d_m \leq m\}$ constitutes the possible states of a sequential test. Any given sequential test procedure subdivides the space S into three mutually exclusive regions R_0, R_1, and R_c ("continue"). The test is terminated the first time the state of the test enters either R_0 or R_1. At the entrance of the subregion R_i, hypothesis H_i is accepted.

SPRT$(p_0, p_1, \alpha, \beta)$
  if $p_0 = 1 \vee p_1 = 0$ **then**
    return SIMPLE-SEQUENTIAL-TEST$(p_0, p_1, \alpha, \beta)$
  else
    $m \Leftarrow 0,\ f_m \Leftarrow 0$
    while $\log \frac{\beta}{1-\alpha} < f_m < \log \frac{1-\beta}{\alpha}$ **do**
      $m \Leftarrow m + 1$
      $f_m \Leftarrow f_{m-1} + x_m \log \frac{p_1}{p_0} + (1 - x_m) \log \frac{1-p_1}{1-p_0}$
    if $f_m \leq \log \frac{\beta}{1-\alpha}$ **then**
      return H_0
    else
      return H_1

Algorithm 2.3: Procedure implementing the sequential probability ratio test.

The subregion R_c represents states where additional observations are required. This region always contains the point $\langle 0, 0\rangle$, meaning that a sequential test starts in this region.

For a sequential test derived from a single sampling plan $\langle n, c\rangle$, we never make more than n observations, so the state space of such a test is $S' = \{\langle m, d_m\rangle \in S \mid m \leq n\}$. We accept H_0 if $d_m > c$. Thus, we set $R_0 = \{\langle m, d_m\rangle \in S' \mid d_m > c\}$. For the same test, we accept H_1 if $d_m \leq m + c - n$, so $R_1 = \{\langle m, d_m\rangle \in S' \mid d_m \leq m + c - n\}$. Figure 2.11 displays the regions graphically for $p_0 = 0.5$, $p_1 = 0.3$, $\alpha = 0.2$, and $\beta = 0.1$ (i.e. $n = 30$ and $c = 12$ as stated in Example 2.1). The shaded regions represent unreachable states ($d_m > m$ and $m > n$). The line $d_m = c$ that defines the boundary between R_c and R_0 is called the *acceptance line*, while the line $d_m = m + c - n$ defining the boundary between R_c and R_1 is called the *rejection line*. The test can be carried out graphically by plotting a curve representing the outcome of the observations. The solid curve in Figure 2.11 represents the observations $x_i = 1$ for $i \in \{1, 3, 4, 6, 7, 8\}$ and $x_i = 0$ for $i \in \{2, 5\}$. Hypothesis H_0 is accepted the moment this curve intersects the acceptance line, and H_1 is accepted (H_0 is rejected) the moment the curve intersects the rejection line.

In contrast, the sequential probability ratio test terminates if $f_m \leq \log \frac{\beta}{1-\alpha}$ (accept H_0) or $f_m \geq \log \frac{1-\beta}{\alpha}$ (accept H_1). We can write these termination criteria as $d_m \geq h_0 + ms$ and $d_m \leq h_1 + ms$ respectively, where h_0, h_1, and s are given by the following expressions:

$$h_0 = \frac{\log \dfrac{\beta}{1-\alpha}}{\log \dfrac{p_1(1-p_0)}{p_0(1-p_1)}} \qquad h_1 = \frac{\log \dfrac{1-\beta}{\alpha}}{\log \dfrac{p_1(1-p_0)}{p_0(1-p_1)}} \qquad s = \frac{\log \dfrac{1-p_0}{1-p_1}}{\log \dfrac{p_1(1-p_0)}{p_0(1-p_1)}} \tag{2.15}$$

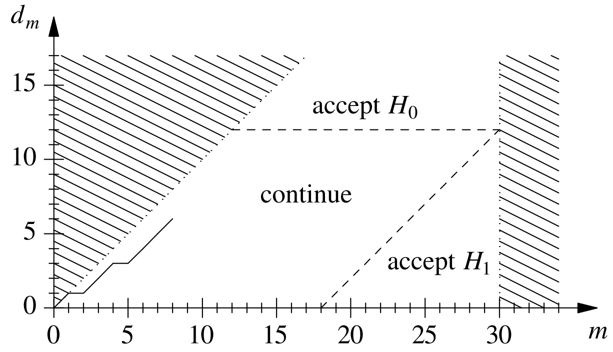


Figure 2.11: Graphical representation of a sequential single sampling plan for $p_0 = 0.5$, $p_1 = 0.3$, $\alpha = 0.2$, and $\beta = 0.1$ ($n = 30$ and $c = 12$).

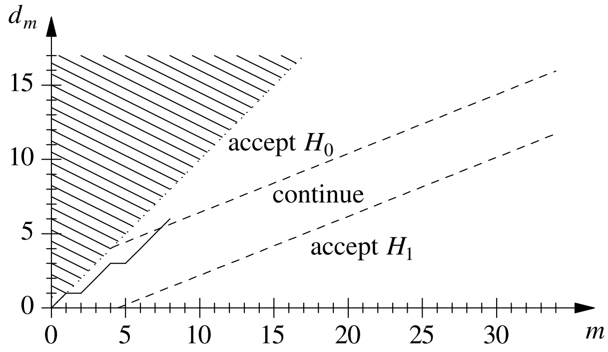


Figure 2.12: Graphical representation of the sequential probability ratio test for $p_0 = 0.5$, $p_1 = 0.3$, $\alpha = 0.2$, and $\beta = 0.1$.

We can therefore define the acceptance region $R_0 = \{\langle m, d_m\rangle \in S \mid d_m \geq h_0 + ms\}$ for the sequential probability ratio test. The line $d_m = h_0 + ms$ is the acceptance line for the test. Similarly, $R_1 = \{\langle m, d_m\rangle \in S \mid d_m \leq h_1 + ms\}$ making $d_m = h_1 + ms$ the rejection line for the test.

Figure 2.12 shows a graphical representation of the sequential probability ratio test for the same parameters that were used in Figure 2.11. The solid curve represents the same observation sequence as was plotted in Figure 2.11. Note that the curve intersects the acceptance line with the eighth observation, so we accept the hypothesis $H_0 : p \geq 0.5$ at this point if we use the sequential probability ratio test. The same observation sequence does not result in acceptance in Figure 2.11, which indicates that we can reduce the expected number of observations by using the sequential probability ratio test. The acceptance and rejection lines are parallel with common slope s. Consequently, the region R_c is unbounded and there is no upper bound on the number of observations that the test will require before terminating. However, the probability is equal to one that the sequential probability ratio test will eventually terminate (Wald 1945, p. 128), although the sample size may vary greatly.

Expected Sample Sizes

The sample size for a sequential acceptance sampling test is a random variable, meaning that the required number of observations can vary from one use of such a test to another. Furthermore, the expected sample size typically depends on the unknown parameter p, so we cannot report a single value as was the case for acceptance sampling with fixed-size samples. The expected sample size varies with the distance of p from the indifference region (p_1, p_0). It tends to be largest when p is close to the center of the indifference region, and decreases the further away p is from the indifference region.

First, consider the sequential variation of a single sampling plan $\langle n, c\rangle$. The test terminates at stage m if $d_m > c$ (accept H_0) or $d_m \leq m + c - n$ (accept H_1). The probability of the test terminating at stage m by accepting H_0 is equal to the probability of observing exactly c ones in the first $m - 1$ observations and then an additional one. This probability can be expressed as $p \cdot f(c; m - 1, p)$, where p is the probability of observing a one and $f(c; n, p)$ is the probability density function for $B(n, p)$. Note that we could not have accepted H_1 prior to stage m under these conditions, because we accept H_1 only if the remaining observations cannot lead to acceptance of H_0. The test terminates at stage m by accepting H_1 if we observe exactly $m + c - n$ ones in the first $m - 1$ observations followed by a zero, which occurs with probability $(1 - p)f(m + c - n; m - 1, p)$. The expected sample size E_p as a function of p can therefore be expressed as follows:

$$E_p = \sum_{m=c+1}^{n} m \cdot p \cdot f(c; m-1, p) + \sum_{m=n-c}^{n} m \cdot (1-p) \cdot f(m+c-n; m-1, p) \tag{2.16}$$

Naturally, E_p can never exceed n, is exactly $n - c$ if $p = 0$, and is exactly $c + 1$ if $p = 1$.

The expected sample size for the sequential probability ratio test is harder to determine. Wald (1945, p. 164) provides

$$\tilde{E}_p = \frac{L_p \log \dfrac{\beta}{1-\alpha} + (1 - L_p) \log \dfrac{1-\beta}{\alpha}}{p \log \dfrac{p_1}{p_0} + (1-p) \log \dfrac{1-p_1}{1-p_0}} \tag{2.17}$$

as a good approximation of E_p when p_1 is not far from p_0, which is typically the case in practice. The quantity L_p is the probability of accepting H_0 when $\Pr[X_i = 1] = p$. Wald provides an approximation formula for L_p as well, but the formula is not suited for computing an approximation of L_p for an arbitrary p. Approximating E_p for an arbitrary p is therefore non-trivial, but we can provide explicit formulae for a few cases of special interest, as shown in Table 2.3.[3] The expected sample size increases from 0 to p_1 and decreases from p_0 to 1. In the indifference region (p_1, p_0), the sample size increases from p_1 to some point p' and decreases from p' to p_0. The point p' is generally equal to s or at least very near s, where s is the common slope given in (2.15) of the acceptance and rejection lines (Wald 1947, p. 101).

[3] The approximation formulae for $p = 0$ and $p = 1$ differ from those derived by Wald (1947, pp. 99–100). This is because we assume $p_0 > p_1$, while Wald assumes the opposite.

p	$\tilde{L}_p$	$\tilde{E}_p$
0	0	$\dfrac{\log \dfrac{1-\beta}{\alpha}}{\log \dfrac{1-p_1}{1-p_0}}$
p_1	β	$\dfrac{\beta \log \dfrac{\beta}{1-\alpha} + (1-\beta)\log \dfrac{1-\beta}{\alpha}}{p_1 \log \dfrac{p_1}{p_0} + (1-p_1)\log \dfrac{1-p_1}{1-p_0}}$
s	$\dfrac{\log \dfrac{1-\beta}{\alpha}}{\log \dfrac{1-\beta}{\alpha} - \log \dfrac{\beta}{1-\alpha}}$	$\dfrac{-\log \dfrac{\beta}{1-\alpha} \log \dfrac{1-\beta}{\alpha}}{\log \dfrac{p_1}{p_0} \log \dfrac{1-p_0}{1-p_1}}$
p_0	$1-\alpha$	$\dfrac{(1-\alpha)\log \dfrac{\beta}{1-\alpha} + \alpha \log \dfrac{1-\beta}{\alpha}}{p_0 \log \dfrac{p_1}{p_0} + (1-p_0)\log \dfrac{1-p_1}{1-p_0}}$
1	1	$\dfrac{\log \dfrac{\beta}{1-\alpha}}{\log \dfrac{p_1}{p_0}}$

Table 2.3: Approximate expected sample size for the sequential probability ratio test.

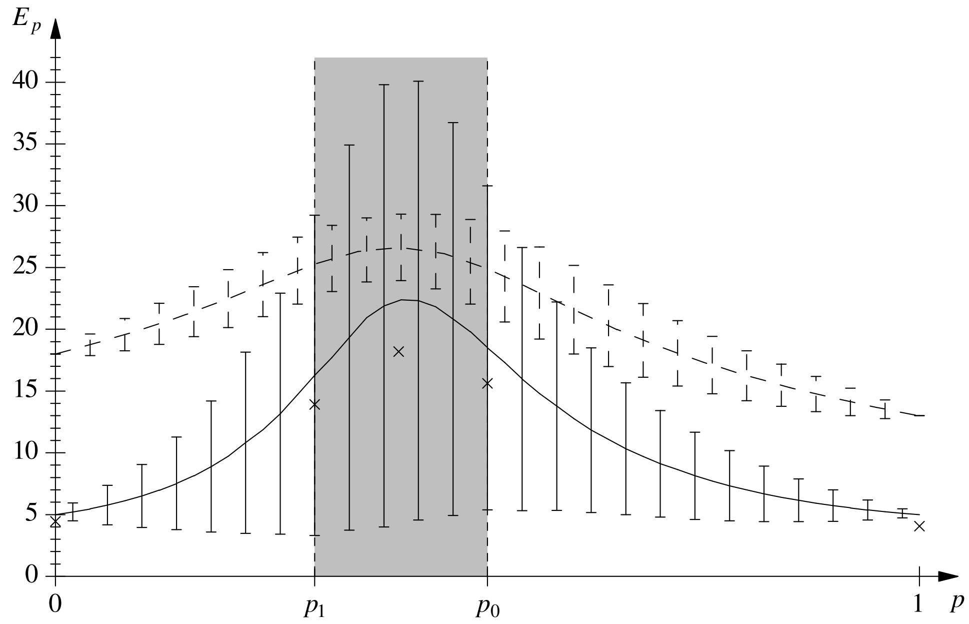


Figure 2.13: Expected sample size for a sequential single sampling plan (dashed curve) and the sequential probability ratio test (solid curve) with $p_0 = 0.5$, $p_1 = 0.3$, $\alpha = 0.2$, and $\beta = 0.1$. The error bars extend a standard deviation in each direction from the curves. The crosses mark the approximate expected sample size for the cases listed in Table 2.3. The indifference region is fairly wide in this case, resulting in a relatively large approximation error. For a narrower indifference region, the approximation error is generally much less noticeable.

Figure 2.13 plots the expected sample size as a function of the true probability p for the sequential single sampling plan and the sequential probability ratio test with $p_0 = 0.5$, $p_1 = 0.3$, $\alpha = 0.2$, and $\beta = 0.1$. The curve for the former was computed using (2.16), while the curve for the latter was generated using computer simulation. We see that the sequential probability ratio test has a lower average than the sequential test derived from a single sampling plan, but that the variance is much larger when p is in, or close to, the indifference region. As we will see next, however, the sequential probability ratio test does not always have a lower expected sample size than a sequential single sampling plan with the same strength.

Optimality of Sequential Tests

For the particular choice of parameters that was used to produce Figure 2.13, the sequential probability ratio test has a lower expected sample size than an optimal single sampling plan for all values of p. In general, however, this is not guaranteed to be the case. While the sequential probability ratio test minimizes the expected sample size at p_0 and p_1 simultaneously, there may very well exist alternative tests that achieve a lower expected sample size for other values of p, in particular for $p \in (p_1, p_0)$.

Example 2.5. For $p_0 = 0.5$, $p_1 = 0.3$, and $\alpha = \beta = 10^{-4}$, the optimal single sampling plan requires exactly 326 observations. In contrast, the expected sample size for the sequential probability ratio test is 510 at $p = s$, which is a 56 percent increase in the expected sample size compared to a single sampling plan.

It is easy to see that the expected sample size at $p = s$ for the sequential probability ratio test can be larger than the fixed sample size of a single sampling plan if α and β are sufficiently small. Consider the case when $\alpha = \beta$. From the approximation formula for $p = s$ in Table 2.3, it follows that the numerator of $\tilde{E}_s$ is equal to $\big(\log(\alpha^{-1} - 1)\big)^2$, which means that E_s is approximately proportional to the square of $\log \alpha$. From (2.8) and (2.9), on the other hand, it follows that the sample size for a single sampling plan is approximately proportional to $\log \alpha$. As α approaches zero, $(\log \alpha)^2$ grows faster than $\log \alpha$, which helps explain the fact that E_s can be larger for the sequential probability ratio test than for a single sampling plan.

Kiefer and Weiss (1957) suggest minimizing the expected sample size at a third point p_2, instead of at p_0 and p_1, by using a *generalized* sequential probability ratio test.[4] If p_2 is chosen with care, the resulting test minimizes the maximum expected sample size. Weiss (1962) derives such a test for the symmetric case with $p_0 = \frac{1}{2} + \delta$ and $p_1 = \frac{1}{2} - \delta$, while Freeman and Weiss (1964) consider approximate solutions for the general case. The test is designed to minimize

$$b_0 \Pr[H_1 \text{ accepted} | p = p_0] + b_1 \Pr[H_0 \text{ accepted} | p = p_1] + b_2 E_{p_2} \ ,$$

where b_0, b_1, and b_2 are user-specified positive constants such that $b_0 + b_1 + b_2 = 1$. For some choice of these constants, the resulting test has strength $\langle \alpha, \beta \rangle$, although the exact relationship is unknown (Freeman and Weiss 1964, p. 69). While this surely is an interesting alternative problem formulation, we will not explore it further in this thesis because it represents a departure from the model where the user specifies the desired strength of the test. Schwarz (1962) and Lai (1988) consider yet another problem formulation where the objective is to minimize the expected cost subject to a cost c per observation and a unit cost for accepting a false hypothesis. We refer the interested reader to Lai (2001) for a more detailed account of the developments in the field of sequential hypothesis testing since the ground-breaking work of Wald.

[4]The condition for making an additional observation at stage m when using a generalized sequential probability ratio test is $B_m < p_{1m}/p_{0m} < A_m$ (Weiss 1953). The test is a regular sequential probability ratio test if $A_m = A$ and $B_m = B$ for all m.

2.3 Stochastic Discrete Event Systems

This section formally defines the class of systems for which we develop verification and planning algorithms in later chapters. We rely heavily on the notion of a *stochastic process*, which is any process that evolves over time, and whose evolution we can follow and predict in terms of probability (Doob 1942, 1953). At any point in time, a stochastic process is said to occupy some state. If we attempt to observe the state of a stochastic process at a specific time, the outcome of such an observation is governed by some probability law. Mathematically, we define a stochastic process as a family of random variables.

Definition 2.1 (Stochastic Process). Let S and T be two sets. A *stochastic process* is a family of random variables $\mathcal{X} = \{X_t \mid t \in T\}$, with each random variable X_t having range S.

The index set T in Definition 2.1 represents time and is typically the set of non-negative integers, $\mathbb{Z}^*$, for discrete-time stochastic processes and the set of non-negative real numbers, $[0, \infty)$, for continuous-time stochastic processes. We will generally assume that T is such that if $t \in T$ and $t' \in T$ for $t' \geq t$, then $t' - t \in T$. The set S represents the states that the stochastic process can occupy, and this can be an infinite, or even uncountable, set.

The definition of a stochastic process as a family of random variables is quite general and includes systems with both continuous and discrete dynamics. We will focus our attention on a limited, but important, class of stochastic processes: *stochastic discrete event systems*. This class includes any stochastic process that can be thought of as occupying a single state for a duration of time before an *event* causes an instantaneous state transition to occur. The canonical example of such a process is a queuing system, with the state being the number of items currently in the queue. Thus, the state space S is $\{0, 1, \ldots, n\}$ if the queue has finite capacity n and $\mathbb{Z}^*$ if it has infinite capacity. The state changes at the occurrence of an event representing the arrival or departure of an item. We call this a *discrete event* system because the state change is discrete rather than continuous and is caused by the triggering of an event.

2.3.1 Trajectories

A random variable $X_t \in \mathcal{X}$ represents the chance experiment of observing the stochastic process $\mathcal{X}$ at time t. If we record our observations at consecutive time points for all $t \in T$, then we have a *trajectory*, or *sample path*, for $\mathcal{X}$. Our work in probabilistic model checking is centered around the verification of temporal logic

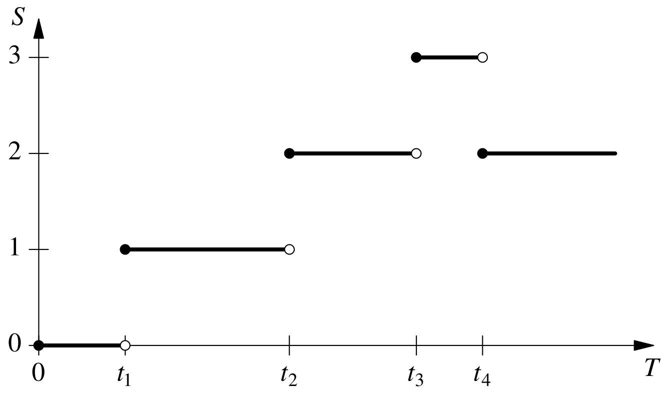


Figure 2.14: A trajectory for a simple queuing system with arrival events occurring at t_1, t_2 and t_3 and a departure event occurring at t_4. The state of the system represents the number of items in the queue.

formulae over trajectories for stochastic discrete event systems. The terminology and notation introduced here is used extensively in later chapters.

Definition 2.2 (Trajectory). A *trajectory* for a stochastic process $\mathcal{X}$ is any set of observations $\{x_t \in S \mid t \in T\}$ of the random variables $X_t \in \mathcal{X}$.

The trajectory of a stochastic discrete event system is *piecewise constant* and can therefore be represented as a sequence $\sigma = \{\langle s_0, t_0 \rangle, \langle s_1, t_1 \rangle, \ldots\}$, with $s_i \in S$ and $t_i \in T \setminus \{0\}$. Zero is excluded to ensure that only a single state can be occupied at any point in time. Figure 2.14 plots part of a trajectory for a simple queuing system. Let

$$T_i = \begin{cases} 0 & \text{if } i = 0 \\ \sum_{j=0}^{i-1} t_j & \text{if } i > 0 \end{cases}, \tag{2.18}$$

i.e. T_i is the time at which state s_i is entered and t_i is the duration of time for which the process remains in s_i before an event triggers a transition to state s_{i+1}. A trajectory σ is then a set of observations of $\mathcal{X}$ with $x_t = s_i$ for $T_i \leq t < T_i + t_i$. According to this definition, trajectories of stochastic discrete event systems are *right-continuous*. A finite trajectory is a sequence $\sigma = \{\langle s_0, t_0 \rangle, \ldots, \langle s_n, \infty \rangle\}$ where s_n is an *absorbing* state, meaning that no events can occur in s_n and that $x_t = s_n$ for all $t \geq T_n$.

An infinite trajectory is *convergent* if $T_\infty < \infty$. In this case, x_t is not well-defined for all $t \in T$. For a trajectory to be convergent, however, an infinite sequence of events must occur in a finite amount of time, which is unrealistic for any physical system. Hoel et al. (1972) use the term *explosive* to describe processes for which such sequences can occur with non-zero probability. It is common to assume *time divergence* for infinite trajectories of real-time systems (cf. Alur and Dill 1994), i.e. that the systems are non-explosive, and most finite-state systems satisfy this property by default.

2.3.2 Measurable Stochastic Discrete Event Systems

Of utmost importance to probabilistic model checking is the definition of a *probability measure* over sets of trajectories for a system. The set of trajectories must be *measurable*. Formally, a *measurable space* is a set Ω with a σ-algebra $\mathcal{F}_\Omega$ of subsets of Ω (Halmos 1950). A *probability space* is a measurable space $\langle \Omega, \mathcal{F}_\Omega \rangle$ and a probability measure μ. When we say that a set Ω must be measurable, we really mean that there must be a σ-algebra for the set. The elements of this σ-algebra are the measurable subsets of Ω.

For stochastic discrete event systems, the elements of the σ-algebra are sets of trajectories with common *prefix*. A prefix of a trajectory $\sigma = \{\langle s_0, t_0 \rangle, \langle s_1, t_1 \rangle, \ldots\}$ is a sequence $\sigma_{\leq \tau} = \{\langle s'_0, t'_0 \rangle, \ldots, \langle s'_k, t'_k \rangle\}$, with $s'_i = s_i$ for all $i \leq k$, $\sum_{i=0}^{k} t'_i = \tau$, $t'_i = t_i$ for all $i < k$, and $t'_k < t_k$. Let $Path(\sigma_{\leq \tau})$ denote the set of trajectories with common prefix $\sigma_{\leq \tau}$. This set must be measurable, and we assume that a probability measure μ over the set of trajectories with common prefix exists. For our work on probabilistic model checking, we assume only that we can generate sample trajectory prefixes distributed according to μ.

A probability measure μ over sets of trajectories with common prefix can be defined for virtually all systems of practical interest, although the precise definition thereof will of course depend on the specific probability structure of the stochastic discrete event system being studied. In general, a stochastic discrete event system is measurable if the sets S and T are measurable. We can show this by defining a σ-algebra over the set of trajectories with common prefix $\sigma_{\leq \tau} = \{\langle s_0, t_0 \rangle, \ldots, \langle s_k, t_k \rangle\}$, denoted $Path(\sigma_{\leq \tau})$, as follows. Let $\mathcal{F}_S$ be a σ-algebra over the state space S, and let $\mathcal{F}_T$ be a σ-algebra over the index set T of the stochastic process. Such σ-algebras exist if S and T are measurable sets, which by assumption they are. Then $C(\sigma_{\leq \tau}, I_k, S_{k+1}, \ldots, I_{n-1}, S_n)$, with $S_i \in \mathcal{F}_S$ and $I_i \in \mathcal{F}_T$, denotes the set of trajectories $\sigma = \{\langle s'_0, t'_0 \rangle, \langle s'_1, t'_1 \rangle, \ldots\}$ such that $s'_i = s_i$ for $i \leq k$, $s'_i \in S_i$ for $k < i \leq n$, $t'_i = t_i$ for $i < k$, $t'_k > t_k$, and $t'_i \in I_i$ for $k \leq i < n$. In other words, $C(\sigma_{\leq \tau}, I_k, S_{k+1}, \ldots, I_{n-1}, S_n)$ is a subset of $Path(\sigma_{\leq \tau})$. The sets $C(\sigma_{\leq \tau}, I_k, S_{k+1}, \ldots, I_{n-1}, S_n)$ are the elements of a σ-algebra over the set $Path(\sigma_{\leq \tau})$ with set operations applied element-wise, for example $C(\sigma_{\leq \tau}, I_k, S_{k+1}, \ldots, I_{n-1}, S_n) \cup C(\sigma_{\leq \tau}, I'_k, S'_{k+1}, \ldots, I'_{n-1}, S'_n) = C(\sigma_{\leq \tau}, I_k \cup I'_k, S_{k+1} \cup S'_{k+1}, \ldots, I_{n-1} \cup I'_{n-1}, S_n \cup S'_n)$.

2.3.3 Structured Stochastic Discrete Event Systems

So far, we have defined stochastic discrete event systems in rather general terms as any stochastic process with piecewise constant trajectories. Most stochastic discrete event systems of interest have more structure than that. Any additional structure simplifies the specification of a stochastic discrete event system and can often be exploited in the analysis of such systems.

The probability measure on sets of trajectories for a stochastic discrete event system can be expressed using a holding time distribution with probability density function $h(\cdot;\sigma_{\leq\tau})$ and a next-state distribution $p(\cdot;\sigma_{\leq\tau},t)$. The probability measure for $C(\sigma_{\leq\tau},I_k,S_{k+1},\ldots,I_{n-1},S_n)$ can now be defined recursively as

$$\mu(C(\sigma_{\leq\tau},I_k,S_{k+1},\ldots,I_{n-1},S_n)) = \int_{I_k} h(t_k+t;\sigma_{\leq\tau}) \int_S p(s;\sigma_{\leq\tau},t)\mu(C(\sigma_{\leq\tau}\oplus\langle t,s\rangle,I_{k+1},S_{k+2},\ldots,I_{n-1},S_n)) \,, \tag{2.19}$$

where $\{\langle s_0,t_0\rangle,\ldots,\langle s_k,t_k\rangle\}\oplus\langle t,s\rangle = \{\langle s_0,t_0\rangle,\ldots,\langle s_k,t_k+t\rangle,\langle s,0\rangle\}$. The base case for the recursive definition is $\mu(C(\sigma_{\leq\tau}))=1$. This is a *factored* representation of the probability measure μ.

In addition to structure in the probability measure on sets of trajectories, we can also have structure in the state space. Instead of a flat state representation, it is often natural to describe the state of a system by using multiple state variables which leads to a factored state space. A factored representation of the state space S of a measurable stochastic discrete event system is a set of state variables SV and a value assignment function $V(s,x)$ providing the value of $x\in SV$ in state s. The domain of x is the set $D_x=\bigcup_{s\in S}V(s,x)$ of possible values that x can take on. A tuple $\langle S,T,\mu,SV,V\rangle$ represents a measurable stochastic discrete event system with a factored state space. Note that $|S|$ is at most $\prod_{x\in SV}|D_x|$, which is exponential in the number of state variables, but the actual size of S can of course be smaller than $\prod_{x\in SV}|D_x|$ if certain combinations of variable assignments do not correspond to an actual state $s\in S$.

We will now discuss a few common models of stochastic discrete event systems with specific structural properties. By making limiting assumptions regarding the shape of the probability density functions $h(\cdot;\sigma_{\leq\tau})$ and $p(\cdot;\sigma_{\leq\tau},t)$, we enable a succinct representation of μ. This is important for efficient generation of sample trajectories for stochastic discrete event systems, which is a large component of our statistical model checking algorithm. We include a brief description of Markov and semi-Markov processes. More detailed accounts on this topic are provided by, for example, Kolmogoroff (1931), Doob (1953), Bartlett (1966), Howard (1971a, 1971b), and Çinlar (1975).

Markov Processes

A stochastic discrete event system is a *time homogeneous Markov process* if the future behavior at any point in time depends only on the state at that point in time, and not in any way on how that state was reached. This implies that the probability measure on sets of trajectories satisfies the following property:

$$\mu(Path(\{\langle s_0, t_0\rangle, \ldots, \langle s_k, t_k\rangle\})) = \mu(Path(\{\langle s_k, 0\rangle\})) \tag{2.20}$$

Equation 2.20 is known as the *Markov property*, named after the Russian mathematician A. A. Markov who in the early 1900's systematically studied discrete-time stochastic processes satisfying this property. A Markov process is *time inhomogeneous* if the distribution over future trajectories depends on the time of observation, in addition to the current state.

For a factored representation of μ, condition (2.20) holds if and only if $h(t_k + t; \sigma_{\leq\tau}) = h(t; s_k)$ and $p(\cdot; \sigma_{\leq\tau}, t) = p(\cdot; s_k)$ for all trajectory prefixes $\sigma_{\leq\tau} = \{\langle s_0, t_0\rangle, \ldots, \langle s_k, t_k\rangle\}$. The first condition implies that $h(\cdot; \sigma_{\leq\tau})$ is a memoryless distribution. Thus, a discrete-time Markov process has geometric holding time distributions for each state, so the probability of remaining in state s for t more time units before a state transition occurs is $h(t; s) = q_s(1 - q_s)^{t-1}$ for some $q_s \in [0, 1]$. The dynamics of a discrete-time Markov process with state space S is fully specified with q_s and $p(\cdot; s)$ for each state $s \in S$. If S is countable, then the dynamics is captured by a state transition probability matrix P with elements

$$P_{ij} = \begin{cases} 1 - q_i(1 - p(i; i)) & \text{if } i = j \\ q_i p(j; i) & \text{if } i \neq j \end{cases},$$

where P_{ij} is the probability that the discrete-time Markov process occupies state j at time $t + 1$ when the process occupies state i at time t.

Example 2.6. Consider a simple queuing system, and let s_i, $i \geq 0$, denote the state with i items in the queue. Assume that the holding time in s_0 is geometrically distributed with parameter $q_0 = \frac{1}{5}$ and the holding time in all other states is geometrically distributed with parameter $q_i = \frac{2}{5}$. The expected holding time in s_0 is greater than in the other states because no departures can occur in s_0. Furthermore, assume that a state transition in s_i, for $i > 0$, is caused by a departure with probability $\frac{3}{4}$ and an arrival with probability $\frac{1}{4}$. The resulting discrete-time Markov process is depicted in Figure 2.15.

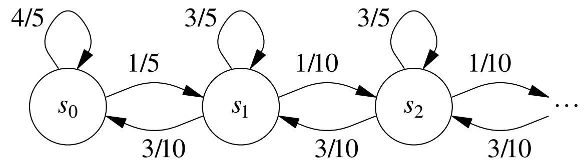


Figure 2.15: A discrete-time Markov process representing a queuing system. The arcs are labeled with the entries of the state transition probability matrix for the process.

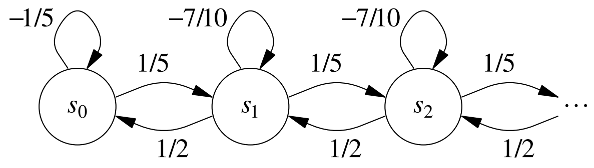


Figure 2.16: A continuous-time Markov process representing a queuing system. The arcs are labeled with the entries of the infinitesimal generator matrix for the process.

For continuous-time Markov processes, the holding time in state s is exponentially distributed: $h(t; s) = \lambda_s e^{-\lambda_s t}$. The parameter λ_s is the *exit rate* for state s. The probability that a state transition occurs in the next t' time units is $\int_0^{t'} \lambda_s e^{-\lambda_s t}\, dt = 1 - e^{-\lambda_s t'}$. The dynamics of a continuous-time Markov process with countable state space can be fully characterized by a matrix Q with elements

$$Q_{ij} = \begin{cases} -\lambda_i(1 - p(i; i)) & \text{if } i = j \\ \lambda_i p(j; i) & \text{if } i \neq j \end{cases} .$$

The matrix Q is typically referred to as the *infinitesimal generator* of a continuous-time Markov process (Puterman 1994, p. 561).

Example 2.7. Consider a queuing system similar to that in Example 2.6, but with time as a continuous quantity. The holding time for s_i is exponentially distributed with rate $\frac{1}{5}$ for $i = 0$ and $\frac{7}{10}$ for $i > 0$. In s_i, for $i > 0$, a state transition is caused by a departure with probability $\frac{5}{7}$ and by an arrival with probability $\frac{2}{7}$. Figure 2.16 shows the resulting continuous-time Markov process.

As was mentioned earlier, it is common to assume time divergence for infinite trajectories of stochastic discrete event systems, i.e. that the system is non-explosive. Obviously, any discrete-time Markov process is non-explosive because there is always at least a unit delay between state transitions. It can be shown that a sufficient condition for a continuous-time Markov process to be non-explosive is that there exists a constant c such that $\lambda_s \leq c$ for all $s \in S$ (cf. Baier et al. 2003, Prop. 1). As a direct consequence, all finite-state time homogeneous Markov processes are non-explosive. Not all infinite-state continuous-time Markov processes are non-explosive, however, as the following example illustrates.

Example 2.8. Consider the continuous-time Markov process depicted in Figure 2.17, with an infinite state space $S = \{s_0, s_1, \dots\}$ and exit rates $\lambda_i = 2^{2(i+1)}$. The exit rates for this Markov process rapidly increase

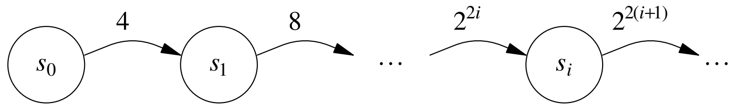


Figure 2.17: An explosive continuous-time Markov process.

with each state transition. Any trajectory with a holding time in the interval $(0, 2^{-(i+1)})$ in state s_i, for each $i \geq 0$, is convergent because the total time never exceeds 1. The probability measure for the set of all such trajectories is

$$\prod_{i=0}^{\infty}\left(1 - e^{-\lambda_i \cdot 2^{-(i+1)}}\right) = \prod_{i=1}^{\infty}\left(1 - e^{-2^i}\right) \ .$$

This infinite product converges to a value approximately equal to 0.849. Thus, the probability measure of the set of convergent trajectories is non-zero, which means that the Markov process is explosive.

In order to simulate the execution of a Markov process, we need to be able to sample from the next-state distribution of any state. If we are to simulate execution for an extended period of time, we need a long sequence of pseudorandom numbers. Unless we are careful in our choice of pseudorandom number generator, subtle correlations in pseudorandom number sequences may be a source of systematic error in the analysis of the simulation output (Ferrenberg et al. 1992). The Mersenne Twister (Matsumoto and Nishimura 1998), with its exceptionally long period, is thought to be a suitable pseudorandom number generator for simulation studies of stochastic processes.

Semi-Markov Processes

Not all phenomena in nature are accurately captured by memoryless distributions. The lifetime of a system component, for example, is often best modeled using a Weibull distribution (Nelson 1985). The Weibull distribution can be used to model increasing failure rates, for example representing increasing likelihood of failure due to wear, as well as decreasing failure rates.

A *semi-Markov process* is a stochastic process for which in order to accurately predict future behavior one may need to know not only the current state but also the amount of time spent in that state (although it is still inconsequential how the current state was reached). We can state the semi-Markov property as a constraint on the probability measure μ:

(2.21) $$\mu(Path(\{\langle s_0, t_0\rangle, \dots, \langle s_k, t_k\rangle\})) = \mu(Path(\{\langle s_k, t_k\rangle\}))$$

The probability measure over sets of trajectories for a semi-Markov process can be represented by a holding time distribution $h(\cdot; s_k, t_k)$ and a next-state distribution $p(\cdot; s_k, t)$. The probability of transitioning out of state s_k within t time units, provided that we have already been in s_k for t_k time units, is given by $\int_0^t h(t_k + x; s_k, t_k)$. Given that a state transition occurs after t time units in state s_k, the probability that the next state belongs to the set S' is $\int_{S'} p(s'; s_k, t)$.

Example 2.9. Consider a computer system that can be in one of two states: running or crashed. The uptime is modeled by a standard Weibull distribution with shape parameter 1.5, denoted $W(1, 1.5)$. This means that the likelihood of a crash increases with time. When crashed, the system can be brought back to the running state through a reboot. The reboot time is uniformly distributed in the interval $(1, 2)$. This computer system is a semi-Markov process because the holding time distributions are not memoryless.

To simulate execution of a semi-Markov process, we need to be able to generate non-uniform pseudo-random numbers. Typical pseudorandom number generators produce observations for a random variable U with a uniform distribution $U(0, 1)$. We can transform these observations into pseudorandom numbers distributed according to an arbitrary distribution function $F(x)$. The random variable $X = F^{-1}(U)$, where F^{-1} is the inverse of F, has distribution function $F(x)$, so an observation u of U can be transformed into an observation $x = F^{-1}(u)$ of X (von Neumann 1951). For example, the exponential distribution has cumulative distribution function $F(x) = 1 - e^{-\lambda x}$, so we can use $x = -\log(1 - u)/\lambda$ as a sample from the exponential distribution. The inverse method works well for many common probability distributions for which the inverse of $F(x)$ can be computed efficiently. Various other methods for generating non-uniform pseudorandom numbers are described by Devroye (1986).

Generalized Semi-Markov Processes

Both Markov and semi-Markov processes can be used to model a wide variety of stochastic discrete event systems, but without emphasis on the event structure. The queuing systems in Examples 2.6 and 2.7 are naturally described as having arrival and departure events, although the Markov processes we use to model the systems represent only the joint effects of all events enabled in a state. The *generalized semi-Markov process* (GSMP), first introduced by Matthes (1962), is an established formalism in queuing theory for modeling stochastic discrete event systems with focus on the event structure of a system (Glynn 1989).

A GSMP consists of a set of states S and a set of events E. At any time, the process occupies some state $s \in S$ in which a subset E_s of the events are enabled. Associated with each event $e \in E$ is a positive trigger time distribution G_e, and a next-state distribution $p_e(\cdot; \sigma_{\leq \tau}, t)$. The probability density function for G_e, $h_e(\cdot; \sigma_{\leq \tau})$, can depend on the entire execution history, which separates GSMPs from semi-Markov processes. Let T_e be a random variable representing the trigger time of e. If e just became enabled, then $\Pr[T_e \leq t \mid \sigma_{\leq \tau}] = H_e(t; \sigma_{\leq \tau}) = \int_0^t h_e(x; \sigma_{\leq \tau})\, dx$ is the probability that e triggers within t time units, provided that e remains continuously enabled. If e has already been enabled for u_e time units, then the probability of e triggering in the next t time units is

$$\Pr[T_e \leq t + u_e \mid T_e > u_e, \sigma_{\leq \tau}] = 1 - \Pr[T_e > t + u_e \mid T_e > u_e, \sigma_{\leq \tau}] = 1 - \frac{1 - H_e(t + u_e; \sigma_{\leq \tau})}{1 - H_e(u_e; \sigma_{\leq \tau})} \ . \tag{2.22}$$

By taking the derivative of (2.22) we get

$$h_e(t; u_e, \sigma_{\leq \tau}) = \frac{1}{1 - H_e(u_e; \sigma_{\leq \tau})} h_e(t + u_e; \sigma_{\leq \tau}) \ , \tag{2.23}$$

which is the conditional probability density function for the distribution G_e. The enabled events in a state race to trigger first, and the event that triggers causes a transition to a state $s' \in S$ according to the next-state distribution for the triggering event.

Example 2.10. Consider a queuing system with infinite capacity, and a state of this system is simply the number of items currently in the queue. There is an arrival event a, enabled in every state, that has an exponential trigger time distribution with rate $\frac{1}{5}$. There is also a departure event d that is enabled in states s_i for $i > 0$. This event has an exponential trigger time distribution with rate $\frac{1}{2}$. This queuing system is a GSMP with state space $S = \mathbb{Z}^*$ and event set $E = \{\mathsf{a}, \mathsf{d}\}$. Furthermore, we have $E_0 = \{\mathsf{a}\}$, $E_i = E$ for $i > 0$, $h_{\mathsf{a}}(t) = \frac{1}{5}e^{-t/5}$, $p_{\mathsf{a}}(i + 1; i) = 1$ for all $i \in S$, $h_{\mathsf{d}}(t) = \frac{1}{2}e^{-t/2}$, and $p_{\mathsf{d}}(i - 1; i) = 1$ for all $i > 0$.

For many stochastic discrete event systems, the trigger time and next-state distributions do not depend on every aspect of an entire trajectory prefix, as is clearly the case in Example 2.10. Let u_e, for each $e \in E$, represent the time that e has been continuously enabled without triggering. If $h_e(\cdot; \sigma_{\leq \tau}) = h_e(\cdot; s_k, u_e)$ and $p_e(\cdot; \sigma_{\leq \tau}, t) = p_e(\cdot; s_k)$, for all $e \in E$, then we have a *time homogeneous* GSMP (Glynn 1989, p. 18). A time homogeneous GSMP where all events have an exponential trigger time distribution with rate λ_e is also a time homogeneous Markov process. The holding time distribution for state s is an exponential distribution with rate $\lambda_s = \sum_{e \in E_s} \lambda_e$ and the transition probabilities are $p(s'; s) = \sum_{e \in E_s} p_e(s'; s)\lambda_e/\lambda_s$. The stochastic discrete event systems in Examples 2.7 and 2.10 are in fact equivalent.

To simulate the execution of a time homogeneous GSMP model, we associate a real-valued clock t_e with each event that indicates the time remaining until e is scheduled to trigger in the current state. The process starts in some initial state s with events E_s enabled. For each enabled event $e \in E_s$, we sample a trigger time according to G_e and set t_e to the sampled value. For disabled events, we set $t_e = \infty$. Let e^* be the event in E_s with the smallest clock value. This becomes the triggering event in s. Provided that at most one of the time distributions is not continuous, the probability of two events triggering at exactly the same time is zero so e^* is uniquely defined (Glynn 1989, p. 17). When e^* triggers after t_{e^*} time units in s, we sample a successor state s' according to $p_{e^*}(\cdot; \{\langle s, 0\rangle\}, t_{e^*})$ and update each clock t_e as follows:

1. if $e \in E_{s'} \cap \big(\{e^*\} \cup (E \setminus E_s)\big)$, then t'_e is sampled from G_e;
2. if $e \in E_{s'} \cap \big(E_s \setminus \{e^*\}\big)$, then $t'_e = t_e - t_{e^*}$;
3. otherwise, if $e \notin E_{s'}$ then $t'_e = \infty$.

The first rule covers events that are enabled in s' and either triggered or were not enabled in s. All such events are rescheduled. Events that remain enabled across state transitions without triggering are not rescheduled (rule 2). The final rule states that events disabled in s' are scheduled not to trigger. Given a new state s' and new clock values t'_e for each $e \in E$, we repeat the procedure just specified with $s = s'$ and $t_e = t'_e$ so long as $E_s \neq \emptyset$. Enabled events, annotated by a scheduled trigger time, can be stored in a heap to accommodate fast retrieval of e^* (Gonnet 1976). McCormack and Sargent (1981) compare various data structures for storing event schedules. Discrete event simulation is further discussed by Bratley et al. (1987) and Shedler (1993).

2.4 Stochastic Decision Processes

So far, we have discussed stochastic processes with a fixed structure. Now, let us consider the case when a decision maker can influence the structure and dynamics of the process, to some degree, and wants to select a structure that achieves some design objective. We then have a stochastic *decision* process.

The most widely adopted stochastic decision process is the *Markov decision process* (MDP; Bellman 1957; Howard 1960, 1971b; Puterman 1994; Boutilier et al. 1999). The dynamics of a discrete-time Markov process is captured by a transition probability matrix P. For an MDP, there are multiple transition probability matrices that a decision maker may choose from at each stage during execution. Each choice corresponds to an *action* on behalf of the decision maker. A transition probability matrix P^a represents the behavior of the system in the next time step if action a is chosen by the decision maker. For continuous-time MDPs, an action is instead represented by a infinitesimal generator matrix Q^a.

The decision maker designs a *policy*, denoted π, which is a mapping from *situations* to actions.[5] A situation can constitute the entire execution history (*history dependent policy*), the current time and state (*time dependent policy*), or just the current state (*stationary policy*). A policy may designate a fixed action to be used in a situation (*deterministic policy*), or a distribution over actions (*randomized policy*). The policy fixes the dynamics of a system, and an MDP coupled with a policy is a Markov process. For example, given a stationary randomized policy π and a set of actions A, the probability of transitioning from state i to j in the next time step is $\int_A P_{ij}^a \, d\pi(i)$.

Rewards and *costs* (negative rewards) are used to encode perceived value for a decision maker. Different reward structures can be used, but it is common to associate rewards with state transitions. For example, a transition from s to s' earns the decision maker an immediate reward $k(s, s')$. The transition rewards can depend on the action that is chosen for a state. For continuous-time models, it is also common to earn reward at some rate $c(s)$ for the duration of time that state s is occupied.

A decision maker chooses a policy according to some optimality criterion. The objective is generally to maximize the expected reward accumulated during execution, but this can be given different interpretations. Possibly the most straightforward interpretation is to maximize the *expected total reward*. This can be unbounded, however, if execution can proceed ad infinitum. To ensure that a bound exists, we can halt execution after a fixed time bound (*finite-horizon* total reward) or discount reward earned t time units into the future by a factor γ^t (*infinite-horizon* discounted reward). Other optimization criteria exist as well (cf. Puterman 1994).

Depending on the optimality criterion, it may not be necessary to consider the most general class of policies in order to act optimally. For example, to find a policy that maximizes the infinite-horizon discounted reward for an MDP, it is sufficient to consider the class of deterministic stationary policies. The

[5]In the model checking literature, a policy is called a *schedule*. Model checking for MDPs involves verifying that a property holds for a certain class of schedulers (cf. Bianco and de Alfaro 1995).

infinite-horizon discounted reward in state i of a discrete-time MDP controlled by a deterministic stationary policy π is given by the recurrence relation

$$v^{\pi}(i) = \bar{r}_{\pi(i)}(i) + \gamma \sum_{j \in S} P_{ij}^{\pi(i)} \cdot v^{\pi}(j) \ ,$$

where $\bar{r}_a(i) = \sum_{j \in S} P_{ij}^a \cdot k(i,j)$ is the expected transition reward in state i for action a (Howard 1960, p. 77). The optimal value is obtained by maximizing over the set of actions:

$$v^*(i) = \max_{a \in A} \left(\bar{r}_a(i) + \gamma \sum_{j \in S} P_{ij}^a \cdot v^*(j) \right) \tag{2.24}$$

This equation forms the basis for *value iteration*, which is a *dynamic programming* (Bellman 1957) technique for finding the optimal policy of an MDP. An alternative solution method is *policy iteration* (Howard 1960), which often requires fewer iterations than value iteration to converge, but with a higher cost per iteration. A middle ground is provided by Puterman and Shin's (1978) *modified policy iteration*.

Howard (1960) shows that the continuous-time MDP with discounting is computationally equivalent to its discrete-time counterpart, and describes how a continuous-time MDP can be transformed into a discrete-time MDP using a technique analogous to *uniformization* (Jensen 1953) for Markov processes. The equivalence between continuous and discrete-time MDPs is further explored by Lippman (1975), and generalized to countable state spaces by Serfozo (1979). Uniformization is a technique by which a continuous-time MDP with state-dependent exit rates can be transformed into an equivalent continuous-time MDP with the same (uniform) exit rate for all states. The uniform continuous-time MDP can then be treated as a discrete-time MDP resulting from observing the original continuous-time MDP at a constant rate. Uniformization introduces self-transitions not present in the original model, because it is possible to remain in the same state from one observation to another. Puterman (1994) presents uniformization as the preferred method for solving continuous-time MDPs, but we show in Chapter 9 that it can be more efficient to solve a continuous-time MDP directly, without first transforming it into a discrete-time MDP.

The *semi*-Markov decision process (SMDP; Howard 1963, 1971b), a decision theoretic extension of the semi-Markov process, permits time between state transitions to be governed by a general positive distribution. Chitgopekar (1969), Stone (1973), Cantaluppi (1984) consider generalizations of the SMDP model where the action choice is allowed to change not only at the time of state transitions, but also at time points between state transitions. Chapter 9 discusses this issue further.

Chapter 3

Related Work

This chapter discusses related research in the model checking, operations research, and AI planning literature. We focus primarily on research dealing with probabilistic systems, although in the case of planning we also mention efforts involving nondeterministic systems. We do not attempt to produce an exhaustive account of all past research concerning probabilistic verification and planning under uncertainty, as it would be a daunting task. The research efforts mentioned in this chapter should instead be thought of as a representative sample of all related work.

3.1 Probabilistic Verification

Early work on probabilistic verification has a clear focus on discrete time models, with the verification of randomized algorithms as the primary application in mind. Hart et al. (1983) analyze termination of concurrent probabilistic programs. Lehmann and Shelah (1982) and Hart and Sharir (1984) introduce probabilistic temporal logics for specifying properties of probabilistic programs. These logics can only express properties that either hold with probability one or with non-zero probability, so a verification algorithm can ignore the actual probabilities in a model. In contrast, the logic of Reif (1980) permits properties with rational probability thresholds other than zero and one. Automated verification as *model checking* was pioneered by Clarke and Emerson (1982). Vardi (1985) and Courcoubetis and Yannakakis (1995) describe model checking algorithms for linear temporal logic, LTL, when the model is a probabilistic program and the LTL formula is required to hold with probability one.

Hansson and Jonsson (1989, 1994) present the *probabilistic real-time computation tree logic*, PCTL, based on CTL (Clarke and Emerson 1982; Clarke et al. 1986) but with the path quantifiers "for all trajectories" ($\forall$) and "there exists a trajectory" ($\exists$) replaced by a single probabilistic path quantifier with a probability threshold not restricted to the values zero and one. In PCTL, one can also associate a time bound with a path operator, such as "until", enabling one to impose deadlines for reaching certain states. PCTL formulae are interpreted over discrete time Markov processes, and each state transition corresponds to a time unit. Hansson and Jonsson provide algorithms for PCTL model checking with finite-state models. In the general case, with a finite time bound and a probability threshold in the interval $(0, 1)$, PCTL model checking can be solved numerically in either $O(t \cdot (|S| + |E|))$ or $O(\log(t) \cdot |S|^3)$ time, where t is the time bound, $|S|$ is the size of the state space, and $|E|$ is the number of state transitions with non-zero probability in the Markov process. Since $|E|$ is at most $|S|^2$, and typically no less than $|S|$, PCTL model checking is polynomial in the size of the state space. To handle large state spaces, Baier et al. (1997) propose using multi-terminal binary decision diagrams, MTBDDs (Clarke et al. 1993; Bahar et al. 1993; Fujita et al. 1997), to carry out the numerical computations.

Aziz et al. (1996, 2000) propose the logic CSL, the *continuous stochastic logic*, as a variation of PCTL for expressing properties of continuous-time Markov processes. They prove that CSL model checking is decidable for rational time bounds, but do not provide a practical model checking algorithm. Baier et al. (1999) present a numerical model checking algorithm, using MTBDDs, for a variation of CSL with the addition of a steady-state operator. Model checking of time-bounded CSL formulae amounts to solving a system of Volterra integral equations, but solving this equation system is time consuming and numerical stability is hard to achieve (Hermanns et al. 2000). A better solution method is provided by Baier et al. (2000), who show that CSL model checking of time-bounded formulae can be reduced to transient analysis of continuous-time Markov processes and suggest the use of sparse matrices instead of MTBDDs. The former means that time-bounded CSL properties can be verified using existing techniques for transient analysis, in particular *uniformization*[1] (Jensen 1953), which have been used extensively in the performance evaluation literature (Grassmann 1977; Gross and Miller 1984; Reibman and Trivedi 1988; Malhotra et al. 1994). While Baier et al. suggest that uniformization should be applied to each individual state separately, resulting in a time complexity of $O(q \cdot t \cdot |S| \cdot |E|)$ for time-bounded CSL formulae (q is the *uniformization*

[1]Other names for this technique are *randomization* and *Jensen's method*.

constant and can be set to the maximum exit rate for the model), Katoen et al. (2001) improve the time complexity by a factor $O(|S|)$ by noting that uniformization can be performed for all states simultaneously. These contributions are summarized by Baier et al. (2003).

While MTBDDs often can represent the transition matrix of a Markov process in a compact manner, they are not always an efficient representation for numerical computation. Kwiatkowska et al. (2002b, 2004) explore different representations, including a *hybrid* approach that combines the MTBDD representation of transition matrices with a flat representation of iteration vectors. This hybrid approach is generally faster than MTBDDs, while handling larger systems than sparse matrices. Another promising approach is presented by Buchholz et al. (2003), who use Kronecker products to exploit structure in the models.

Infante López et al. (2001) go beyond Markov models by considering the CSL model checking problem for semi-Markov processes. For CSL formulae without a time bound, the problem reduces to probabilistic model checking for discrete-time Markov processes. For time-bounded formulae, the model checking problem amounts to solving a system of Volterra integral equations. A different approach is taken by Kwiatkowska et al. (2002a). Time bounds in CSL are typically specified as intervals of real numbers, but Kwiatkowska et al. associate positive probability distributions with time bounds and suggest that this can be used to express certain properties of systems with general distributions while still using Markov models of the systems. Alur et al. (1991) describe a model checking algorithm for generalized semi-Markov processes, but only for probability thresholds zero or one and restricted to trigger time distributions with finite support. Kwiatkowska et al. (2000) use a similar approach for probabilistic timed automata, and permit arbitrary probability thresholds.

Even with the use of clever data structures, numerical solution techniques tend to suffer greatly from the state space explosion problem. The statistical approach presented in Chapter 5 is an attempt to overcome the limitations of numerical solution techniques for large state spaces, while providing only statistical correctness guarantees. Lassaigne and Peyronnet (2002) propose a statistical approach for model checking a fragment of LTL. They do not formulate a *hypothesis testing* problem, but instead rely on less efficient techniques for statistical *estimation*. In probabilistic model checking, the question is whether a probability is above or below some threshold, and it would typically be a waste of effort to obtain an accurate estimate of a probability only to realize that it is far from the specified threshold. Grosu and Smolka (2004) present a Monte Carlo approach to LTL model checking for non-probabilistic systems, but take the same approach as Lassaigne and Peyronnet by relying on statistical estimation rather than hypothesis testing. In this case, it makes even less sense to use estimation techniques because the estimated probability has no clear meaning—there are no probabilities in the model. Sen et al. (2004) describe a statistical approach, based on hypothesis testing, for verifying probabilistic systems. They assume that the system has already been deployed so that execution traces cannot be generated on demand. We discuss their approach in further detail in Chapter 7, where we expose some serious flaws in their proposed solution method.

3.2 Planning under Uncertainty

Current approaches to planning under uncertainty can be divided roughly into distinct categories based on their representation of uncertainty, how goals are specified, the model of time used, and assumptions made regarding observability. Two prevalent representations of uncertainty are *nondeterministic* and *stochastic* models. In nondeterministic models, uncertainty is represented strictly logically, using disjunction, while in stochastic models uncertainty is specified with probability distributions over the possible outcomes of events and actions.

The objective when planning with nondeterministic models is often, although not always, to generate a *universal plan* (Schoppers 1987) that is guaranteed to achieve a specified goal regardless of the actual outcomes of events and actions. A goal can be a set of desirable states, as in the work of Cimatti et al. (1998) and Jensen and Veloso (2000), or a modal temporal logic formula as proposed by Kabanza et al. (1997) and Pistore and Traverso (2001). Conditional planners, such as CNLP (Peot and Smith 1992) and PLINTH (Goldman and Boddy 1994a), are also examples of planners for nondeterministic domains.

Ginsberg (1989) questions the practical value of universal nondeterministic planning. His main concern is that the representation of a universal plan is bound to be infeasibly large for interesting problems. It is impractical, Ginsberg argues, for an agent to precompute its response to every situation in which it might find itself, simply because the number of situations is prohibitively large. In control theory, Balemi et al. (1993) propose the use of ordered *binary decision diagrams* (BDDs; Bryant 1986) as a compact representation of supervisory controllers, and this representation has more recently also been used in the AI community for nondeterministic planning (Cimatti et al. 1998; Jensen and Veloso 2000). Kabanza et al. (1997) attempt to address the time complexity problem by proposing an incremental algorithm for constructing partial policies.

Their planning system relies on domain-specific search control rules for efficiency, and produces a universal plan if given enough time.

By requiring a stochastic domain model, with state transitions weighted by probabilities, a probabilistic planner has a more detailed model of uncertainty to work with. It can therefore choose to focus planning effort on the most relevant parts of the state space. A plan may fail because some contingencies have not been planned for, but this is acceptable so long as the success probability of the plan is high. Having to deal with probabilities can, however, be computationally more challenging than working with nondeterministic models. In recent work, Jensen et al. (2004) present a compromise solution, distinguishing between primary and secondary effects of actions without assigning probabilities to state transitions. The resulting planning framework can produce plans that are robust for up to n faults.

Drummond and Bresina (1990) present an anytime algorithm for generating partial policies with high probability of achieving goals expressed using a modal temporal logic. Other research on probabilistic planning typically considers only propositional goals. Kushmerick et al. (1995) and Lesh et al. (1998) work with plans consisting of actions that are executed in sequence regardless of the outcome of the previous actions. This is often called *conformant* planning (Smith and Weld 1998). Conditional probabilistic plans (Blythe 1994; Draper et al. 1994; Goldman and Boddy 1994b) allow for some adaptation to the situation during plan execution. In the work by Draper et al., this adaptation is obtained by means of explicit sensing actions that are made part of the plan.

Sampling techniques have been used for probabilistic plan assessment by Blythe (1994) and Lesh et al. (1998). In both cases, however, the probability of plan success is estimated using flawed statistical methods. The estimation is based on the normal assumption, which is known to give unreliable results when used to estimate proportions (see, e.g., Fujino 1980; Hall 1982; Agresti and Coull 1998; Newcombe 1998; Brown et al. 2001). Furthermore, statistical hypothesis testing would be more appropriate in both cases because the probability estimate is only used to compare two plans or to test if the success probability exceeds a specified threshold. Lesh et al. use an interesting data mining technique, however, for analyzing simulation traces in order to discover plan flaws. The technique, which is more thoroughly described by Zaki et al. (2000), targets discrete-time planning domains. It has some similarities with our failure analysis approach presented in Chapter 8, and is in many ways more ambitious than our approach.

In decision theoretic planning, a reward structure is added to the probabilistic model, and the objective is to find a control policy that maximizes the expected reward during execution. The discrete-time MDP formalism (Section 2.4) has received significant attention in the AI community in the past decade, with applications ranging from robot navigation (Koenig et al. 1995) to elevator control (Nikovski and Brand 2003). Considerable progress has been made on algorithms for MDP planning that exploit structure in the model. Boutilier et al. (1995) use *dynamic Bayesian networks* (Dean and Kanazawa 1989) to represent transition probability matrices and decision trees to represent conditional probability tables and policies, and propose the *structured policy iteration* algorithm. Hoey et al. (1999) use a similar approach, but replace decision trees with MTBDDs.[2]

Even structured solution techniques suffer from the state space explosion problem. Approximate solution techniques, including automated state abstraction (Boutilier and Dearden 1994; Dearden and Boutilier 1997) and value function approximation (Bellman et al. 1963; Gordon 1995; Guestrin et al. 2003) aim to address this problem by sacrificing optimality for efficiency.

Boyan and Littman (2001) propose an extension of MDPs—time-dependent MDPs (TMDPs)—where the time between state transitions can depend on the current time. The model corresponds to a *general state space* MDP with a single continuous state variable representing global time. State spaces with multiple continuous state variables are considered by Feng et al. (2004), but restricted to discrete transition functions (i.e. each state can only have a finite number of possible successors).

In Chapter 9, we introduce the *generalized* semi-Markov decision process (GSMDP), which can be used to model decision theoretic planning problems with *asynchronous* events and actions. GSMDPs can be viewed as compositions of asynchronous SMDPs. They differ from TMDPs in that they essentially require one local clock for each event in the model. The algorithm of Feng et al. (2004) can handle multiple continuous state variables, but the restriction to discrete transition functions makes it inadequate for GSMDP planning. Some attention has recently been given to planning with *concurrent* actions. Guestrin et al. (2002) and Mausam and Weld (2004) use discrete-time MDPs to model and solve planning problems with concurrent actions, but these approaches are restricted to instantaneous actions executed in synchrony. Rohanimanesh and Mahadevan (2001) consider planning problems with temporally extended actions that can be executed in parallel. By restricting the temporally extended actions to *Markov options*, the resulting planning problems can be modeled as discrete-time SMDPs.

[2] MTBDDs are also know as *algebraic decision diagrams* (ADDs), and this is the name typically used in the AI literature.

The GSMDP framework can be thought of as a probabilistic and decision theoretic extension of the planning framework developed by Musliner et al. (1995), which is part of the CIRCA architecture. The CIRCA domain model is a nondeterministic timed automata, with uncertainty in the duration and outcome of events and actions. The nondeterministic domain model is essentially a GSMP, but with intervals of possible delays in place of delay distributions and without probabilities associated with state transitions. The CIRCA planner can generate plans that are guaranteed to maintain system safety. Plan generation is done incrementally. Starting from the initial state, actions are assigned to states as the states are determined to be reachable. Following each action assignment, the current plan is verified to see if failure is always avoided. If a failure state is reachable, the planner backtracks to consider alternative action assignments. A counterexample, in the form of an execution trace, is generated by the verifier if a plan is determined to be unsafe. The counterexample traces can be used to guide plan repair (Goldman et al. 2004). The probabilistic planning framework presented in Chapter 8 is based on the CIRCA planning framework. In particular, it makes use of a verifier to find reasons for plan failure.

Atkins et al. (1996) describe a probabilistic extension of CIRCA, but it does not permit a modular specification of asynchronous events. The user is required to specify the joint distribution for any set of events that can be enabled simultaneously, which can be rather cumbersome. Furthermore, their approach does not handle state spaces with cycles. Li et al. (2003) attempt to address some of these issues, but rely on ad hoc approximation techniques using "probability rate functions." The use of phase-type distributions, as described in Chapter 9, is a more principled way of dealing with general delay distributions for asynchronous events and actions.

Part I

Verification

Chapter 4

Specifying Properties of Stochastic Discrete Event Systems

Given a stochastic discrete event system, it is often of interest to be able to specify properties of the system. These properties could represent behavior that we want the system to exhibit during execution. For example, a desirable property of a telephone system might be that the probability of a call getting dropped is low. To enabled automatic verification of stochastic discrete event systems, we need a formalism for expressing interesting properties of such systems. This chapter introduces the *unified temporal stochastic logic* (UTSL), which can be used to express properties such as "the probability is at most 0.01 that a call is dropped within 60 minutes from now." UTSL has essentially the same syntax as the existing logics PCTL and CSL, but UTSL provides a unified semantics for both discrete-time and continuous-time systems, as well as systems with discrete, continuous, and general state spaces. This will allow us, for the most part, to treat all stochastic discrete event systems uniformly when presenting a statistical approach to probabilistic model checking in the next chapter.

4.1 Temporal Logic

The use of *temporal logic* (Rescher and Urquhart 1971) for specifying properties of deterministic and non-deterministic systems with program verification in mind was pioneered by Pnueli (1977) and is now a widespread practice in the model checking community. The propositional branching-time logic CTL (computation tree logic; Clarke and Emerson 1982; Clarke et al. 1986), a particularly popular formalism, can be used to express properties such as "for all trajectories, Ψ eventually becomes true with Φ holding continuously until then" and "there exists a trajectory such that Φ holds after the next state transition." CTL is related to $\mathscr{UB}$ (Ben-Ari et al. 1981, 1983) and the branching-time temporal logic described by Lamport (1980). Emerson (1990) provides an excellent survey of temporal logics with a model checking perspective.

For many real-time systems, it is important to ensure that deadlines are met. To reason about deadlines, we need to be able to express quantitative temporal properties of a system. Extensions of CTL with time as a discrete (RTCTL; Emerson et al. 1990, 1992) or continuous (TCTL; Alur et al. 1990, 1993) quantity have therefore been proposed. With RTCTL and TCTL, it is possible to express timed properties such as "for all trajectories, Φ becomes true within t time units." Earlier work in the same direction includes Bernstein and Harter's (1981) extension of Lamport's logic that associates time bounds with eventualities. A survey on the topic of logics for real-time systems is provided by Alur and Henzinger (1992).

The logic TCTL has also been proposed as a formalism for expressing properties of continuous-time stochastic systems, but with "for all trajectories" ($\forall$) and "there exists a trajectory" ($\exists$) reinterpreted as "with probability one" and "with positive probability", respectively (Alur et al. 1991). The same interpretation is given to the path quantifiers $\forall$ and $\exists$ in earlier work by Hart and Sharir (1984) on the branching-time logic PTL for discrete-time stochastic processes.

4.2 UTSL: The Unified Temporal Stochastic Logic

In many cases, it is not economically or physically feasible to ensure certain behaviors with probability one, but simply guaranteeing that the behavior can be exhibited by the system with positive probability may be too weak. For example, designing a telephone system where no call is ever dropped would be excessively costly, but it is not satisfactory to just know that a call can possibly go through. For the telephone system, we would like to ensure that calls go though with a reasonably high probability, for example 0.9999. Neither TCTL nor PTL permit us to express such a property. For this, we need a different path quantifier, which is provided by PCTL (Hansson and Jonsson 1989, 1994). PCTL has quantitative time bounds just as RTCTL, on which PCTL is based, but the path quantifiers $\forall$ and $\exists$ are replaced by a single probabilistic path quantifier.

This lets us express quantitative bounds on the probability of a set of trajectories. For example, PCTL can express the property "with probability at least θ, Φ will be satisfied within t time units."

PCTL formulae are interpreted over discrete-time Markov processes. Aziz et al. (1996, 2000) propose a similar logic, CSL (continuous stochastic logic), with formulae interpreted over continuous-time Markov processes. A variation of CSL has been proposed by Baier et al. (1999, 2003), which also includes a facility for expressing bounds on steady-state probabilities. This version of CSL has also been used for expressing properties of semi-Markov processes (Infante López et al. 2001). Yet another logic, with essentially the same syntax as PCTL, has been proposed for expressing properties of probabilistic timed automata (Kwiatkowska et al. 2000). While the difference in syntax is minimal between all mentioned logics for expressing probabilistic real-time properties, the semantics of the various logics are tied to specific classes of stochastic processes, for example discrete-time Markov processes in the case of PCTL. To avoid having to refer to different logics for different classes of systems, we introduce the logic UTSL, with a unified semantics for all measurable stochastic discrete event systems.

The syntactic structure of UTSL is the same as that of both CSL (without the steady-state operator) and PCTL, although we use the notation of Baier et al. (2003) rather than that of Hansson and Jonsson (1994).

Definition 4.1 (UTSL Syntax). Let $\mathcal{M} = \langle S, T, \mu, SV, V\rangle$ be a factored stochastic discrete event system. The syntax for UTSL is defined inductively as follows:

1. $x \sim v$ is a UTSL formula for $x \in SV$, $v \in D_x$, and $\sim \in \{\leq, =, \geq\}$.
2. $\neg\Phi$ is a UTSL formula if Φ is a UTSL formula.
3. $\Phi \wedge \Psi$ is a UTSL formula if both Φ and Ψ are UTSL formulae.
4. $\mathcal{P}_{\bowtie\theta}\big[X^I\ \Phi\big]$, for $\bowtie \in \{\leq, \geq\}$, $\theta \in [0,1]$ and $I \subset T$, is a UTSL formula if Φ is a UTSL formula.
5. $\mathcal{P}_{\bowtie\theta}\big[\Phi\ \mathcal{U}^I\ \Psi\big]$, for $\bowtie \in \{\leq, \geq\}$, $\theta \in [0,1]$ and $I \subset T$, is a UTSL formula if both Φ and Ψ are UTSL formulae.

If the time domain T is the non-negative integers, UTSL syntax coincides with PCTL syntax, and we get CSL syntax by letting T be the non-negative real numbers.

The standard logic operators, $\neg$ and $\wedge$, have their usual meaning. The UTSL operator $\mathcal{P}_{\bowtie\theta}[\cdot]$ replaces the traditional CTL path quantifiers $\forall$ and $\exists$. The truth value of a path formula φ, i.e. either $X^I\ \Phi$ ("next") or $\Phi\ \mathcal{U}^I\ \Psi$ ("until"), is determined over a trajectory (sample path) for a system. The path formula $X^I\ \Phi$ asserts that the next state transition occurs $t \in I$ time units into the future and that Φ holds at the time instant immediately following the state transition, while $\Phi\ \mathcal{U}^I\ \Psi$ asserts that Ψ becomes true $t \in I$ time units into the future while Φ holds continuously prior to time t. Since we are dealing with measurable stochastic systems, there is some probability associated with the set of trajectories that satisfy φ. The probabilistic path quantifier $\mathcal{P}_{\bowtie\,\theta}[\cdot]$ permits us to compare this probability against an arbitrary threshold θ.

Definition 4.1 provides a bare-bones version of UTSL. Additional UTSL formulae can be derived in the usual way. For example, $\perp\ \equiv\ (x = v) \wedge \neg(x = v)$ for some $x \in SV$ and $v \in D_x$, $\top \equiv \neg\perp$, $x < v \equiv \neg(x \geq v)$, $\Phi \vee \Psi \equiv \neg(\neg\Phi \wedge \neg\Psi)$, $\Phi \rightarrow \Psi \equiv \neg\Phi \vee \Psi$, and $\mathcal{P}_{<\,\theta}[\varphi] \equiv \neg\mathcal{P}_{\geq\,\theta}[\varphi]$. We have associated a time bound I with the path operators X and $\mathcal{U}$. The unbounded versions of these operators are obtained by letting I equal the time domain T, for example $\mathcal{P}_{\bowtie\,\theta}[\Phi\ \mathcal{U}\ \Psi] \equiv \mathcal{P}_{\bowtie\,\theta}\big[\Phi\ \mathcal{U}^T\ \Psi\big]$. We can derive additional path operators, such as $\mathcal{W}$ ("weak until"), $\Diamond$ ("eventually"), and $\Box$ ("continuously"), as follows (Hansson and Jonsson 1994):

$$
\begin{aligned}
\mathcal{P}_{\geq\,\theta}\big[\Phi\ \mathcal{W}^I\ \Psi\big] &\equiv \mathcal{P}_{\leq\,1-\theta}\big[\neg\Psi\ \mathcal{U}^I\ \neg(\Phi \vee \Psi)\big] \\
\mathcal{P}_{\leq\,\theta}\big[\Phi\ \mathcal{W}^I\ \Psi\big] &\equiv \mathcal{P}_{\geq\,1-\theta}\big[\neg\Psi\ \mathcal{U}^I\ \neg(\Phi \vee \Psi)\big] \\
\mathcal{P}_{\bowtie\,\theta}\big[\Diamond^I\ \Phi\big] &\equiv \mathcal{P}_{\bowtie\,\theta}\big[\top\ \mathcal{U}^I\ \Phi\big] \\
\mathcal{P}_{\bowtie\,\theta}\big[\Box^I\ \Phi\big] &\equiv \mathcal{P}_{\bowtie\,\theta}\big[\Phi\ \mathcal{W}^I\ \perp\big]
\end{aligned}
$$

Unbounded versions of these path operators can be derived in the same way as for X and $\mathcal{U}$.

4.3 UTSL Semantics and Model Checking Problems

The validity of a UTSL formula is determined relative to a trajectory prefix. For simple UTSL formulae of the form $x \sim v$, the validity depends only on the last state of the trajectory prefix, but this is not necessarily the case for UTSL formulae containing one or more probabilistic operators. The formal semantics of UTSL is given by the following inductive definition.

Definition 4.2 (UTSL Semantics). Let $\mathcal{M} = \langle S, T, \mu, SV, V\rangle$ be a factored stochastic discrete event system. With $Path(\sigma_{\leq\tau})$ denoting the set of trajectories with common prefix $\sigma_{\leq\tau}$ and the definition of T_i

given by (2.18), satisfaction relations for UTSL formulae and path formulae are inductively defined by the following rules:

$$
\begin{aligned}
\mathcal{M}, \{\langle s_0, t_0\rangle, \ldots, \langle s_k, t_k\rangle\} &\models x \sim v && \text{if } V(s_k, x) \sim v \\
\mathcal{M}, \sigma_{\leq\tau} &\models \neg\Phi && \text{if } \mathcal{M}, \sigma_{\leq\tau} \not\models \Phi \\
\mathcal{M}, \sigma_{\leq\tau} &\models \Phi \wedge \Psi && \text{if } (\mathcal{M}, \sigma_{\leq\tau} \models \Phi) \wedge (\mathcal{M}, \sigma_{\leq\tau} \models \Psi) \\
\mathcal{M}, \sigma_{\leq\tau} &\models \mathcal{P}_{\bowtie\theta}[\varphi] && \text{if } \mu(\{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \models \varphi\}) \bowtie \theta
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{M}, \sigma, \tau &\models X^I\, \Phi && \text{if } \exists k \in \mathbb{N}.\big((T_{k-1} \leq \tau) \wedge (\tau < T_k) \wedge (T_k - \tau \in I) \wedge (\mathcal{M}, \sigma_{\leq T_k} \models \Phi)\big) \\
\mathcal{M}, \sigma, \tau &\models \Phi\, \mathcal{U}^I\, \Psi && \text{if } \exists t \in I.\big((\mathcal{M}, \sigma_{\leq\tau+t} \models \Psi) \wedge \forall t' \in T.\big((t' < t) \rightarrow (\mathcal{M}, \sigma_{\leq\tau+t'} \models \Phi)\big)\big)
\end{aligned}
$$

Definition 4.2 specifies the validity of a UTSL formula at any time during execution of a stochastic discrete event system. We typically want to know whether a property Φ holds for a model $\mathcal{M}$ if execution starts in a specific state s. The triple $\langle\mathcal{M}, s, \Phi\rangle$ is a *model checking problem* with an affirmative answer if and only if $\mathcal{M}, \{\langle s, 0\rangle\} \models \Phi$. More generally, we can define the validity of a UTSL formula relative to a probability measure μ_0, such that $\mu_0(S')$ is the probability that execution starts in a state $s \in S'$. This is accomplished with the addition of the following rules:

$$
\begin{aligned}
\mathcal{M}, \mu_0 &\models x \sim v && \text{if } \forall s \in \sup \mu_0.\big(\mathcal{M}, \{\langle s, 0\rangle\} \models x \sim v\big) \\
\mathcal{M}, \mu_0 &\models \neg\Phi && \text{if } \mathcal{M}, \mu_0 \not\models \Phi \\
\mathcal{M}, \mu_0 &\models \Phi \wedge \Psi && \text{if } (\mathcal{M}, \mu_0 \models \Phi) \wedge (\mathcal{M}, \mu_0 \models \Psi) \\
\mathcal{M}, \mu_0 &\models \mathcal{P}_{\bowtie\theta}[\varphi] && \text{if } \int_S \mu(\{\sigma \in \mathit{Path}(\{\langle s, 0\rangle\}) \mid \mathcal{M}, \sigma, 0 \models \varphi\})\, d\mu_0(S) \bowtie \theta
\end{aligned}
$$

The probability integral in the last rule reduces to $\sum_{s \in S} \mu_0(s)\mu(\{\sigma \in \mathit{Path}(\{\langle s, 0\rangle\}) \mid \mathcal{M}, \sigma, 0 \models \varphi\})$ if the state space S is countable. A UTSL model checking problem can now be specified as a triple $\langle\mathcal{M}, \mu_0, \Phi\rangle$. This definition subsumes the definition with a single initial state.

The semantics of $\Phi\, \mathcal{U}^I\, \Psi$ requires that Φ holds continuously, i.e. at every point in time, along a trajectory until Ψ is satisfied. If Φ and Ψ are both free of any probabilistic operators, however, then the truth values of these subformulae do not depend on the amount of time that is spent in a specific state. Without nested probabilistic operators, it is therefore sufficient to verify the subformulae Φ and Ψ at the entry of each

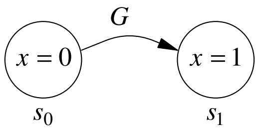


Figure 4.1: A simple two-state semi-Markov process. The time from when the left state (s_0) is entered until the transition to the right state (s_1) occurs is a random variable with distribution G.

state along a trajectory. The same can be said for stochastic discrete event systems that satisfy the Markov property (2.20), even with nested probabilistic operators present, because the Markov property ensures that the amount of time spent in a state does not have an impact on the future behavior of the process.

In general, with nested probabilistic operators and without the Markov assumption, it may not be sufficient to verify the subformulae Φ and Ψ of $\Phi\ \mathcal{U}^I\ \Psi$ at the time of state transitions. We illustrate this with two simple examples. The first example shows that a UTSL formula can be true at the entry of a state without holding continuously while remaining in the same state. The second example shows that a UTSL formula can become true while remaining in a state without being true at the entry of the state. It should be noted that the statistical model checking algorithm we present in the next chapter can deal with nested probabilistic operators only if it is sufficient to verify nested formulae at discrete points along a trajectory, which generally means that we must be dealing with a discrete-time model or a model satisfying the Markov property.

Example 4.1. Consider the semi-Markov process with two states depicted in Figure 4.1. Assume that G is a standard Weibull distribution with shape parameter 0.5, denoted $W(1, 0.5)$, and that we want to verify the UTSL formula $\Phi = \mathcal{P}_{\geq 0.5}[\varphi]$, where φ is the path formula $\mathcal{P}_{\geq 0.5}\big[x{=}0\ \mathcal{U}^{[0,1]}\ x{=}1\big]\ \mathcal{U}^{[0,1]}\ x{=}1$, relative to the trajectory prefix $\{\langle s_0, 0\rangle\}$.

To solve this problem, we compute the probability measure of the set of trajectories that start in s_0 at time 0 and satisfy the path formula φ. Let P denote this set. Members of P are of the form $\{\langle s_0, t\rangle, \langle s_1, \infty\rangle\}$ with $t \in [0, t']$ for some $t' \leq 1$. The probability measure of P is therefore at most $F(1) \approx 0.632$, where $F(\cdot)$ is the cumulative distribution function for $W(1, 0.5)$. Of the trajectories with $t \in [0, 1]$, only the ones where $\Psi = \mathcal{P}_{\geq 0.5}\big[x{=}0\ \mathcal{U}^{[0,1]}\ x{=}1\big]$ holds until s_1 is reached satisfy the path formula φ.

If we require Ψ to hold continuously along a trajectory until s_1 is reached, then we have to rule out trajectories with $t \geq t'$ such that Ψ does not hold if verified relative to the trajectory prefix $\{\langle s_0, t'\rangle\}$. The probability of reaching s_1 within 1 time unit, given that we have already spent t' time units in s_0, is given by the formula

$$q(t') = \frac{1}{1 - F(t')} \int_{t'}^{t'+1} f(x)\,dx$$

where $f(\cdot)$ is the probability density function for $W(1, 0.5)$. The value of q is greater than 0.5 for $t' = 0.1$, but less than 0.5 for $t' = 0.2$. Since q is a decreasing function of t', it means that Ψ does not hold continuously over trajectories starting in s_0 if $t \geq 0.2$. It follows that the probability measure of the set P is less than $F(0.2) \approx 0.361$, so Φ does not hold. We would reach the opposite conclusion if we simply verified the nested formulae at the entry of each state, since Ψ holds initially in s_0.

Example 4.2. Consider the same two-state semi-Markov process as in the previous example, but this time with G equal to $W(1, 1.5)$. Assume that we want to verify the UTSL formula $\Phi = \mathcal{P}_{\geq 0.5}[\varphi]$, where φ is the path formula $x{=}0 \ \mathcal{U}^{[0,1]} \ \mathcal{P}_{\geq 0.7}\big[x{=}0 \ \mathcal{U}^{(0,1]} \ x{=}1\big]$. Note that the time interval is open to the left in the formula $\Psi = \mathcal{P}_{\geq 0.7}\big[x{=}0 \ \mathcal{U}^{(0,1]} \ x{=}1\big]$, so Ψ cannot hold in s_1 because $x{=}0$ must hold at the entry of a state for Ψ to hold in that state. Ψ does not hold immediately in s_0 either: the probability of reaching s_1 within 1 time unit is $F(1) \approx 0.632 < 0.7$ at time 0 in s_0. The formula Ψ does become true, however, along trajectories that remain in s_0 for 0.2 time units or more before transitioning to s_1. Since $F(1) - F(0.2) \approx 0.547 \geq 0.5$, it follows that Φ holds with the semantics given by Definition 4.2.

Our semantics for time-bounded until is consistent with that of TCTL defined by Alur et al. (1991). Infante López et al. (2001) propose a semantics of CSL for semi-Markov processes that does not require subformulae to hold continuously in a state along a trajectory. With their semantics, one would get the opposite result in both of the examples above. While the semantics of Infante López et al. makes it easier to verify properties with nested probabilistic operators, it is not consistent with the common definition of a trajectory for a continuous-time discrete event system as a piecewise linear function of time. Furthermore, one could imagine using phase-type distributions to approximate the Weibull distributions in the two examples and verify the properties for the resulting Markov processes. The introduction of phase transitions would result in nested formulae possibly being verified at different times in the same state, which is inconsistent with the semantics of Infante López et al.

Chapter 5

Statistical Probabilistic Model Checking

This chapter presents a statistical approach to probabilistic model checking, employing hypothesis testing and discrete event simulation. The proposed solution method works for any discrete event system that can be simulated, although the method for verifying properties with nested probabilistic statements is limited to discrete-time systems or systems satisfying the Markov property. We prove two fundamental theorems that establish efficient verification procedures for conjunctive and nested probabilistic statements, we discuss benefits and hazards of using distributed sampling, and we provide complexity results for the statistical solution method.

Consider the UTSL model checking problem $\langle \mathcal{M}, \mu_0, \mathcal{P}_{\bowtie\,\theta}[\varphi] \rangle$. The set of trajectories satisfying φ and with the initial state distributed according to μ_0 has probability measure

$$p = \int_S \mu(\{\sigma \in \mathit{Path}(\{\langle s, 0 \rangle\}) \mid \mathcal{M}, \sigma, 0 \models \varphi\})\, d\mu_0(S) \ .$$

We could solve the model checking problem by computing p and then compare it to the threshold θ, but a numerical computation of p is not feasible for certain classes of stochastic discrete event systems, in particular many infinite-state systems and generalized semi-Markov processes. For Markov processes, efficient numerical techniques for computing p do exist (Hansson and Jonsson 1994; Baier et al. 2003), but the computational complexity of these techniques is proportional to the size of the state space, which puts limits on their applicability for verifying properties of stochastic systems with large state spaces.

Simulation has often been advertised as a last resort when numerical techniques fail (see, e.g., Teichroew and Lubin 1966; Buchholz 1998) and it is a technique with roots in the infancy of computer science. The

Monte Carlo method (Ulam and von Neumann 1947; Metropolis and Ulam 1949), which is essentially a statistical approach to the study of integro-differential equations, was conceived by S. Ulam in 1946 to solve problems in mathematical physics on ENIAC—the first digital computer (Metropolis 1987; Eckhardt 1987).

It is therefore reasonable to consider statistical techniques, involving simulation and sampling, to solve UTSL model checking problems. For this purpose, we set up a chance experiment represented by Bernoulli variates X_i with parameter p. We could then proceed by estimating p with a confidence interval using techniques for estimating the mean of a distribution with unknown variance (see, e.g., Chow and Robbins 1965; Nádas 1969; Raatikainen 1995). Note, however, that in order to verify the UTSL formula $\mathcal{P}_{\bowtie\theta}[\varphi]$, we do not need to have an accurate estimate of p—we need to know only if p is above or below the threshold θ. It would be a waste of effort to obtain an accurate estimate of p, only to realize that p is far from θ.

In Section 2.2, we discussed acceptance sampling, a statistical technique for testing if the parameter p of a Bernoulli variate is above or below a threshold θ. This is exactly what we need for UTSL model checking. The verification of a probabilistic UTSL formula, for example $\Phi = \mathcal{P}_{\geq\theta}[\varphi]$, can be thought of in terms of hypothesis testing. To verify Φ we need to test the hypothesis $H : p \geq \theta$ against the alternative hypothesis $K : p < \theta$. We first restrict our attention to UTSL formulae without nested probabilistic operators. In Section 5.2, we consider the general case and show how nested probabilistic operators can be dealt with using statistical techniques, at least for certain classes of stochastic discrete event systems.

5.1 Model Checking without Nested Probabilistic Operators

To use acceptance sampling for the purpose of UTSL model checking, we need to introduce the concept of an indifference region. With each formula of the form $\mathcal{P}_{\bowtie\theta}[\varphi]$, we associate an indifference region centered around θ with half-width $\delta(\theta)$. The half-width can be a constant, such as 10^{-1}, but it is sometimes desirable to let the half-width be a function of θ. A reasonable choice in that case is

$$\delta(\theta) = \begin{cases} 2\delta_0\theta & \text{if } \theta \leq 0.5 \\ 2\delta_0(1-\theta) & \text{if } \theta > 0.5 \end{cases}, \tag{5.1}$$

which makes the half-width δ_0 if θ is 0.5 and smaller if θ is close to 0 or 1. We modify the semantics of UTSL to account for indifference regions. This is done by replacing the satisfaction relation $\models$ with two relations $\approx_\top$ and $\approx_\bot$ representing satisfaction and unsatisfaction, respectively, for UTSL formulae when taking indifference regions into account. The relations $\approx_\top$ and $\approx_\bot$ are mutually exclusive, but not exhaustive, so $\approx_\bot$ is not equivalent to $\not\approx_\top$.

Definition 5.1 (UTSL Semantics with Indifference Regions). Let $\mathcal{M} = \langle S, T, \mu, SV, V\rangle$ be a factored stochastic discrete event system, and let $\delta(\theta)$ be a function determining the half-width of an indifference region centered around θ. A satisfaction relation $\approx_\top$ and an unsatisfaction relation $\approx_\bot$ for UTSL with indifference regions are simultaneously defined by induction as follows:

$$
\begin{array}{rl}
\mathcal{M}, \{\langle s_0, t_0\rangle, \ldots, \langle s_k, t_k\rangle\} \approx_\top x \sim v & \text{if } V(s_k, x) \sim v \\
\mathcal{M}, \{\langle s_0, t_0\rangle, \ldots, \langle s_k, t_k\rangle\} \approx_\bot x \sim v & \text{if } V(s_k, x) \not\sim v \\
\mathcal{M}, \sigma_{\leq\tau} \approx_\top \neg\Phi & \text{if } \mathcal{M}, \sigma_{\leq\tau} \approx_\bot \Phi \\
\mathcal{M}, \sigma_{\leq\tau} \approx_\bot \neg\Phi & \text{if } \mathcal{M}, \sigma_{\leq\tau} \approx_\top \Phi \\
\mathcal{M}, \sigma_{\leq\tau} \approx_\top \Phi \wedge \Psi & \text{if } (\mathcal{M}, \sigma_{\leq\tau} \approx_\top \Phi) \wedge (\mathcal{M}, \sigma_{\leq\tau} \approx_\top \Psi) \\
\sigma_{\leq\tau} \approx_\bot \Phi \wedge \Psi & \text{if } (\mathcal{M}, \sigma_{\leq\tau} \approx_\bot \Phi) \vee (\mathcal{M}, \sigma_{\leq\tau} \approx_\bot \Psi) \\
\mathcal{M}, \sigma_{\leq\tau} \approx_\top \mathcal{P}_{\geq\theta}[\varphi] & \text{if } \mu(\{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \models \varphi\}) \geq \theta + \delta(\theta) \\
\mathcal{M}, \sigma_{\leq\tau} \approx_\bot \mathcal{P}_{\geq\theta}[\varphi] & \text{if } \mu(\{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \models \varphi\}) \leq \theta - \delta(\theta) \\
\mathcal{M}, \sigma_{\leq\tau} \approx_\top \mathcal{P}_{\leq\theta}[\varphi] & \text{if } \mu(\{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \models \varphi\}) \leq \theta - \delta(\theta) \\
\mathcal{M}, \sigma_{\leq\tau} \approx_\bot \mathcal{P}_{\leq\theta}[\varphi] & \text{if } \mu(\{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \models \varphi\}) \geq \theta + \delta(\theta)
\end{array}
$$

It should be clear from Definitions 4.2 and 5.1 that, for any UTSL formula Φ, $(\mathcal{M}, \sigma_{\leq\tau} \approx_\top \Phi) \implies (\mathcal{M}, \sigma_{\leq\tau} \models \Phi)$ and $(\mathcal{M}, \sigma_{\leq\tau} \approx_\bot \Phi) \implies (\mathcal{M}, \sigma_{\leq\tau} \not\models \Phi)$. However, the inverse does not hold, in general. For example, it is possible that $\mathcal{M}, \sigma_{\leq\tau} \models \Phi$ is satisfied without $\mathcal{M}, \sigma_{\leq\tau} \approx_\top \Phi$ being so because of the indifference regions. In fact, the triple $\langle \mathcal{M}, \sigma_{\leq\tau}, \mathcal{P}_{\bowtie\theta}[\varphi]\rangle$ does not belong to either of the two relations $\approx_\top$ and $\approx_\bot$ if $\mu(\{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \models \varphi\})$ falls into the indifference region for $\mathcal{P}_{\bowtie\theta}[\varphi]$, i.e. is less than $\delta(\theta)$ away from θ.

Since we are resorting to statistical techniques for solving UTSL model checking problems, we must accept that we sometimes produce an incorrect answer. This is satisfactory, so long as we can guarantee certain *a priori* bounds on the probability of an incorrect result. Simply put, we want the probability of accepting a UTSL formula as true when it is false (or vice versa) to be below a predetermined threshold. To be precise, we want our statistical model checking algorithm to accept a UTSL formula Φ as true with probability at least $1-\alpha$ if $\mathcal{M}, \sigma_{\leq\tau} \approx\!\!\!\!\models_\top \Phi$ holds, and the probability should be at most β that Φ is accepted as true if $\mathcal{M}, \sigma_{\leq\tau} \approx\!\!\!\!\models_\bot \Phi$ holds. Let $\mathcal{M}, \sigma_{\leq\tau} \vdash \Phi$ represent the fact that our model checking algorithm accepts Φ as true, and let $\mathcal{M}, \sigma_{\leq\tau} \nvdash \Phi$ represent the fact that Φ is rejected as false by our algorithm. For the remainder of this section, we will often leave out $\mathcal{M}$ from relations for the sake of brevity. The requirements for our algorithm can then be summarized by the following two conditions:

$$\Pr[\sigma_{\leq\tau} \vdash \Phi \mid \sigma_{\leq\tau} \approx\!\!\!\!\models_\top \Phi] \geq 1-\alpha \tag{5.2}$$

$$\Pr[\sigma_{\leq\tau} \vdash \Phi \mid \sigma_{\leq\tau} \approx\!\!\!\!\models_\bot \Phi] \leq \beta \tag{5.3}$$

We require that our model checking algorithm always produces a result, i.e. it either accepts a UTSL formula as true or rejects it as false. In other words, the algorithm is required to satisfy the condition $\neg(\sigma_{\leq\tau} \vdash \Phi) \iff (\sigma_{\leq\tau} \nvdash \Phi)$. It follows from this requirement that

$$\Pr[\sigma_{\leq\tau} \nvdash \Phi \mid \sigma_{\leq\tau} \approx\!\!\!\!\models_\top \Phi] \leq \alpha \tag{5.4}$$

is equivalent to condition (5.2). The parameter α can be interpreted as a bound on the probability of a type I error (false negative) and β can be thought of a bound on the probability of a type II error (false positive), provided that we do not consider it an error to produce an incorrect answer for a model checking problem when some of the probabilities fall into an indifference region. By narrowing the indifference regions for probabilistic UTSL operators, we can get arbitrarily close to a statistical algorithm that implements the true semantics for UTSL given by Definition 4.2, although this will most certainly come at a cost.

Let us now consider the problem of verifying a UTSL formula Φ relative to a trajectory prefix so that conditions (5.4) and (5.3) are satisfied, under the assumption that Φ does not contain any nested probabilistic operators. To begin with, if Φ is of the form $x \sim v$, then it is trivial to satisfy the two conditions for any α and β. Given a trajectory prefix $\{\langle s_0, t_0\rangle, \ldots, \langle s_k, t_k\rangle\}$, we simply observe the value of x in state s_k and compare it to v. The probability of error in this case is always zero.

5.1.1 Probabilistic Operator

To verify the UTSL formula $\mathcal{P}_{\bowtie\theta}[\varphi]$, we introduce Bernoulli variates X_i with parameter p, as stated in the introduction to this chapter, where p is the probability measure of the set of trajectories that satisfy φ. An observation of X_i can be obtained by first generating a trajectory for $\mathcal{M}$ using discrete event simulation and then verifying φ over the sampled trajectory. If φ does not contain any probabilistic operators, as is assumed for now, then we can verify φ without any uncertainty in the result. If φ is determined to hold over the sampled trajectory, then the observation is 1, otherwise it is 0.

While a trajectory for a stochastic discrete event system can be infinite, we assume that we never need to generate more than a finite prefix of a trajectory in order to determine the truth value of φ over the entire trajectory. If φ is $X^I\ \Phi$, then this assumption holds with certainty because we only need to simulate a single state transition. If φ is $\Phi\ \mathcal{U}^I\ \Psi$, then the assumption holds if the probability is zero that an infinite number of state transitions occur before $\sup I$ time units have passed. A sufficient condition for this to be the case is that the system is non-explosive and $\sup I$ is finite, as is stated by the following theorem.

Theorem 5.1 (Sufficient Conditions for Tractability). *The probability is zero that an infinite trajectory is needed to determine the truth value of* $\Phi\ \mathcal{U}^I\ \Psi$*, with* $\sup I < \infty$*, for a non-explosive stochastic discrete event system.*

Proof. If we have not already encountered a state satisfying $\neg\Phi \vee \Psi$ within the first $\sup I$ time units along a trajectory, then we can conclude that $\Phi\ \mathcal{U}^I\ \Psi$ does not hold without having to look further along the trajectory. If the stochastic discrete event system is non-explosive, then the probability measure is zero for an infinite trajectory $\{\langle s_0, t_0\rangle, \langle s_1, t_1\rangle, \ldots\}$ with $\sum_{i=0}^{\infty} t_i < \infty$. It follows that within a finite interval of time, in particular the interval $[0, \sup I]$, only a finite number of state transitions can occur. Consequently, the probability is one that we can determine the truth value of $\Phi\ \mathcal{U}^I\ \Psi$ by looking at a finite prefix of a trajectory. $\square$

We can now set up a hypothesis testing problem for verifying $\mathcal{P}_{\geq\theta}[\varphi]$. We should test the hypothesis $H_0 : p \geq \theta + \delta(\theta)$ against the alternative hypothesis $H_1 : p \leq \theta - \delta(\theta)$ (for $\mathcal{P}_{\leq\theta}[\varphi]$, we simply reverse the roles of the two hypotheses). The hypothesis H_0 holds if and only if $\sigma_{\leq\tau} \approx_{\top} \mathcal{P}_{\geq\theta}[\varphi]$ holds, and H_1 is similarly related to the judgment $\sigma_{\leq\tau} \approx_{\perp} \mathcal{P}_{\geq\theta}[\varphi]$. Thus, by using an acceptance sampling test with strength $\langle\alpha, \beta\rangle$ to decide $\sigma_{\leq\tau} \vdash \mathcal{P}_{\geq\theta}[\varphi]$, we can satisfy conditions (5.4) and (5.3) with our model checking algorithm.

5.1.2 Composite State Formulae

To complete the model checking algorithm, we need to verify negated UTSL formulae and conjunctions of UTSL formulae. We take a compositional approach to verification of such formulae. To verify $\neg\Phi$, we verify Φ and reverse the result. To verify a conjunction, we verify each conjunct separately. The following two rules formally define the behavior of the model checking algorithm:

$$\mathcal{M}, \sigma_{\leq \mathcal{T}} \vdash \neg\Phi \qquad \text{if } \mathcal{M}, \sigma_{\leq \mathcal{T}} \nvdash \Phi$$

$$\mathcal{M}, \sigma_{\leq \mathcal{T}} \vdash \Phi \wedge \Psi \qquad \text{if } (\mathcal{M}, \sigma_{\leq \mathcal{T}} \vdash \Phi) \wedge (\mathcal{M}, \sigma_{\leq \mathcal{T}} \vdash \Psi)$$

Next, we show how to bound the probability of error for a composite UTSL formula, assuming that we have bounds for the probability of error in the verification results for the subformulae.

First, consider the verification of $\neg\Phi$, assuming we have already verified Φ so that conditions (5.4) and (5.3) are satisfied. Since we negate the verification result for Φ, a type I error for Φ becomes a type II error for $\neg\Phi$, and a type II error for Φ becomes a type I error for $\neg\Phi$. To verify $\neg\Phi$ with error bounds α and β, we therefore have to verify Φ with error bounds β and α as stated by the following proposition.

Proposition 5.2. *To verify $\neg\Phi$ with type I error probability α and type II error probability β, it is sufficient to verify Φ with type I error probability β and type II error probability α.*

Proof. Assume that $\Pr[\sigma_{\leq \mathcal{T}} \nvdash \Phi \mid \sigma_{\leq \mathcal{T}} \approx\!\!\!\!\models_{\top} \Phi] \leq \beta$ and $\Pr[\sigma_{\leq \mathcal{T}} \vdash \Phi \mid \sigma_{\leq \mathcal{T}} \approx\!\!\!\!\models_{\bot} \Phi] \leq \alpha$. It follows from Definition 5.1 that $\sigma_{\leq \mathcal{T}} \approx\!\!\!\!\models_{\top} \Phi \iff \sigma_{\leq \mathcal{T}} \approx\!\!\!\!\models_{\bot} \neg\Phi$ and $\sigma_{\leq \mathcal{T}} \approx\!\!\!\!\models_{\bot} \Phi \iff \sigma_{\leq \mathcal{T}} \approx\!\!\!\!\models_{\top} \neg\Phi$. Our model checking algorithm is such that $\sigma_{\leq \mathcal{T}} \nvdash \Phi \iff \sigma_{\leq \mathcal{T}} \vdash \neg\Phi$ and $\sigma_{\leq \mathcal{T}} \vdash \Phi \iff \sigma_{\leq \mathcal{T}} \nvdash \neg\Phi$. Consequently, $\Pr[\sigma_{\leq \mathcal{T}} \nvdash \neg\Phi \mid \sigma_{\leq \mathcal{T}} \approx\!\!\!\!\models_{\top} \neg\Phi] = \Pr[\sigma_{\leq \mathcal{T}} \vdash \Phi \mid \sigma_{\leq \mathcal{T}} \approx\!\!\!\!\models_{\bot} \Phi] \leq \alpha$ and $\Pr[\sigma_{\leq \mathcal{T}} \vdash \neg\Phi \mid \sigma_{\leq \mathcal{T}} \approx\!\!\!\!\models_{\bot} \neg\Phi] = \Pr[\sigma_{\leq \mathcal{T}} \nvdash \Phi \mid \sigma_{\leq \mathcal{T}} \approx\!\!\!\!\models_{\top} \Phi] \leq \beta$. $\square$

Next, consider the verification of $\Phi \wedge \Psi$. The conjunction is determined to hold by our algorithm if and only if both Φ and Ψ are determined to hold. A type I error occurs if we believe that at least one of Φ and Ψ does not hold, when in reality both are true. A type II error occurs if we believe that both Φ and Ψ hold, when at least one of the conjuncts actually is false. We will show that in order to verify a conjunction with error bounds α and β, it is sufficient to verify each conjunct with the same error bounds. To prove this, we use the following elementary lemma from probability theory.

Lemma 5.3. *For arbitrary events A and B, $\Pr[A \wedge B] \leq \min(\Pr[A], \Pr[B])$.*

Using this lemma, we can derive bounds on the error probabilities associated with the verification of a conjunction based on error bounds for the verification of the individual conjuncts.

Theorem 5.4 (Conjunction). *If Φ_i is verified with type I error probability α_i and type II error probability β_i for all $1 \leq i \leq n$, then $\Phi = \bigwedge_{i=1}^{n} \Phi_i$ can be verified with type I error probability $\min_{i \in Rej(\Phi)} \alpha_i$ and type II error probability $\max_{1 \leq i \leq n} \beta_i$, where $Rej(\bigwedge_{i=1}^{n} \Phi_i) = \{i \mid \sigma_{\leq \tau} \nvdash \Phi_i\}$.*

Proof by induction. If $n = 1$, then $\bigwedge_{i=1}^{n} \Phi_i \equiv \Phi_1$, which by assumption can be verified with type I error probability $\alpha_1 = \min \alpha_1$ and type II error probability $\beta_1 = \max \beta_1$.

Assume that $\Phi = \bigwedge_{i=1}^{n} \Phi_i$, for some $n \geq 1$, can be verified with type I error probability $\alpha = \min_{i \in Rej(\Phi)} \alpha_i$ and type II error probability $\beta = \max_{1 \leq i \leq n} \beta_i$. Furthermore, assume that $\Pr[\sigma_{\leq \tau} \nvdash \Phi_{n+1} \mid \sigma_{\leq \tau} \approx_{\top} \Phi_{n+1}] \leq \alpha_{n+1}$ and $\Pr[\sigma_{\leq \tau} \vdash \Phi_{n+1} \mid \sigma_{\leq \tau} \approx_{\perp} \Phi_{n+1}] \leq \beta_{n+1}$.

It follows from Definition 5.1 that $\sigma_{\leq \tau} \approx_{\top} \Phi \wedge \Phi_{n+1} \iff (\sigma_{\leq \tau} \approx_{\top} \Phi) \wedge (\sigma_{\leq \tau} \approx_{\top} \Phi_{n+1})$. We thus have $\Pr[\sigma_{\leq \tau} \nvdash \Phi \wedge \Phi_{n+1} \mid \sigma_{\leq \tau} \approx_{\top} \Phi \wedge \Phi_{n+1}] = \Pr[\sigma_{\leq \tau} \nvdash \Phi \wedge \Phi_{n+1} \mid (\sigma_{\leq \tau} \approx_{\top} \Phi) \wedge (\sigma_{\leq \tau} \approx_{\top} \Phi_{n+1})]$. There are three ways in which a type I error can occur, i.e. our model checking algorithm can conclude $\sigma_{\leq \tau} \nvdash \Phi \wedge \Phi_{n+1}$:

1. If both Φ and Φ_{n+1} are verified to be false, then $\Pr[\sigma_{\leq \tau} \nvdash \Phi \wedge \Phi_{n+1} \mid (\sigma_{\leq \tau} \approx_{\top} \Phi) \wedge (\sigma_{\leq \tau} \approx_{\top} \Phi_{n+1})] = \Pr[(\sigma_{\leq \tau} \nvdash \Phi) \wedge (\sigma_{\leq \tau} \nvdash \Phi_{n+1}) \mid (\sigma_{\leq \tau} \approx_{\top} \Phi) \wedge (\sigma_{\leq \tau} \approx_{\top} \Phi_{n+1})]$, which by Lemma 5.3 is at most $\min(\Pr[\sigma_{\leq \tau} \nvdash \Phi \mid (\sigma_{\leq \tau} \approx_{\top} \Phi) \wedge (\sigma_{\leq \tau} \approx_{\top} \Phi_{n+1})], \Pr[\sigma_{\leq \tau} \nvdash \Phi_{n+1} \mid (\sigma_{\leq \tau} \approx_{\top} \Phi) \wedge (\sigma_{\leq \tau} \approx_{\top} \Phi_{n+1})])$. By assumption, this is at most $\min(\alpha, \alpha_{n+1}) = \min_{i \in Rej(\Phi \wedge \Phi_{n+1})} \alpha_i$.

2. If Φ is verified to be false and Φ_{n+1} is verified to be true, then $\Pr[\sigma_{\leq \tau} \nvdash \Phi \wedge \Phi_{n+1} \mid (\sigma_{\leq \tau} \approx_{\top} \Phi) \wedge (\sigma_{\leq \tau} \approx_{\top} \Phi_{n+1})] = \Pr[(\sigma_{\leq \tau} \nvdash \Phi) \wedge (\sigma_{\leq \tau} \vdash \Phi_{n+1}) \mid (\sigma_{\leq \tau} \approx_{\top} \Phi) \wedge (\sigma_{\leq \tau} \approx_{\top} \Phi_{n+1})] \leq \min(\alpha, 1) = \min_{i \in Rej(\Phi \wedge \Phi_{n+1})} \alpha_i$.

3. If Φ is verified to be true and Φ_{n+1} is verified to be false, then $\Pr[\sigma_{\leq \tau} \nvdash \Phi \wedge \Phi_{n+1} \mid (\sigma_{\leq \tau} \approx_{\top} \Phi) \wedge (\sigma_{\leq \tau} \approx_{\top} \Phi_{n+1})] = \Pr[(\sigma_{\leq \tau} \vdash \Phi) \wedge (\sigma_{\leq \tau} \nvdash \Phi_{n+1}) \mid (\sigma_{\leq \tau} \approx_{\top} \Phi) \wedge (\sigma_{\leq \tau} \approx_{\top} \Phi_{n+1})] \leq \min(1, \alpha_{n+1}) = \alpha_{n+1}$. If Φ is verified as true, then $Rej(\Phi) = \emptyset$. We therefore have $\alpha_{n+1} = \min_{i \in Rej(\Phi \wedge \Phi_{n+1})} \alpha_i$.

In all three cases the probability of a type I error is bounded by $\min_{i \in Rej(\Phi \wedge \Phi_{n+1})} \alpha_i$ as required.

Our model checking algorithm will conclude $\sigma_{\leq \tau} \vdash \Phi \wedge \Phi_{n+1}$ if and only if it can conclude both $\sigma_{\leq \tau} \vdash \Phi$ and $\sigma_{\leq \tau} \vdash \Phi_{n+1}$. There are three ways in which this can lead to a type II error:

1. If both $\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi$ and $\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi_{n+1}$ hold, then the probability of a type II error is $\Pr[(\sigma_{\leq\mathcal{T}} \vdash \Phi) \wedge (\sigma_{\leq\mathcal{T}} \vdash \Phi_{n+1}) \mid (\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi) \wedge (\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi_{n+1})]$. This is at most $\min(\Pr[\sigma_{\leq\mathcal{T}} \vdash \Phi \mid (\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi) \wedge (\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi_{n+1})], \Pr[\sigma_{\leq\mathcal{T}} \vdash \Phi_{n+1} \mid (\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi) \wedge (\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi_{n+1})])$ by Lemma 5.3, which in turn is at most $\min(\beta, \beta_{n+1})$ by assumption.

2. If $\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi$ holds, but not $\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi_{n+1}$, then the probability of a type II error is $\Pr[(\sigma_{\leq\mathcal{T}} \vdash \Phi) \wedge (\sigma_{\leq\mathcal{T}} \vdash \Phi_{n+1}) \mid \sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi] \leq \min(\beta, 1) = \beta$.

3. If $\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi_{n+1}$ holds, but not $\sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi$, then the probability of a type II error is $\Pr[(\sigma_{\leq\mathcal{T}} \vdash \Phi) \wedge (\sigma_{\leq\mathcal{T}} \vdash \Phi_{n+1}) \mid \sigma_{\leq\mathcal{T}} \not\approx_{\perp} \Phi_{n+1}] \leq \min(1, \beta_{n+1}) = \beta_{n+1}$.

We take the maximum over the three cases to obtain the bound $\max_{1\leq i\leq n+1} \beta_i$. □

Intuitively, we can explain Theorem 5.4 as follows. To conclude that $\Phi \wedge \Psi$ does not hold, we only need to be convinced that one of the conjuncts does not hold. We can base the decision for the conjunction solely on the rejection of a single conjunct, in which case the probability of a type I error will be the same for the conjunction as for the rejected conjunct. We get $\min_{i\in Rej(\Phi)} \alpha_i$ by basing our decision for the entire conjunction on the conjunct that has been verified with the smallest probability of a type I error. To conclude that $\Phi \wedge \Psi$ holds, we must be convinced that both conjuncts hold. We get a type II error if at least one of the conjuncts does not hold and we accept the conjunction as true. If Φ does not hold, the probability of a type II error for the conjunction is bounded by the type II error probability for Φ. If Ψ does not hold, the type II error probability for Ψ bounds the type II error probability for the conjunction. Since we cannot know if either Φ or Ψ is actually false, we know only that the type II error probability is at most the maximum of the type II error probabilities for the conjuncts.

If we knew that the verification results for the individual conjuncts were obtained independently, then we could actually bound the type I error probability for the verification of the conjunction by $\prod_{i\in Rej(\Phi)} \alpha_i$, but Theorem 5.4 does not make any assumptions regarding independence. For example, if the same set of sampled trajectories were used to verify all of the conjuncts, then the verification results for the individual conjuncts would not be independent.

Example 5.1. Consider the UTSL formula $\Phi = \mathcal{P}_{\geq 0.5}[\varphi_1] \wedge \mathcal{P}_{\geq 0.75}[\varphi_2]$. Let $\alpha_1 = 0.01$ and $\beta_1 = 0.04$ be the error bounds used to verify the first conjunct, and let $\alpha_2 = 0.03$ and $\beta_2 = 0.02$ be the error bounds used

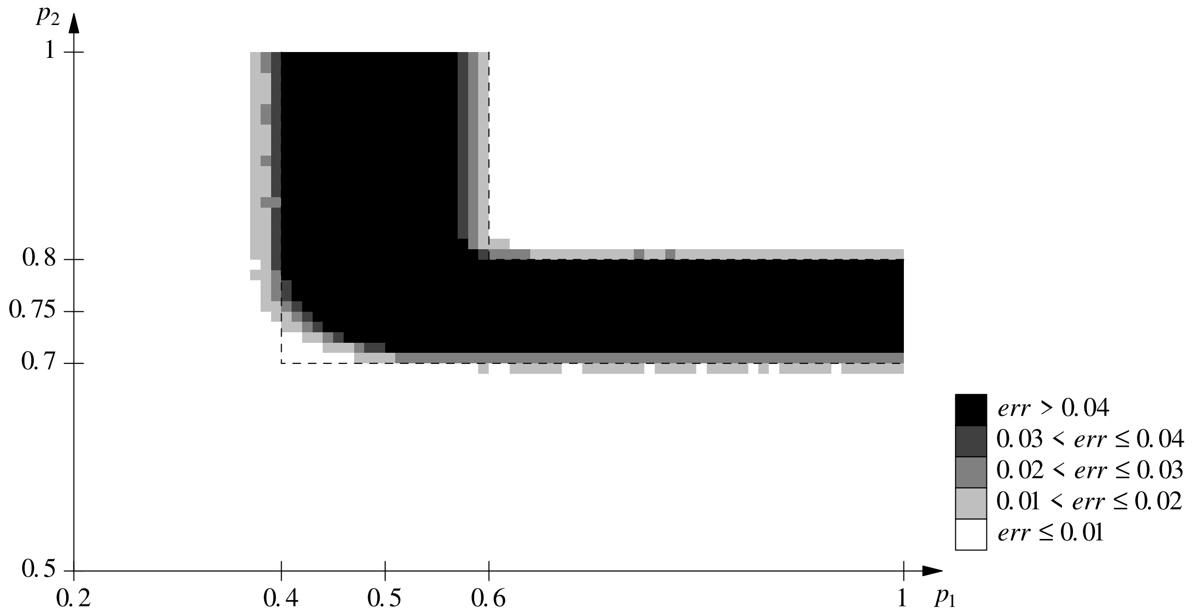


Figure 5.1: Probability of an incorrect verification result for a conjunction $\Phi = \mathcal{P}_{\geq 0.5}[\varphi_1] \wedge \mathcal{P}_{\geq 0.75}[\varphi_2]$ as a function of the probabilities p_1 and p_2 that φ_1 and φ_2, respectively, hold over trajectories starting in some initial state s_0. The error bounds are $\alpha_1 = 0.01$, $\beta_1 = 0.04$, $\alpha_2 = 0.03$ and $\beta_2 = 0.02$. The border of the L-shaped indifference region is indicated by dashed lines. The plot was obtained using computer simulation, with 50,000 runs per data point.

to verify the second conjunct. Furthermore, let p_i be the probability measure of the set of trajectories that start in s_0 at time 0 and satisfy φ_i, for $i \in \{1, 2\}$. We assume that the function in (5.1), with $\delta_0 = 0.1$, is used to determine the indifference region for each probabilistic operator. This gives the indifference region $(0.4, 0.6)$ for the first conjunct and $(0.7, 0.8)$ for the second conjunct. According to Theorem 5.4, if $p_1 > 0.6$ and $p_2 > 0.8$, then the probability of rejecting the conjunction as false is at most $\min(\alpha_1, \alpha_2) = 0.03$. On the other hand, if $p_1 < 0.4$ or $p_2 < 0.7$, then the probability of accepting the conjunction as true is at most $\max(\beta_1, \beta_2) = 0.04$. Figure 5.1 plots the probability of incorrectly verifying the given conjunction as a function of p_1 and p_2. The simulation results confirm that, while the probability of error is large inside of the L-shaped indifference region, the error bounds are respected outside of the indifference region.

The following result follows immediately from Theorem 5.4 and establishes the procedure for the verification of a conjunction, namely that we use the target error bounds for the conjunction when verifying the individual conjuncts.

Corollary. *To verify $\bigwedge_{i=1}^{n} \Phi_i$ with type I error probability α and type II error probability β, it is sufficient* *to verify each conjunct Φ_i with type I error probability α and type II error probability β.*

We have now shown how to verify a UTSL formula without nested probabilistic operators so that conditions (5.4) and (5.3) are satisfied. An observation is obtained by generating a trajectory using discrete event simulation and verifying the path formula over the generated trajectory. To verify a negation, we verify the negated UTSL formula while reversing the role of the error bounds. A conjunction is verified by verifying each conjunct using the same error bounds as intended for the conjunction (note that the fact that we can use the same error bounds to verify the individual conjuncts will prove essential when dealing with nested probabilistic operators). For probabilistic operators, we can use one of the acceptance sampling tests described in Section 2.2. In the next chapter, we present empirical results for our model checking algorithm using two different tests: the sequential version of a single sampling plan and Wald's sequential probability ratio test.

5.2 Model Checking with Nested Probabilistic Operators

In this section, we consider UTSL formulae with nested probabilistic operators. If a path formula contains probabilistic operators, we can no longer assume that it can be verified without error. To deal with the possibility of making an error in verifying a path formula, we modify the semantics given in Definition 5.1.

Definition 5.2 (UTSL Semantics with Indifference Regions and Nesting). Let $\mathcal{M} = \langle S, T, \mu, SV, V \rangle$ be a factored stochastic discrete event system, and let $\delta(\theta)$ be a function determining the half-width of an indifference region centered around θ. A satisfaction relation $\approx_\top$ and an unsatisfaction relation $\approx_\bot$ for UTSL with indifference regions and nested probabilistic operators are simultaneously defined by induction as follows (the first six rules are the same as in Definition 5.1 and are therefore not repeated here):

$$\vdots$$

$$\begin{array}{ll}
\mathcal{M}, \sigma_{\leq\tau} \approx_\top \mathcal{P}_{\geq\theta}[\varphi] & \text{if } \mu(\{\sigma \in Path(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \approx_\top \varphi\}) \geq \theta + \delta(\theta) \\
\mathcal{M}, \sigma_{\leq\tau} \approx_\bot \mathcal{P}_{\geq\theta}[\varphi] & \text{if } \mu(\{\sigma \in Path(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \approx_\bot \varphi\}) \geq 1 - (\theta - \delta(\theta)) \\
\mathcal{M}, \sigma_{\leq\tau} \approx_\top \mathcal{P}_{\leq\theta}[\varphi] & \text{if } \mu(\{\sigma \in Path(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \approx_\top \varphi\}) \leq \theta - \delta(\theta) \\
\mathcal{M}, \sigma_{\leq\tau} \approx_\bot \mathcal{P}_{\leq\theta}[\varphi] & \text{if } \mu(\{\sigma \in Path(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \approx_\bot \varphi\}) \leq 1 - (\theta + \delta(\theta))
\end{array}$$

$$
\begin{array}{ll}
\mathcal{M}, \sigma, \tau \approx_{\top} X^{I}\,\Phi & \text{if } \exists k \in \mathbb{N}.\big((T_{k-1} \leq \tau) \wedge (\tau < T_k) \wedge (T_k - \tau \in I) \wedge (\mathcal{M}, \sigma_{\leq T_k} \approx_{\top} \Phi)\big) \\
\mathcal{M}, \sigma, \tau \approx_{\perp} X^{I}\,\Phi & \text{if } \forall k \in \mathbb{N}.\big(\big((T_{k-1} \leq \tau) \wedge (\tau < T_k) \wedge (T_k - \tau \in I)\big) \rightarrow (\mathcal{M}, \sigma_{\leq T_k} \approx_{\perp} \Phi)\big) \\
\mathcal{M}, \sigma, \tau \approx_{\top} \Phi\,\mathcal{U}^{I}\,\Psi & \text{if } \exists t \in I.\big((\mathcal{M}, \sigma_{\leq \tau + t} \approx_{\top} \Psi) \wedge \forall t' \in T.\big((t' < t) \rightarrow (\mathcal{M}, \sigma_{\leq \tau + t'} \approx_{\top} \Phi)\big)\big) \\
\mathcal{M}, \sigma, \tau \approx_{\perp} \Phi\,\mathcal{U}^{I}\,\Psi & \text{if } \forall t \in I.\big((\mathcal{M}, \sigma_{\leq \tau + t} \approx_{\perp} \Psi) \vee \exists t' \in T.\big((t' < t) \wedge (\mathcal{M}, \sigma_{\leq \tau + t'} \approx_{\perp} \Phi)\big)\big)
\end{array}
$$

Definition 5.2 is equivalent to Definition 5.1 for UTSL formulae that do not have nested probabilistic operators, so the semantics just given can be used even without nested probabilistic operators. To prove this, we first show that for UTSL formulae free of any probabilistic operators, the relations $\approx_{\top}$ and $\approx_{\perp}$ are equivalent to $\models$ and $\not\models$, respectively.

Lemma 5.5. *If Φ is a UTSL formula that does not contain any probabilistic operators, then $(\mathcal{M}, \sigma_{\leq \tau} \approx_{\top} \Phi) \iff (\mathcal{M}, \sigma_{\leq \tau} \models \Phi)$ and $(\mathcal{M}, \sigma_{\leq \tau} \approx_{\perp} \Phi) \iff (\mathcal{M}, \sigma_{\leq \tau} \not\models \Phi)$.*

Proof by structural induction. If Φ is $x \sim v$, then the two equivalences follow immediately from Definitions 4.2 and 5.2. If the UTSL formula is $\neg\Phi$ or $\Phi \wedge \Psi$ where Φ and Ψ are free of any probabilistic operators, assume that the equivalences hold for Φ and Ψ. It follows from Definitions 4.2 and 5.2 that the equivalences hold for the compound UTSL formulae. This covers all UTSL formulae that can be formed without any probabilistic operators according to Definition 4.1. □

Proposition 5.6. *For UTSL formulae that do not contain nested probabilistic operators, Definitions 5.1 and 5.2 are equivalent.*

Proof. The first six rules are identical for the two definitions. It follows from Lemma 5.5 that the rules for path formulae are equivalent to the rules in Definition 4.2, which Definition 5.1 inherits, because the path formulae are assumed not to contain probabilistic operators. From this, it follows that the sets $\{\sigma \in \mathit{Path}(\sigma_{\leq \tau}) \mid \mathcal{M}, \sigma, \tau \approx_{\top} \varphi\}$ and $\{\sigma \in \mathit{Path}(\sigma_{\leq \tau}) \mid \mathcal{M}, \sigma, \tau \models \varphi\}$ are equivalent. The rules for $\mathcal{M}, \sigma_{\leq \tau} \approx_{\top} \mathcal{P}_{\bowtie \theta}[\varphi]$ are therefore equivalent for the two definitions. Analogously, the sets $\{\sigma \in \mathit{Path}(\sigma_{\leq \tau}) \mid \mathcal{M}, \sigma, \tau \approx_{\perp} \varphi\}$ and $\{\sigma \in \mathit{Path}(\sigma_{\leq \tau}) \mid \mathcal{M}, \sigma, \tau \not\models \varphi\}$ are equivalent. Since $\mathcal{M}, \sigma, \tau \not\models \varphi$ is equivalent to $\neg(\mathcal{M}, \sigma, \tau \models \varphi)$, we have $\mu(\{\sigma \in \mathit{Path}(\sigma_{\leq \tau}) \mid \mathcal{M}, \sigma, \tau \approx_{\perp} \varphi\}) = 1 - \mu(\{\sigma \in \mathit{Path}(\sigma_{\leq \tau}) \mid \mathcal{M}, \sigma, \tau \models \varphi\})$. Hence, the rules for $\mathcal{M}, \sigma_{\leq \tau} \approx_{\perp} \mathcal{P}_{\bowtie \theta}[\varphi]$ are also equivalent for the two definitions, and this covers all rules. □

It is still the case that $\mathcal{M}, \sigma_{\leq\tau} \approx\!\!\!\models_{\top} \Phi$ implies $\mathcal{M}, \sigma_{\leq\tau} \models \Phi$ and $\mathcal{M}, \sigma_{\leq\tau} \approx\!\!\!\models_{\perp} \Phi$ implies $\mathcal{M}, \sigma_{\leq\tau} \not\models \Phi$. We want our model checking algorithm to satisfy conditions (5.4) and (5.3) for the modified definition of the relations $\approx\!\!\!\models_{\top}$ and $\approx\!\!\!\models_{\perp}$ for UTSL formulae. Negation and conjunction can be handled in the same way as before, because the definition is unmodified in these cases, but probabilistic statements must now be handled differently.

5.2.1 Probabilistic Operator

Consider the UTSL model checking problem $\langle \mathcal{M}, \sigma_{\leq\tau}, \mathcal{P}_{\geq\theta}[\varphi] \rangle$ (the case $\mathcal{P}_{\leq\theta}[\varphi]$ is analogous), and let $p = \mu(\{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \approx\!\!\!\models_{\top} \varphi\})$ and $q = \mu(\{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \approx\!\!\!\models_{\perp} \varphi\})$. The two conditions (5.4) and (5.3) can then be expressed as $\Pr[\sigma_{\leq\tau} \nvdash \mathcal{P}_{\geq\theta}[\varphi] \mid p \geq \theta + \delta(\theta)] \leq \alpha$ and $\Pr[\sigma_{\leq\tau} \vdash \mathcal{P}_{\geq\theta}[\varphi] \mid q \geq 1 - (\theta - \delta(\theta))] \leq \beta$, respectively. If these conditions are satisfied, then we accept $\mathcal{P}_{\geq\theta}[\varphi]$ with probability at least $1 - \alpha$ if $p \geq \theta + \delta(\theta)$ and with probability at most β if $q \geq 1 - (\theta - \delta(\theta))$. If $p \geq \theta + \delta(\theta)$, then $\mu(\{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \models \varphi\}) \geq \theta$ definitely holds, so there is a high probability of accepting $\mathcal{P}_{\geq\theta}[\varphi]$ when it holds with some margin. Conversely, if $q \geq 1 - (\theta - \delta(\theta))$, then $\mu(\{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \models \varphi\}) \leq \theta$ definitely holds, so $\mathcal{P}_{\geq\theta}[\varphi]$ is rejected with high probability when it is false with some margin.

We want to use acceptance sampling, as before, to verify probabilistic statements. With probabilistic operators in the path formulae, it is possible that observations we use for the acceptance sampling test are incorrect. If we can at least bound the probability of a path formula being incorrectly verified, then we can modify the acceptance sampling test to account for the possibility of observation errors. In particular, we assume that $\Pr[\sigma, \tau \nvdash \varphi \mid \sigma, \tau \approx\!\!\!\models_{\top} \varphi] \leq \alpha'$ and $\Pr[\sigma, \tau \vdash \varphi \mid \sigma, \tau \approx\!\!\!\models_{\perp} \varphi] \leq \beta'$ for some α' and β'. To understand the general theoretical results presented below regarding acceptance sampling with observation error, it may help to have the following interpretation for the random variables X and Y in mind:

$$
\begin{aligned}
Y = 1 &\iff \mathcal{M}, \sigma, \tau \vdash \varphi & \qquad X = 1 &\iff \mathcal{M}, \sigma, \tau \approx\!\!\!\models_{\top} \varphi \\
Y = 0 &\iff \mathcal{M}, \sigma, \tau \nvdash \varphi & \qquad X = 0 &\iff \mathcal{M}, \sigma, \tau \approx\!\!\!\models_{\perp} \varphi
\end{aligned}
$$

Note that Y has exactly two outcomes and is therefore a Bernoulli variate, but X can have more than two outcomes. Before establishing a modified acceptance sampling test, we need the following intermediate result regarding two arbitrary random variables X and Y with some correlation between their observations.

Lemma 5.7. *Let X and Y be two random variables such that* $\Pr[Y = 0 \mid X = 1] \leq \alpha'$ *and* $\Pr[Y = 1 \mid X = 0] \leq \beta'$. *If* $\Pr[X = 1] = p$ *and* $\Pr[X = 0] = q$, *then* $p(1 - \alpha') \leq \Pr[Y = 1] \leq 1 - q(1 - \beta')$.

Proof. By the formula of total probability we have

$$\begin{aligned}\Pr[Y = 1] &= \Pr[X = 1] \Pr[Y = 1 \mid X = 1] + \Pr[X = 0] \Pr[Y = 1 \mid X = 0] \\ &\quad + \Pr[X \notin \{0,1\}] \Pr[Y = 1 \mid X \notin \{0,1\}] \\ &= p(1 - \Pr[Y = 0 \mid X = 1]) + q \Pr[Y = 1 \mid X = 0] + (1 - p - q) \Pr[Y = 1 \mid X \notin \{0,1\}] \ .\end{aligned}$$

As an upper bound for $\Pr[Y = 1]$, we get $\Pr[Y = 1] \leq p(1-0) + q\beta' + (1-p-q) \cdot 1 = 1 - q(1-\beta')$. The lower bound for $\Pr[Y = 1]$ is derived as follows: $\Pr[Y = 1] \geq p(1-\alpha') + q \cdot 0 + (1-p-q) \cdot 0 = p(1-\alpha')$. $\square$

We can now show that with the observation error bounded by α' and β', it is sufficient to replace the probability thresholds p_0 and p_1 of a standard acceptance sampling test with $p_0(1-\alpha')$ and $1-(1-p_1)(1-\beta')$, respectively. In effect, this means that we narrow the indifference region for the acceptance sampling test in order to cope with the possibility of inaccurate observations.

Theorem 5.8 (Acceptance Sampling with Observation Error). *Let Y be a Bernoulli variate whose observations are related to the observations of a random variable X as follows:* $\Pr[Y = 0 \mid X = 1] \leq \alpha'$ *and* $\Pr[Y = 1 \mid X = 0] \leq \beta'$. *Furthermore, let* $\Pr[X = 1] = p$, $\Pr[X = 0] = q$, *and* $\Pr[Y = 1] = p'$. *To test the hypothesis $H_0 : p \geq p_0$ against the alternative hypothesis $H_1 : q \geq 1 - p_1$, for probability thresholds $p_0 > p_1$, so that the probability of accepting H_1 when H_0 holds (type I error) is at most α and the probability of accepting H_0 when H_1 holds (type II error) is at most β, it is sufficient to test $H_0' : p' \geq p_0(1 - \alpha')$ against $H_1' : p' \leq 1 - (1 - p_1)(1 - \beta')$ with probability at most α that H_1' is accepted when H_0 holds and probability at most β that H_0' is accepted when H_1 holds, provided that acceptance of H_0' leads to acceptance of H_0 and acceptance of H_1' leads to acceptance of H_1.*

Proof. From (2.3), assuming a single sampling plan $\langle n, c \rangle$ is used, we get $F(c; n, p')$ as the probability of accepting hypothesis H_1'. We know from Lemma 5.7 that $p' \geq p(1 - \alpha')$. Since $F(c; n, p)$ is a non-increasing function of p in the interval $[0,1]$, we have $F(c; n, p') \leq F(c; n, p(1-\alpha'))$, which if $H_0 : p \geq p_0$ holds is at most $F(c; n, p_0(1 - \alpha'))$. By choosing n and c so that $F(c; n, p_0(1 - \alpha')) \leq \alpha$, we ensure that the probability of accepting H_1', and therefore also H_1, is at most α when H_0 holds.

The probability of accepting H_0' is $1 - F(c; n, p')$ when using the single sampling plan $\langle n, c \rangle$. It follows from Lemma 5.7 that $p' \leq 1 - q(1 - \beta')$. Thus, $1 - F(c; n, p') \leq 1 - F(c; n, 1 - q(1 - \beta'))$, which in turn is at most $1 - F(c; n, 1 - (1 - p_1)(1 - \beta'))$ if $H_1 : q \geq 1 - p_1$ holds. Consequently, if we choose n and c so that $1 - F(c; n, 1 - (1 - p_1)(1 - \beta')) \leq \beta$, we are guaranteed that the probability of accepting H_0', and therefore also H_0, is at most β when H_1 holds. □

The above proof establishes Theorem 5.8 specifically for single sampling plans, but the result is more general because we only need to modify the probability thresholds in order to cope with observation error while leaving the rest of the test procedure intact. We can use the exact same modification for other acceptance sampling tests, for example Wald's sequential probability ratio test. Note that the probability thresholds equal p_0 and p_1 if the observation error is zero, so the modified test is identical to the original test in that case, as should be expected. Note also that the observation error can be chosen independently of the desired strength of the test. A procedure for verifying probabilistic UTSL formulae with nested probabilistic operators follows immediately from Theorem 5.8.

Corollary. *An acceptance sampling test with strength $\langle \alpha, \beta \rangle$ and probability thresholds $(\theta + \delta(\theta))(1 - \alpha')$ and $1 - (1 - (\theta - \delta(\theta)))(1 - \beta')$ can be used to verify $\mathcal{P}_{\geq\theta}[\varphi]$ with type I error probability α and type II error probability β, provided that φ can be verified over trajectories with type I error probability α' and type II error probability β'.*

To better understand the verification procedure for the UTSL formula $\mathcal{P}_{\geq\theta}[\varphi]$ with nested probabilistic operators, consider the following four sets of trajectories:

$$P = \{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \models \varphi\} \qquad Q = \{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \not\models \varphi\}$$
$$\tilde{P} = \{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \mathrel{|\!\!\approx_\top} \varphi\} \qquad \tilde{Q} = \{\sigma \in \mathit{Path}(\sigma_{\leq\tau}) \mid \mathcal{M}, \sigma, \tau \mathrel{|\!\!\approx_\bot} \varphi\}$$

We cannot determine membership in P or Q for a sampled trajectory $\sigma \in \mathit{Path}(\sigma_{\leq\tau})$ if φ contains probabilistic operators. We assume, however, that we have a probabilistic procedure for determining membership in $\tilde{P}$ or $\tilde{Q}$. We require a probability of at most α' that σ is determined to be in $\tilde{Q}$ if it is really in $\tilde{P}$, and the probability of determining that σ is in $\tilde{P}$ should be at most β' if σ is actually in $\tilde{Q}$. Given such a procedure, Theorem 5.8 provides us with a way to test the hypothesis $H_0 : \mu(\tilde{P}) \geq \theta + \delta(\theta)$ against the alternative hypothesis $H_1 : \mu(\tilde{Q}) \geq 1 - (\theta - \delta(\theta))$. Acceptance of H_0 leads to acceptance of $\mathcal{P}_{\geq\theta}[\varphi]$ as true, and

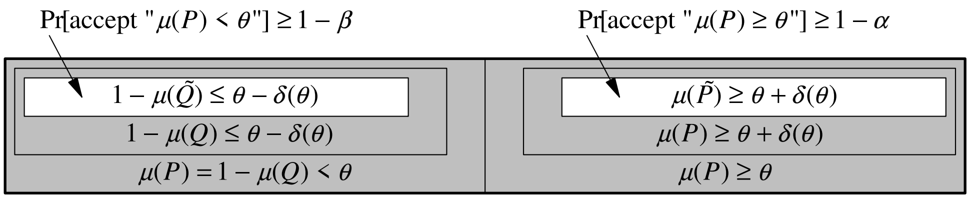


Figure 5.2: Probabilistic guarantees for model checking problems with UTSL formulae of the form $\mathcal{P}_{\geq\,\theta}[\varphi]$, with probabilistic operators in φ. The thick box represents all such model checking problems. In the right half are problems with an affirmative answer. A subset of these problems have an affirmative answer even with an indifference region at the top level of half-width $\delta(\theta)$. For some of the latter set of problems, the UTSL formula holds with indifference regions at all levels. It is for this last set of problems that we can guarantee an affirmative answer with probability at least $1 - \alpha$. There is a similar hierarchy for the problems with a negative answer, in the left half of the thick box. The gray area represents the set of model checking problems for which we give no correctness guarantees.

acceptance of H_1 leads to rejection of $\mathcal{P}_{\geq\,\theta}[\varphi]$ as false. We are guaranteed that H_0 is accepted with probability at least $1 - \alpha$ if H_0 holds. Since $\tilde{P} \subset P$, we know that $\mu(P) \geq \theta$ when H_0 holds, so there is a high probability of accepting $\mathcal{P}_{\geq\,\theta}[\varphi]$ when it holds with some margin. We also know that H_1 is accepted with probability at least $1 - \beta$ if H_1 holds, and $\mu(P) < \theta$ in that case, so there is a high probability of rejecting $\mathcal{P}_{\geq\,\theta}[\varphi]$ when it is false with some margin.

Figure 5.2 gives a graphical representation of the correctness guarantees provided by the algorithm for UTSL formulae with nested probabilistic operators. For the subset of all model checking problems such that $\mu(\tilde{P}) \geq \theta + \delta(\theta)$, it is guaranteed that an affirmative answer is given with probability at least $1 - \alpha$. For the problems such that $1 - \mu(\tilde{Q}) \leq \theta - \delta(\theta)$, it is guaranteed that a negative answer is given with probability at least $1 - \beta$. For the remaining problems, no guarantees are made regarding the correctness of the result.

5.2.2 Path Formulae with Probabilistic Operators

We have established a procedure for verifying probabilistic statements when the path formula cannot be verified without some probability of error. It remains to show how to verify path formulae containing probabilistic operators so that the following conditions are satisfied:

$$\Pr[\sigma, \tau \nvdash \varphi \mid \sigma, \tau \mathrel{|}\!\!\approx_{\top} \varphi] \leq \alpha' \tag{5.5}$$

$$\Pr[\sigma, \tau \vdash \varphi \mid \sigma, \tau \mathrel{|}\!\!\approx_{\bot} \varphi] \leq \beta' \tag{5.6}$$

This is straightforward for $X^I\ \Phi$. We simulate a single state transition and verify Φ in the resulting state.

Path formulae of the form $\Phi\ \mathcal{U}^I\ \Psi$ are more difficult to handle. We need to find a $t \in I$ such that Ψ is satisfied at time t and Φ is satisfied at all time points t' prior to t. Examples 4.1 and 4.2 showed that it is not sufficient to consider only the time points at which a state transition occurs for models that do not satisfy the Markov property. However, if the model is a Markov process, then it is sufficient to consider the time points at which state transitions occur, as mentioned in Chapter 4. This is guaranteed to be a finite number of time points if the assumptions of Theorem 5.1 are satisfied. The same can be said for any discrete-time model, provided that $\sup I$ is finite. If only a finite number of time points need be considered, then we can treat the verification of $\Phi \,\mathcal{U}^I\, \Psi$ as a large disjunction of conjunctions. Let $\{t_1, \ldots, t_n\}$ be the set of time points at which we may have to verify the subformulae, with $t_i \leq \sup I$. For Markov processes, these are the time points at which state transitions occur, and for discrete-time models these are all time points no later than $\sup I$. Furthermore, let t_{n+1} be some time point later than $\sup I$. We can verify $\Phi \,\mathcal{U}^I\, \Psi$ as follows:

$$\sigma, \tau \vdash \Phi \,\mathcal{U}^I\, \Psi \qquad \text{if } \bigvee_{i=1}^{n} \Big((t_i \geq \tau) \wedge \big([t_i, t_{i+1}) \cap I \neq \emptyset\big) \wedge (\sigma_{\leq t_i} \vdash \Psi) \wedge \big((t_i \in I) \vee (\sigma_{\leq t_i} \vdash \Phi)\big) \wedge \bigwedge_{j=1}^{i-1} (\sigma_{\leq t_j} \vdash \Phi) \Big)$$

Since disjunction can be expressed using conjunction and negation, and we already know how to verify negations and conjunctions using statistical techniques, this gives us a way to verify $\Phi \,\mathcal{U}^I\, \Psi$ so that conditions (5.5) and (5.6) are satisfied. Thus, it is sufficient simply to verify the UTSL formulae Φ and Ψ with error bounds α' and β' at each relevant time point along a trajectory.

5.2.3 Observation Error

A noteworthy consequence of Theorem 5.8 is that the bounds on the observation error, α' and β', can be chosen independently of the bounds on the probability of a verification error occurring, α and β. We can decrease α' and β' to increase the indifference region of the outer probabilistic statement and therefore lower the sample size required to verify this part of the formula, but this will increase the effort required per observation, since we have to verify the nested probabilistic statements with higher accuracy. If we increase α' and β' to lower the effort per observation, then we need to make more observations. Clearly, there is a tradeoff here, and the choice for the bounds on the observation error can have a great impact on performance.

Ideally, we want to use the observation error that minimizes the expected verification effort, but this quantity is non-trivial to compute. We propose, instead, a heuristic estimate of the verification effort that can be computed efficiently.

Definition 5.3 (Estimated Effort Heuristic). Let $n(p_0, p_1, \alpha, \beta)$ be the expected sample size of an acceptance sampling test with strength $\langle \alpha, \beta \rangle$ for probability thresholds p_0 and p_1, and let q be the expected number of state transitions within a unit interval of time. We define a heuristic estimate of the effort required to verify a UTSL formula inductively as follows:

$$\begin{aligned}
\mathit{effort}(x \sim v, \alpha, \beta, \alpha', \beta') &= 1 \\
\mathit{effort}(\neg\Phi, \alpha, \beta, \alpha', \beta') &= \mathit{effort}(\Phi, \alpha, \beta, \alpha', \beta') \\
\mathit{effort}(\Phi \wedge \Psi, \alpha, \beta, \alpha', \beta') &= \mathit{effort}(\Phi, \alpha, \beta, \alpha', \beta') + \mathit{effort}(\Psi, \alpha, \beta, \alpha', \beta') \\
\mathit{effort}(\mathcal{P}_{\bowtie\theta}[\varphi], \alpha, \beta, \alpha', \beta') &= n((\theta + \delta(\theta))(1 - \alpha'), 1 - (1 - (\theta - \delta(\theta)))(1 - \beta'), \alpha, \beta) \\
&\quad \cdot \min_{\alpha'', \beta''} \mathit{effort}(\varphi, \alpha', \beta', \alpha'', \beta'') \\
\mathit{effort}(X^I\ \Phi, \alpha, \beta, \alpha', \beta') &= \mathit{effort}(\Phi, \alpha, \beta, \alpha', \beta') \\
\mathit{effort}(\Phi\ \mathcal{U}^I\ \Psi, \alpha, \beta, \alpha', \beta') &= q \cdot \sup I \cdot \mathit{effort}(\Phi, \alpha, \beta, \alpha', \beta') \\
&\quad + q \cdot (\sup I - \inf I) \cdot \mathit{effort}(\Psi, \alpha, \beta, \alpha', \beta')
\end{aligned}$$

For discrete-time models, we set q to 1, and for continuous-time Markov processes, q can be set to the maximum exit rate of the model. The quantity $n(p_0, p_1, \alpha, \beta)$ depends on the acceptance sampling test that is used to verify probabilistic properties. If we use a single sampling plan, then we can compute n exactly using Algorithm 2.1 or approximately using (2.8). Estimating the effort of verifying the UTSL formula $\mathcal{P}_{\bowtie\theta}[\varphi]$ when using a sequential sampling plan is trickier because the expected sample size is a function of the unknown probability measure p of the set of trajectories satisfying φ. It may be reasonable to minimize the worst-case estimated effort. For Wald's sequential probability ratio test, we can use the value of $\tilde{E}_p$ for s given in Table 2.3.

The observation error can obviously not be set to zero, but there is an upper bound as well because the width of the indifference region for an acceptance sampling test must be positive. In the case of acceptance sampling with observation error, the condition $1 - (1 - p_1)(1 - \beta') < p_0(1 - \alpha')$ must be satisfied. From this condition, we can derive an upper bound on the symmetric observation error ($\alpha' = \beta'$):

$$1 - (1 - p_1)(1 - \alpha') < p_0(1 - \alpha') \implies 1 < (1 + p_0 - p_1)(1 - \alpha') \implies \alpha' < \frac{p_0 - p1}{1 + (p_0 - p1)} \tag{5.7}$$

The difference $p_0 - p_1$ is the intended width of the indifference region with zero observation error, which in our case equals $2\delta(\theta)$. We can therefore write (5.7) as $\alpha' < \delta(\theta)/(0.5 + \delta(\theta))$.

Example 5.2. With $p_0 = 0.91$ and $p_1 = 0.89$, the maximum symmetric observation error is $0.02/1.02 \approx 0.0196$ according to (5.7). This means that the probability of error must be no more than 0.0196 for each individual observation when using an indifference region of width 0.02.

We can find the optimal symmetric observation error for each probabilistic operator of a UTSL formula using numerical function minimization and systematically working our way outward from the innermost probabilistic operators. For the innermost probabilistic operators, we can use zero observation error because their path formulae do not contain any probabilistic operators. We can find the optimal symmetric observation error for the remaining probabilistic operators by searching for the value of $x = \alpha' = \beta'$ that minimizes $\mathit{effort}(\mathcal{P}_{\bowtie\theta}[\varphi], \alpha, \beta, x, x)$ in the interval $(0, \delta(\theta)/(0.5 + \delta(\theta)))$. A lower effort could, conceivably, be achieved with an asymmetric observation error, but it would require optimization in two dimensions to find the asymmetric observation error with minimal estimated effort.

Example 5.3. Consider the UTSL formula $\Phi = \mathcal{P}_{\geq 0.9}\big[X\ \mathcal{P}_{\geq 0.85}[X\ x{=}1]\big]$, and assume that we use (5.1) with $\delta_0 = 0.05$ to determine the width of indifference regions. This gives us the probability thresholds $p_0 = 0.91$ and $p_1 = 0.89$ for the outer probabilistic operator, and $p'_0 = 0.865$ and $p'_1 = 0.835$ for the inner probabilistic operator. Furthermore, assume that we want to verify Φ with error bounds $\alpha = \beta = 0.01$. Using Definition 5.3 and assuming symmetric observation error, we estimate the effort of verifying Φ as the product of $n(p_0 \cdot (1 - \alpha'), 1 - (1 - p_1)(1 - \alpha'), \alpha, \beta)$ and $n(p'_0, p'_1, \alpha', \alpha')$. Figure 5.3 plots the two factors of the total estimated effort separately for a single sampling plan. This choice of sampling plan means that the estimated effort is equal to the actual effort. The dotted line indicates the upper bound on the symmetric observation error: $0.02/1.02 \approx 0.0196$. The total effort is plotted in Figure 5.4. The effort is minimal at $\alpha' = \beta' \approx 0.00153$, which therefore is the optimal symmetric observation error for a single sampling plan.

5.2.4 Memoization

Statistical verification of UTSL formulae with nested probabilistic operators can be rather costly because each observation for the outermost probabilistic operator involves at least one acceptance sampling test.

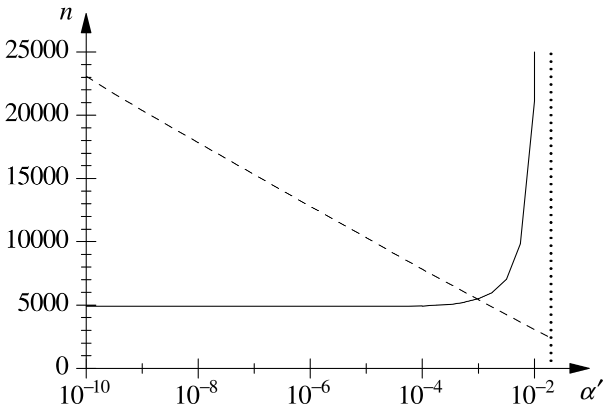


Figure 5.3: Heuristic estimate, as a function of the symmetric observation error α', of the effort needed for the verification of the inner (dashed curve) and outer (solid curve) probabilistic operators of the UTSL formula $\mathcal{P}_{\geq 0.9}\big[X\ \mathcal{P}_{\geq 0.85}[X\ x{=}1]\big]$.

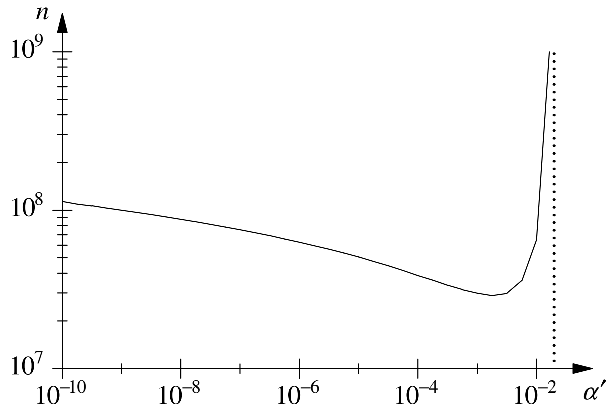


Figure 5.4: Total heuristic estimated effort, as a function of the symmetric observation error α', for the UTSL formula $\mathcal{P}_{\geq 0.9}\big[X\ \mathcal{P}_{\geq 0.85}[X\ x{=}1]\big]$.

When the path formula is $\Phi\ \mathcal{U}^I\ \Psi$ with Φ or Ψ being probabilistic statements, then each observation may require acceptance sampling to be performed for every state visited along a trajectory before time $\sup I$. We can improve performance radically through the use of *memoization* (Michie 1968). This means that each component of a path formula is verified only once in a specific state.

Memoization does not affect the validity of the verification result, since a time-bounded until formula can be treated as a large conjunction, and we have noted that Theorem 5.4 does not require conjuncts to be verified independently. Thus, we can ensure error bounds α' and β' for each observation even if we reuse verification results along a sample trajectory. It is also safe to reuse memoized results across observations. If we ensure that each trajectory is an independent sample, each observation will be independent as well. This means that each nested probabilistic statement needs to be verified only once per unique visited state.

5.3 Distributed Acceptance Sampling

Statistical solution methods that use samples of independent observations are trivially parallelizable. We can use multiple computers to generate the observations, as noted already by Metropolis and Ulam (1949, p. 340), and expect a speedup linear in the added computing power. We must, of course, ensure that the observations generated by different machines are indeed independent, and this requires extra care when

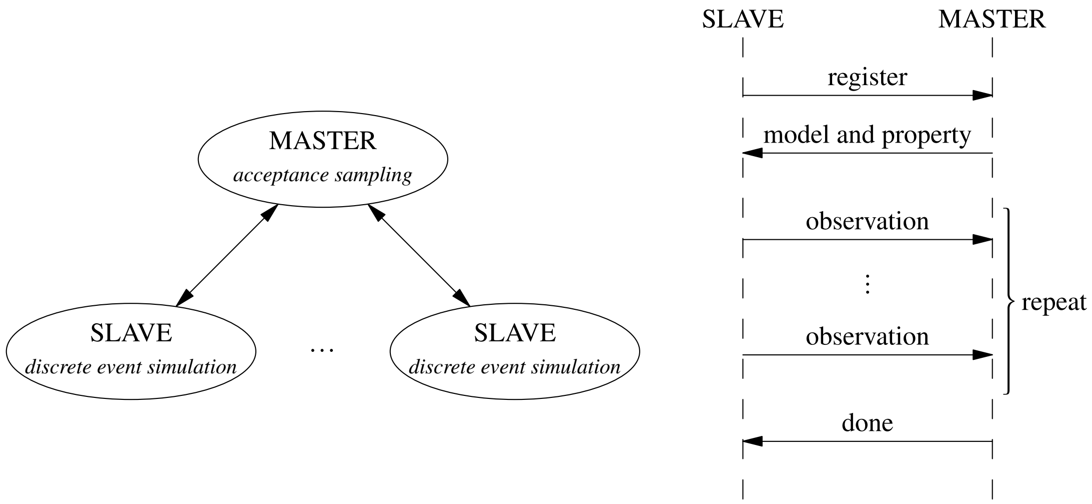


Figure 5.5: Master/slave architecture and communication protocol for distributed acceptance sampling.

initializing the pseudorandom number generators on each machine. It may not be sufficient simply to use a different seed on each machine, because the seed determines only the start of a sequence of numbers and does not alter the way in which these numbers are generated. Independence can be assured, for example, if we use the scheme proposed by Matsumoto and Nishimura (2000), which encodes a process identifier into the pseudorandom number generator. This effectively creates a new pseudorandom number generator for each unique identifier rather than a different segment of a sequence from the same generator, as is the case when only the seed is varied.

It is natural to adopt a master/slave architecture (Figure 5.5) for the distributed verification task. One or more slave processes register their ability to generate observations with a single master process. The master process collects observations from the slave processes and performs an acceptance sampling procedure. Independent observations can be generated by separate slave processes, running on different nodes of a computer network or multiprocessor machine, without the need for communication between the slave processes. Each slave process is assigned a unique identifier by the master process to ensure that the slave processes use different pseudorandom number generators. After the initial communication to register the slave process with the master process and inform the slave process of its identifier and the model it should use, the only communication required is a single bit from a slave process to the master process for each observation that is generated. The right side of Figure 5.5 illustrates a typical communication session between slave and master processes.

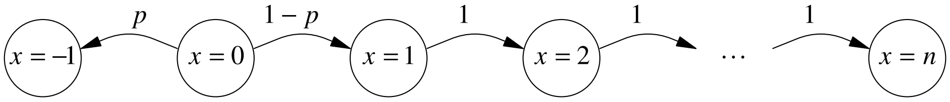


Figure 5.6: Discrete-time Markov process used to illustrate risk of bias in distributed sampling.

5.3.1 Unbiased Distributed Sampling

When using distributed sampling with a sequential test, such as Wald's sequential probability ratio test, it is important not to introduce a bias against observations that take a longer time to generate. For UTSL model checking, each observation involves the generation of a trajectory prefix through discrete event simulation and the verification of a path formula over the generated trajectory prefix. If we were simply to use observations as they became available, we could easily end up violating the probabilistic guarantees of the acceptance sampling test as specified by the parameters α, β, and δ. This is illustrated by the following example.

Example 5.4. Consider the discrete-time Markov process shown in Figure 5.6 and assume that we want to verify the UTSL property $\mathcal{P}_{\geq 0.9}[x{<}n \;\mathcal{U}\; x{<}0]$ in the state satisfying $x{=}0$. Note that sample trajectories starting in the state with $x{=}0$ and satisfying the path formula $x{<}n \,\mathcal{U}\, x{<}0$ involve a single transition, while sample trajectories not satisfying the path formula involve n transitions. Thus, while the property actually holds with probability p, the effort required to produce a negative observation is roughly n times as high as to produce a positive observation. If we use m slave processes to generate observations, and ignore communication overhead, we can expect to see $\sum_{i=1}^{n-1} mp^i = mp(1-p^{n-1})/(1-p)$ positive observations before seeing a negative observation. If, instead, we generate the observations with a single process, the expected number of positive observations before the first negative observation is $\sum_{i=1}^{\infty} ip^i(1-p) = p/(1-p)$. These numbers differ by a factor of $m(1-p^{n-1})$. Figure 5.7 shows how this can introduce bias in the analysis, leading to an acceptance sampling test with a probability of accepting the hypothesis $H_0 : p \geq p_0$ that varies significantly with m.

This bias is avoided by committing, *a priori*, to the order in which observations will be taken into account. This can be accomplished, for example, by processing observations in cyclic order. Thus, if slave process 0 produces two observations before slave process 1 produces a single observation, the master process waits for an observation from slave process 1 before processing the second observation from slave process

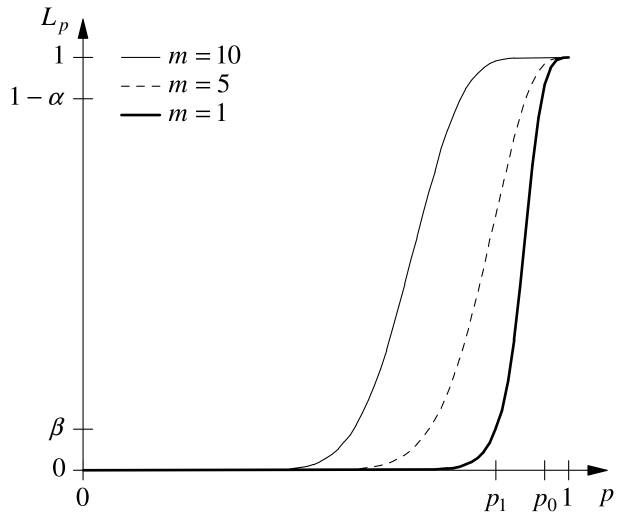


Figure 5.7: Probability of accepting $\mathcal{P}_{\geq 0.9}[x{<}n\ \mathcal{U}\ x{<}0]$ for distributed acceptance sampling with m machines and using observations immediately as they arrive.

0. Observations that are received out-of-order are buffered until it is time to process them.

The cyclic scheme works well if the slave processes are executed on homogeneous nodes. In a heterogeneous environment, however, a pure cyclic scheme will not take full advantage of the available computational resources. In the same amount of time, a slave process running on a fast machine will generate, on average, more observations than a process running on a slow machine. The cyclic scheme, however, will use the same number of observations from both slave processes. As a result, a potentially large fraction of the observations generated on the faster machine will go to waste and the speedup will therefore not be as large as one would expect from the added computing power.

To address this problem, we can maintain a *dynamic* schedule, instead of a *static* schedule, of the order in which observations are processed. At the beginning, we schedule to receive one observation from each slave process in a specific order. When an observation arrives from slave process i, we insert i at the end of the current schedule, leaving two entries for i in the schedule. We then check if i is at the front of the schedule, in which case we immediately process the observation and pop i from the front. Otherwise, we buffer the observation for later use. At the removal of an item from the front of the schedule, we check to see if there is a buffered observation for the new front item. We keep processing buffered observations, removing the front item of the schedule for each processed observation, until the front item has no buffered observations.

By rescheduling the processing of the next observation for a slave process at the arrival of an observation, we get a schedule that automatically adjusts to variations in performance of slave processes. If we have two slave processes, with process 0 running on a machine that is twice as fast as the machine that process 1 is running on, then the adaptive schedule will lead to us processing, on average, twice as many observations from process 0 as from process 1. This happens automatically, without the need for explicit communication of performance characteristics of the nodes on which slave processes are running.

5.3.2 Out-of-Order Observations

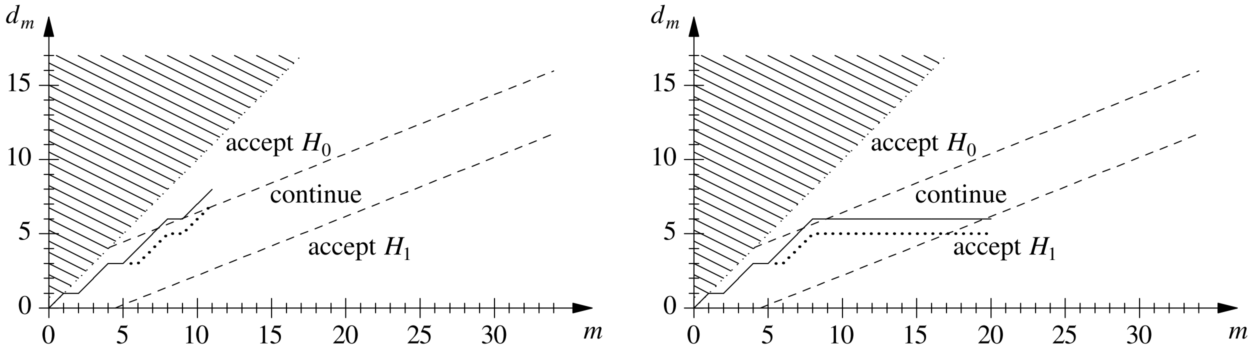
With the adaptive ordering of observations, we are guaranteed linear speedup, at least in the limit. We can potentially do even better by processing out-of-order observations as they arrive, although of course not in the naive way that has already been shown to introduce bias against long sample trajectories.

Recall from Section 2.2.3 that the first m observations $x_1, \ldots, x_m$ can be summarized with the statistic $d_m = \sum_{i=1}^{m} x_i$, and that a sequential acceptance sampling test can be carried out by comparing d_m at each stage to an acceptance number a_m and a rejection number r_m. Assume that we have processed m in-order observations when observation x_l arrives. We proceed as usual if $l = m + 1$, but we want to take the observation into account immediately even if $l > m + 1$ instead of waiting until after we have received observations x_{m+1} through x_{l-1}. This can be done, without altering the probability of accepting H_0 for the acceptance sampling test, by computing lower and upper bounds for d_{m+1} through d_l. We define the following quantities:

$$\check{x}_i = \begin{cases} x_i & \text{if } x_i \text{ has been received} \\ 0 & \text{otherwise} \end{cases} \qquad \hat{x}_i = \begin{cases} x_i & \text{if } x_i \text{ has been received} \\ 1 & \text{otherwise} \end{cases}$$

The lower bound for d_i is $\check{d}_i = \sum_{j=1}^{i} \check{x}_j$ and the upper bound is $\hat{d}_i = \sum_{j=1}^{i} \hat{x}_j$. We can accept H_0 at stage l if $\check{d}_l \geq a_l$ and $\check{d}_i > r_i$ for all $i < l$. The second condition prevents us from accepting H_0 at a stage if it is still possible that H_1 could be accepted at an earlier stage. If we were to ignore this condition, then we could end up with a biased acceptance sampling test again. The conditions for acceptance of H_1 at stage l is $\hat{d}_l \leq r_l$ and $\hat{d}_i < a_i$ for all $i < l$.

Figure 5.8(a) shows an example of sequential acceptance sampling with out-of-order observations. In this case, observations 7 through 11 arrive before observation 6, but it is safe to accept H_0 without waiting



(a) Accept H_0.

(b) Inconclusive.

Figure 5.8: Acceptance sampling with out-of-order observations. The solid curve in each of the plots represents $\hat{d}_m$ and the dotted curve represents $\tilde{d}_m$. Note that both curves cross the acceptance line for H_1 in (b), but that the curve for $\hat{d}_m$ crosses the acceptance line for H_0 at an earlier stage.

for observation 6. The speedup can be significant if observation 6 happens to take an exceptionally long time to generate. In Figure 5.8(b), we have an example of a situation where we have to wait for observation 6, because the final outcome of the test depends on it: if $x_6 = 1$ we will accept H_0, but if $x_6 = 0$ we will accept H_1.

5.4 Complexity of Statistical Probabilistic Model Checking

The time complexity of statistical probabilistic model checking depends on the number of observations (sample size) required to reach a decision, as well as the time required to generate each observation. An observation involves the verification of a path formula φ over a sample trajectory σ_i. The sample size for a sequential acceptance sampling test is a random variable, and so is the time per observation, which means that we can generally only talk about the *expected* complexity of statistical probabilistic model checking.

First, consider the time complexity for UTSL formulae without nested probabilistic operators. The first component of the complexity is the time per observation. A sample trajectory σ_i may very well be infinite, but in order to verify the path formula $X^I\ \Phi$, we only need to consider a finite prefix of σ_i. The same is true for path formulae of the form $\Phi\ \mathcal{U}^I\ \Psi$ if the conditions of Theorem 5.1 are satisfied. Without nested probabilistic operators, nested UTSL formulae will be classical logic expressions, which we assume can be verified in constant time. Let m be the expected effort to simulate a state transition. The time per observation is proportional to m for $X^I\ \Phi$ and proportional to m times the number of state transitions that occur in a time interval of length $\sup I$ for $\Phi\ \mathcal{U}^I\ \Psi$. Let q denote the expected number of state transitions that occur in a unit length interval of time. For continuous-time Markov processes, an upper bound for q is the maximum exit rate of any state. The expected time per observation is then $O(m \cdot q \cdot \sup I)$ for $\Phi\ \mathcal{U}^I\ \Psi$. This is a worst-case estimate, because it assumes that $\neg\Phi \vee \Psi$ is not satisfied prior to time $\sup I$. If we reach a state satisfying $\neg\Phi \vee \Psi$ long before visiting $q \cdot \sup I$ states, then we can determine the truth value of $\Phi\ \mathcal{U}^I\ \Psi$ without considering further states.

The second component of the time complexity for verifying $\mathcal{P}_{\bowtie\theta}[\varphi]$ is the expected sample size, which is a function of the error bounds α and β, and the two probability thresholds p_0 and p_1 (alternatively expressed using the threshold θ and the half-width of the indifference region δ). If we use a sequential test, then the expected sample size also depends on the unknown probability measure p of the set of trajectories that satisfy φ. The expected sample size for various acceptance sampling tests was discussed in Section 2.2. For example, we showed that the sample size for a single sampling plan is approximately proportional to the logarithm of α and β, and inversely proportional to the width of the indifference region.

Let N_p denote the expected sample size of the acceptance sampling test we use to verify probabilistic statements. The verification time for $\mathcal{P}_{\bowtie\theta}\big[X^I\ \Phi\big]$ is then $O(N_p \cdot m)$ and for $\mathcal{P}_{\bowtie\theta}\big[\Phi\ \mathcal{U}^I\ \Psi\big]$ it is $O(N_p \cdot m \cdot q \cdot \sup I)$. Note that there is no direct dependence on the size of the state space of the model, which is in sharp contrast to numerical solution techniques for probabilistic model checking, whose time complexity is proportional to the size of the state space (Hansson and Jonsson 1994; Baier et al. 2003).

The time complexity of statistical probabilistic model checking is independent of the size of the state space for a model if N_p, m, and q are independent of state space size as well. We can make N_p completely model independent by using a single sampling plan, in which case N_p depends only on the parameters α, β, θ, and δ. The factor m is generally both model and implementation dependent and therefore hard to capture. For generalized semi-Markov processes, for example, m could very well be proportional to the number of events in the model. It can also be state space dependent, but models often have structure that can be exploited by the simulator to avoid such dependence. Finally, q is clearly model dependent, but may be independent of the size of the state space. For example, this is the case for the symmetric polling system described in Section 6.1.2.

With nested probabilistic operators, the verification time per state along a sample trajectory is no longer constant. The complexity depends on the level of nesting and the path operators involved. Here, we consider the UTSL formula $\mathcal{P}_{\bowtie\theta}\big[\mathcal{P}_{\bowtie\theta'}\big[\Phi' \ \mathcal{U}^{I'} \ \Psi'\big] \ \mathcal{U}^{I} \ \Psi\big]$ with one level of nesting as an example. In the worst case we need to verify $\mathcal{P}_{\bowtie\theta'}\big[\Phi' \ \mathcal{U}^{I'} \ \Psi'\big]$ in $q \cdot \sup I$ states for each of the N_p observations required for the verification of the outer probabilistic operator. The worst-case complexity for verifying $\mathcal{P}_{\bowtie\theta'}\big[\Phi' \ \mathcal{U}^{I'} \ \Psi'\big]$, assuming Φ' and Ψ' do not contain any probabilistic operators, is $O(N'_p \cdot m \cdot q \cdot \sup I')$, so the total expected worst-case complexity is $O(N_p \cdot N'_p \cdot m^2 \cdot q^2 \cdot \sup I \cdot \sup I')$. However, if we use memoization, the expected worst-case complexity is $O(m \cdot q \cdot (N_p \cdot \sup I + k \cdot N'_p \cdot \sup I'))$ instead, where k is the expected number of unique states visited within $\sup I + \sup I'$ time units from some initial state. The value of k is in the worst case $|S|$, the size of the state space, but can be significantly smaller depending on the dynamics of the model and the time bounds $\sup I$ and $\sup I'$.

The space complexity of statistical probabilistic model checking is generally quite modest. We need to store the current state of a sample trajectory when generating an observation for the verification of a probabilistic UTSL formula, and this typically requires $O(\log |S|)$ space, where $|S|$ is the number of states for the model. For stochastic discrete event systems that do not satisfy the Markov property, we may also need to store additional information, such as scheduled trigger times for enabled events in the case of generalized semi-Markov processes. In the presence of nesting, we may need to store up to d states simultaneously at any point in time during verification, where d is the maximum depth of a nested probabilistic operator. The nesting depth for a UTSL formula Φ is at most $|\Phi|$, so the space requirements are still modest. If we use memoization to speed up the verification of UTSL formulae with nested probabilistic operators, the space complexity can be as high as $O(|\Phi| \cdot |S|)$. Memoization, as usual, is a way of trading space efficiency for time efficiency.

The statistical approach works for infinite-state systems as well, so long as we need to visit only a finite number of states in order to verify a UTSL formula. This is the case if the conditions of Theorem 5.1 are satisfied. To verify $\mathcal{P}_{\bowtie\theta}[\Phi \ \mathcal{U}^{I} \ \Psi]$, the expected number of states that we need to visit is $O(N_p \cdot q \cdot \sup I)$. The expected number of unique states is $O(\min(N_p \cdot q \cdot \sup I, |S|))$, which becomes the expected space complexity for memoization with one level of nesting.

Chapter 6

Empirical Evaluation of Probabilistic Model Checking

In the previous chapter, we described a statistical approach to probabilistic model checking, and concluded with a theoretical discussion regarding the computational complexity of our statistical solution method. To get a better feeling for how well our solution method performs in practice, we evaluate it empirically on a set of case studies taken from the literature on performance evaluation and probabilistic model checking. We also compare the statistical solution method with the leading numerical solution method for transient analysis of Markov processes. The purpose of this empirical study is to show how the performance of the different solution methods depends on input parameters and model characteristics.

Our empirical results indicate that the statistical solution method scales better than the numerical solution method as the size of the state space increases, but that the performance of the two methods scales similarly as a function of the time bounds involved in the UTSL formulae. We also show that the sequential probability ratio test generally outperforms the sequential modification of a single sampling plan, although there are exceptions to this rule, as was noted already in Section 2.2.3.

The empirical evaluation that we present in this chapter is meant as an aid to practitioners who want to use probabilistic model checking to verify their system designs. We cannot recommend a single solution method that is superior in all cases, as the right choice depends on characteristics of the model and the requirements on the accuracy of the model checking result. We show the tradeoffs between accuracy and

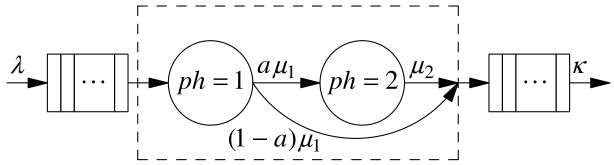


Figure 6.1: Tandem queuing network with a two-phase Coxian distribution governing the routing time between the queues.

speed that exist, and the results we present can help a user make an informed choice regarding solution method and input parameters.

The empirical results presented in this chapter were generated on a 3 GHz Pentium 4 PC running Linux, and with an 800 MB memory limit set per process, unless noted otherwise. The memory limit per process was set lower than the physical memory limit of the machine (1 GB) to avoid swapping.

6.1 Case Studies

We present two case studies, taken from the literature on performance evaluation and probabilistic model checking, and selected to accentuate specific performance characteristics of solution methods for probabilistic model checking. A third simple example is also introduced to illustrate the use of nested probabilistic operators in UTSL.

6.1.1 Tandem Queuing Network

The first case study is based on a model of a tandem queuing network presented by Hermanns et al. (1999). The network consists of two serially connected queues, each with capacity n, making the total capacity of the system $2n$. Figure 6.1 shows a schematic view of the model. Messages arrive at the first queue, get routed to the second queue after having been in the first queue for some time, and eventually leave the system after being processed in the second queue. The interarrival time for messages at the first queue is exponentially distributed with rate $\lambda = 4n$. The processing time at the second queue is exponentially distributed with rate $\kappa = 4$. The routing time distribution is a two-phase Coxian distribution with parameters $\mu_1 = \mu_2 = 2$ and $a = 0.9$. The size of the state space for a tandem queuing network of capacity $2n$ is $O(n^2)$.

We will verify whether the probability is less than 0.5 that a system starting out with both queues empty becomes full within τ time units. Let $s_i \in \{0, \dots, n\}$, for $i \in \{1, 2\}$, be the number of messages currently in the ith queue. The tandem queuing network is full if the formula $s_1{=}n \wedge s_2{=}n$ holds. The UTSL formula $\mathcal{P}_{<\,0.5}[\Diamond^{[0,\tau]}\ s_1{=}n \wedge s_2{=}n]$ represents the property of interest, and we will verify this formula in the state $s_1 = 0 \wedge s_2 = 0$.

6.1.2 Symmetric Polling System

The second case study uses the model of an n-station symmetric polling system described by Ibe and Trivedi (1990). Each station has a single-message buffer and the stations are attended by a single server in cyclic order. The server begins by polling station 1. If there is a message in the buffer of station 1, the server starts serving that station. Once station i has been served, or if there is no message in the buffer of station i when it is polled, the server starts polling station $i + 1$ (or 1 if $i = n$). The polling and service times are exponentially distributed with rates $\gamma = 200$ and $\mu = 1$, respectively. Messages arrive to the system, as a whole, according to a Poisson process, and the inter-arrival time is exponentially distributed with rate 1. At arrival, messages are assigned, with equal probability, to one of the n stations. If a message is assigned to a station whose buffer is full, then the message is dropped. Another way to think of this is that there is a separate arrival event for each station, with the inter-arrival time per station being exponentially distributed with rate $\lambda = 1/n$. The fact that arrival rates are equal for all stations makes the system symmetric. The size of the state space for a system with n stations is $O(n{\cdot}2^n)$.

We will verify the property that, if station 1 is full, then it is polled within τ time units with probability at least θ. We do so for different values of n, τ, and θ in the state where station 1 has just been polled and the buffers of all stations are full. Let $s \in \{1, \dots, n\}$ be the station currently receiving the server's attention, let $a \in \{0, 1\}$ represent the activity of the server (0 for polling and 1 for serving), and let $m_i \in \{0, 1\}$ be the number of messages in the buffer of station i. The property of interest is represented in UTSL as $m_1{=}1 \rightarrow \mathcal{P}_{\geq\,\theta}\big[\Diamond^{[0,\tau]}\ \mathit{poll}_1\big]$, where $\mathit{poll}_1 \equiv s{=}1 \wedge a{=}0$, and the state in which we verify the formula is given by $s{=}1 \wedge a{=}1 \wedge m_1{=}1 \wedge \cdots \wedge m_n{=}1$.

6.1.3 Robot Grid World

The third case study involves a robot navigating in a grid world, and was introduced by Younes et al. (2004) to illustrate the verification of formulae with nested probabilistic operators. We have an $n \times n$ grid world

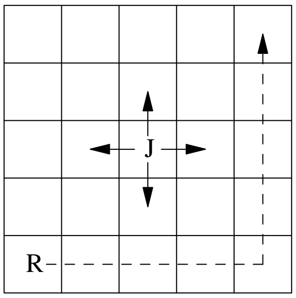


Figure 6.2: A grid world with a robot (R) in the bottom left corner and a janitor (J) in the center. The dashed arrow indicates the path of the robot. The janitor moves with equal probability to any of the adjacent squares.

with a robot moving from the bottom left corner to the top right corner. The robot first moves along the bottom edge and then along the right edge. In addition to the robot, there is a janitor moving randomly around the grid. Figure 6.2 provides a schematic view of a grid world with $n = 5$.

The robot moves at rate $\lambda_R = 1$, unless the janitor occupies the destination square, in which case the robot remains stationary. The janitor moves around randomly in the grid world at rate $\lambda_J = 2$, selecting the destination from the set of neighboring squares according to a discrete uniform distribution. The robot initiates communication with a base station at rate $\mu = 1/10$, and the duration of each communication session is exponentially distributed with rate $\kappa = 1/2$.

The objective is for the robot to reach the goal square at the top right corner within τ_1 time units with probability at least 0.9, while maintaining at least a 0.5 probability of periodically communicating with the base station. Let c be a Boolean state variable that is true when the robot is communicating with the base station, and let x and y be two integer valued state variables holding the current location of the robot. The UTSL formula $\mathcal{P}_{\geq 0.9}\big[\mathcal{P}_{\geq 0.5}\big[\Diamond^{[0,\tau_2]}\ c\big]\ \mathcal{U}^{[0,\tau_1]}\ x{=}n \wedge y{=}n\big]$ expresses the desired objective. The robot moves along a line only, so the size of the state space for the robot grid world is $O(n^3)$.

6.2 Evaluation of Statistical Solution Method

As discussed in Section 5.4, there are two main factors influencing the verification time for the statistical approach: the sample size required to achieve prescribed accuracy and the length of trajectory prefixes (in terms of state transitions) required to determine if a path formula holds.

The sample size depends on the sampling plan that we choose to use, the error bounds α and β that we want to guarantee, the threshold θ, as well as our choice of $\delta(\theta)$ determining the half-width of an indifference region centered around θ. For sequential sampling plans, the sample size is a random variable whose expectation also varies with p, which in our case is the probability measure of a set of trajectories satisfying a path formula. The approximation formulae for the expected sample size of various sampling plans provided in Section 2.2 give us some idea of what to expect, and the empirical results presented in this section show the actual performance on the various case studies.

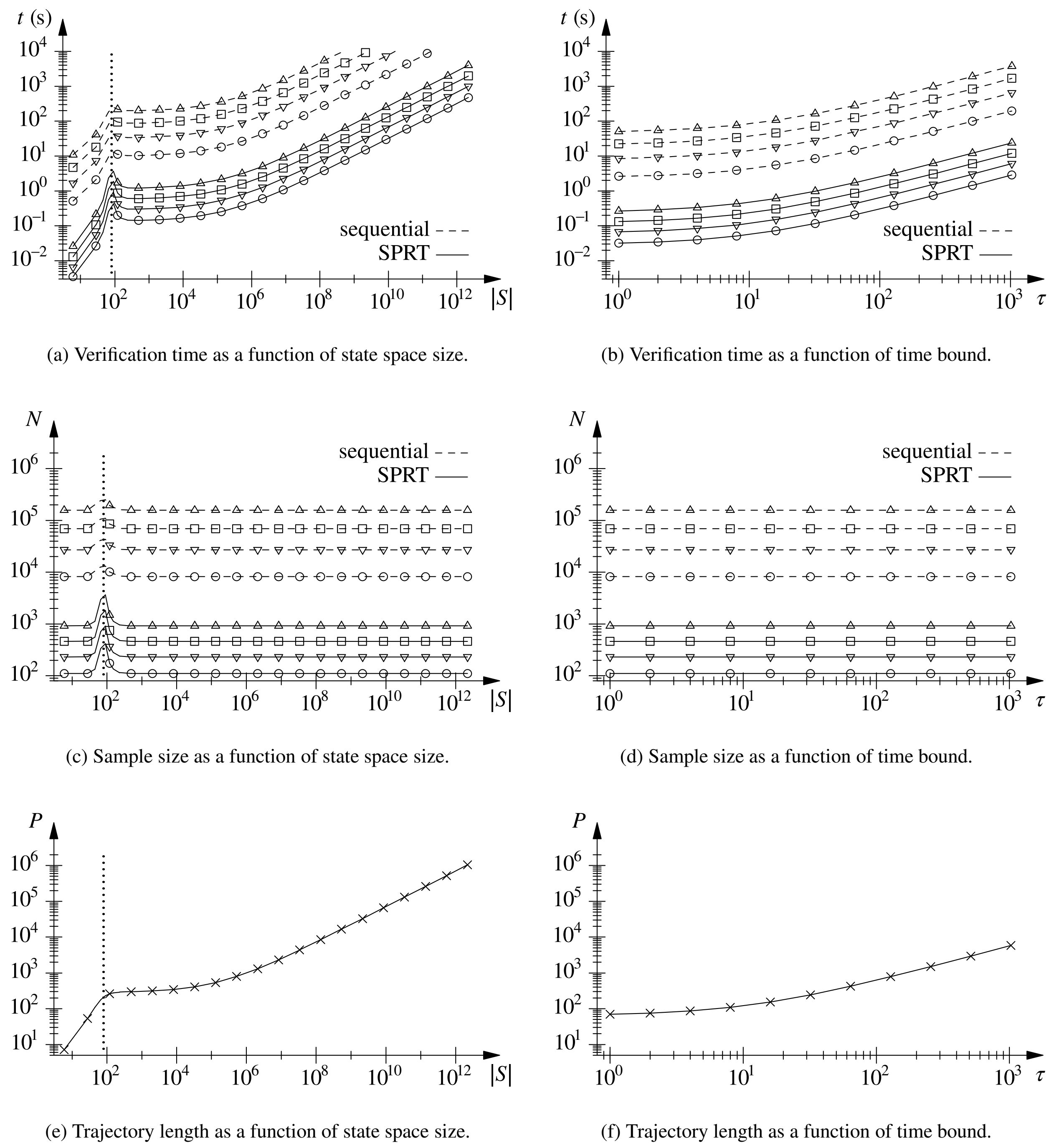
The expected length of trajectories varies with the model and the path formula, as we will see. If we are lucky, we can verify a time-bounded path formula over a sample trajectory by considering only a short prefix that ends long before the time bound is exceeded. For some models, however, the number of state transitions that occur in a given time interval may be large, even if the interval is short, and this will lead to longer verification times.

6.2.1 Comparing Sampling Plans

We consider two different sampling plans introduced in Section 2.2: the sequential version of a single sampling plan (Algorithm 2.2) and Wald's sequential probability ratio test (SPRT; Algorithm 2.3). We do not include experiments with a non-sequential single sampling plan. There is of course a slight overhead introduced by using a sequential stopping rule with a single sampling plan, but this overhead is negligible (essentially three additional integer operations per iteration). The reduction in expected sample size that we get from using a sequential stopping rule dominates the small overhead required to test for early termination.

Figures 6.3 and 6.4 present data for the tandem queuing network and symmetric polling system case studies, respectively. In each case, we show verification time for the simple sequential sampling plan and the SPRT using four different test strengths (subfigures (a) and (b)). We also give details of both sample size (subfigures (c) and (d)) and trajectory length (subfigures (e) and (f)). For all data, we plot the results both against model size (subfigures (a), (c), and (e)) and against the time bound of the path formula (subfigures (b), (d), and (f)). Each data point is an average over 20 runs. We used $\delta(\theta) = 5 \cdot 10^{-3}$ as the half-width of the indifference region. Furthermore, we used a symmetric test strength ($\alpha = \beta$) across the board.

Our data shows that the SPRT outperforms the simple sequential test almost exclusively by a wide margin. We can typically solve the same model checking problem with the SPRT using test strength 10^{-8} in shorter time than it takes to solve the same problem with a simple sequential test using test strength 10^{-1}.



(a) Verification time as a function of state space size.

(b) Verification time as a function of time bound.

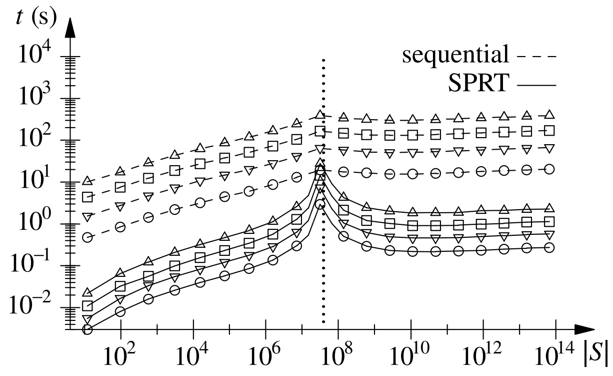
(c) Sample size as a function of state space size.

(d) Sample size as a function of time bound.

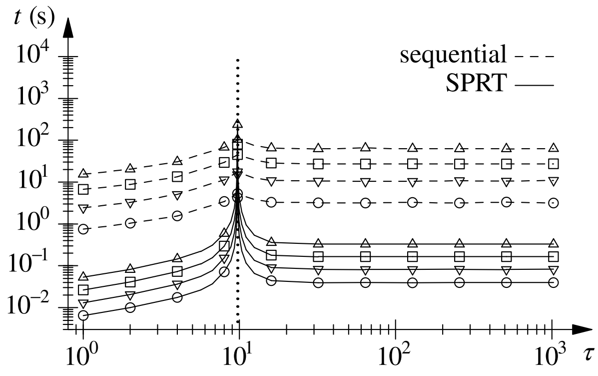
(e) Trajectory length as a function of state space size.

(f) Trajectory length as a function of time bound.

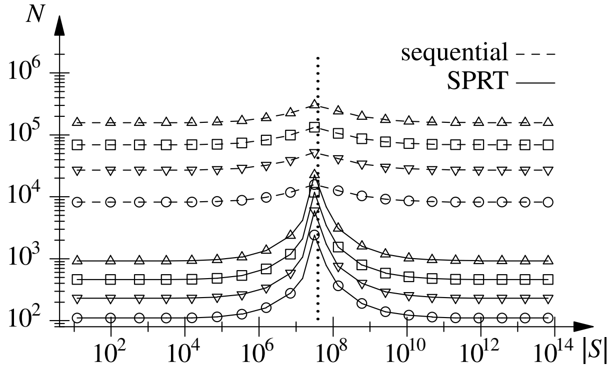
Figure 6.3: Empirical results for the tandem queuing network ($\theta = 0.5$), with $\tau = 50$ (left) and $n = 63$ (right), using acceptance sampling with $2\delta = 10^{-2}$ and symmetric error bounds $\alpha = \beta$ equal to 10^{-8} ($\triangle$), 10^{-4} ($\square$), 10^{-2} ($\triangledown$), and 10^{-1} ($\circ$). The average trajectory length is the same for all values of α and β. The dotted lines mark a change in the truth value of the formula being verified.



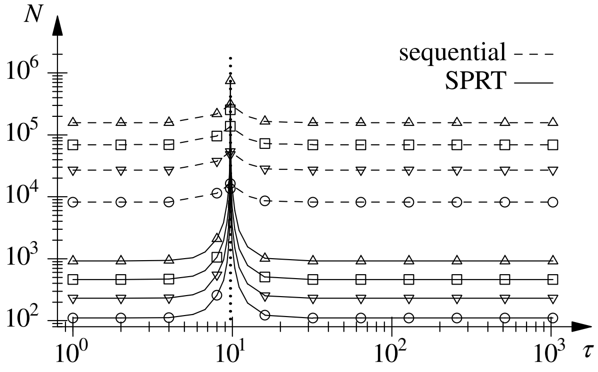
(a) Verification time as a function of state space size.



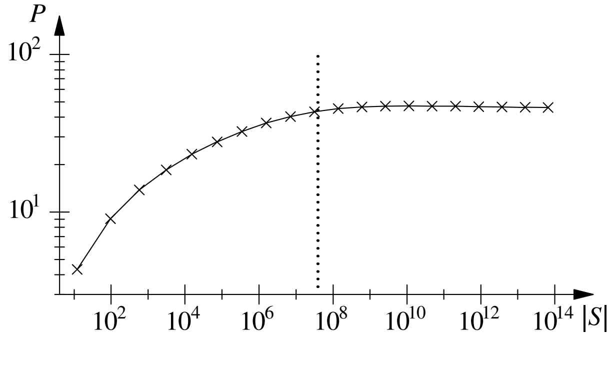
(b) Verification time as a function of time bound.



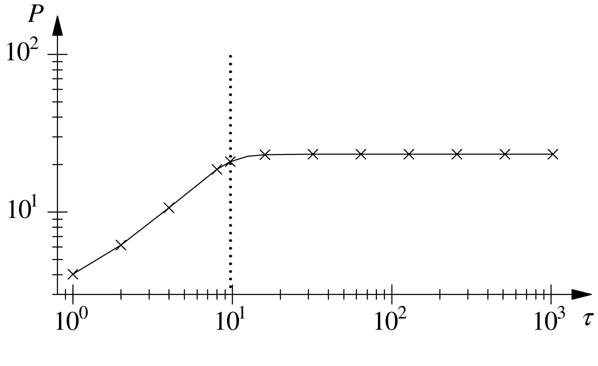
(c) Sample size as a function of state space size.



(d) Sample size as a function of time bound.



(e) Trajectory length as a function of state space size.



(f) Trajectory length as a function of time bound.

Figure 6.4: Empirical results for the symmetric polling system ($\theta = 0.5$), with $\tau = 20$ (left) and $n = 10$ (right), using acceptance sampling with $2\delta = 10^{-2}$ and symmetric error bounds $\alpha = \beta$ equal to 10^{-8} ($\triangle$), 10^{-4} ($\square$), 10^{-2} ($\triangledown$), and 10^{-1} ($\circ$). The average trajectory length is the same for all values of α and β. The dotted lines mark a change in the truth value of the formula being verified.

The difference in performance is due entirely to a difference in sample size, as the average trajectory length is the same for both tests regardless of strength. The average sample size for the SPRT roughly doubles when the test strength goes from 10^{-x} to 10^{-2x}, while the average sample size for the simple sequential test more than doubles (and often more than triples) for the same change in test strength.

The vertical dotted lines in the figures indicate a change in the truth value of the UTSL formula that is being verified. This line marks the value of $|S|$ or τ where the probability measure for the set of trajectories satisfying the path formula is exactly equal to the probability threshold θ. We can see that the average sample size for both tests peaks in the vicinity of the dotted line, with the peaks for the SPRT being more pronounced than those for the simple sequential test.

The average trajectory length for the tandem queuing network increases linearly with the capacity n of the queues. This is because the arrival rate for messages is $4n$, so the average number of state transitions that occur in a fixed interval of time increases with n. Note, however, that the size of the state space is $O(n^2)$ for the tandem queuing network, so the the average trajectory length is proportional to the square root of $|S|$ (Figure 6.3(e)). Thus, the average trajectory length, and therefore also the overall time complexity for the statistical solution method, is sublinear in the size of the state space. In contrast, the rates for the symmetric polling system are independent of the size of the state space. Initially, the average trajectory length increases with the size of the state space (Figure 6.4(e)) because it takes longer time to achieve $poll_1$ with more polling stations. As the state space increases further, the probability of achieving $poll_1$ in the interval $[0, \tau]$ goes to zero, and all sample trajectories end with the time bound τ being exceeded. The expected number of state transitions occurring in the interval $[0, \tau]$ is the same for all state space sizes, since the exit rates are constant, so the verification time does not increase for larger state spaces.

As a function of the time bound τ (Figure 6.3(f)), for a fixed n, the average trajectory length grows linearly with τ for the tandem queuing network, at least for sufficiently large values of τ. The same is true for the symmetric polling system (Figure 6.4(f)) for small values of τ, but as τ increases so does the probability of achieving $poll_1$ in the interval $[0, \tau]$ (Figure 6.5), and the average trajectory length approaches a constant value as τ increases. This shows how the performance of the statistical solution method depends on the formula that is verified in a more complex way than simply through the time bounds of path formulae.

While the SPRT typically has a smaller expected sample size than the simple sequential test for the same test strength, a clear exception is seen in Figure 6.4(d). We witness the same phenomenon in Figure 6.6 for

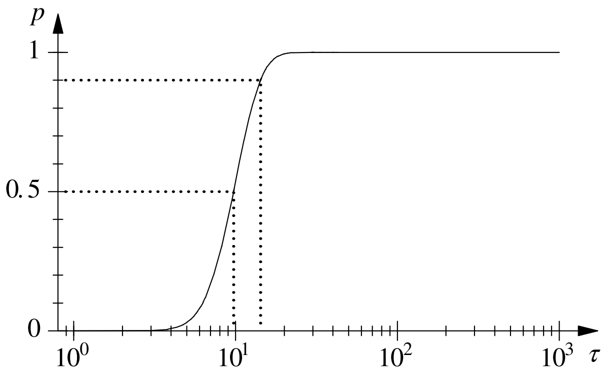


Figure 6.5: Probability p of the set of trajectories satisfying the path formula $\Diamond^{[0,\tau]}$ $poll_1$ for the symmetric polling system.

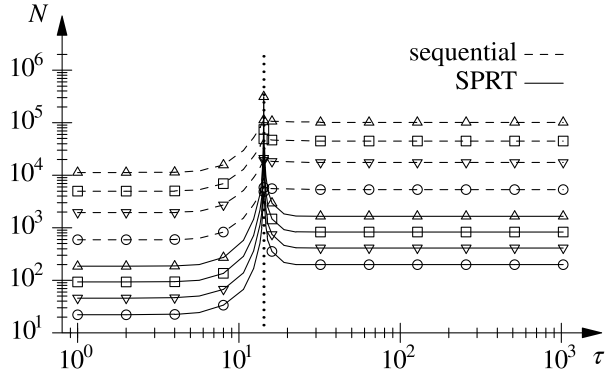
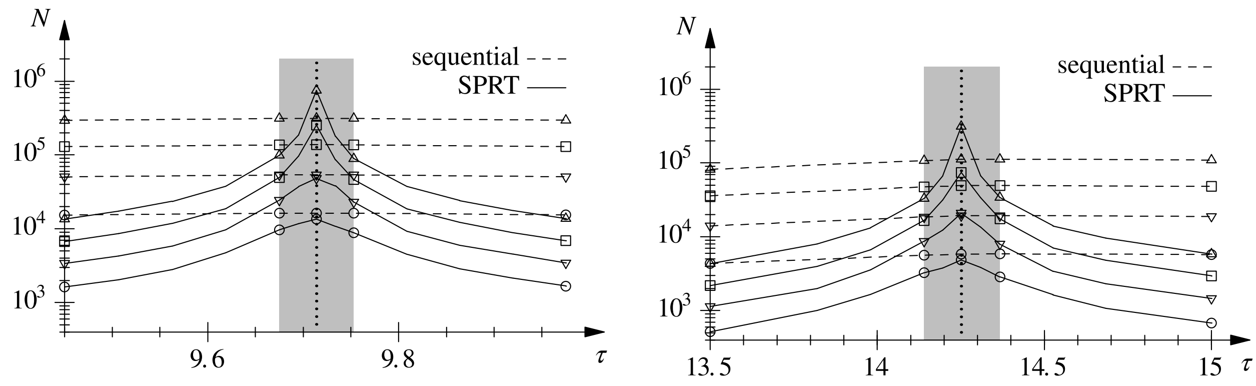


Figure 6.6: Sample size as a function of the formula time bound for the symmetric polling system ($\theta = 0.9$ and $n = 10$), using acceptance sampling with $2\delta = 10^{-2}$ and symmetric error bounds $\alpha = \beta$ equal to 10^{-8} (△), 10^{-4} (□), 10^{-2} (▽), and 10^{-1} (○).

a different θ (0.9 instead of 0.5), which also shows the variation in performance based on the threshold. For $\theta = 0.5$, the sample size is the same on both sides of the vertical dotted line, but it is notably lower to the left of the line for $\theta = 0.9$. There is a sharp peak in the expected sample size for the SPRT close to where the truth value of the UTSL formula changes, as indicated by the dotted line. For $\alpha = \beta$ equal to 10^{-4} and 10^{-8}, the SPRT has a larger expected sample size than the simple sequential test. We can see this more clearly in Figure 6.7, where we have zoomed in on the relevant region. The gray area indicates the range of τ for which the probability measure, p, of the set of trajectories satisfying the path formula $\Diamond^{[0,\tau]}$ $poll_1$ is in the indifference region $(\theta - \delta, \theta + \delta)$. We can see that there is a sharp increase in the expected sample size for the SPRT in and near the indifference region, while the expected sample size for the simple sequential test remains largely unchanged. Still, it is only for a very narrow range of τ that the simple sequential test outperforms the SPRT on average, for this particular choice of δ ($5 \cdot 10^{-3}$). We would not expect that p is this close to θ for typical model checking problems. Furthermore, neither of the two tests give any valuable accuracy guarantees in the indifference region. If we do expect p to be very close to θ, and we want to know on which side of the threshold p really is, then we may have to resort to numerical solution techniques.

We can increase the accuracy of the model checking result by strengthening the test (decreasing α and β) or narrowing the indifference region. Figure 6.8 shows how the expected sample size for the two sampling plans depends on the half-width of the indifference region. The plots are for the symmetric polling system with $n = 10$ and two different values of θ and τ. We can see that it is generally more costly to narrow the



(a) $\theta = 0.5$

(b) $\theta = 0.9$

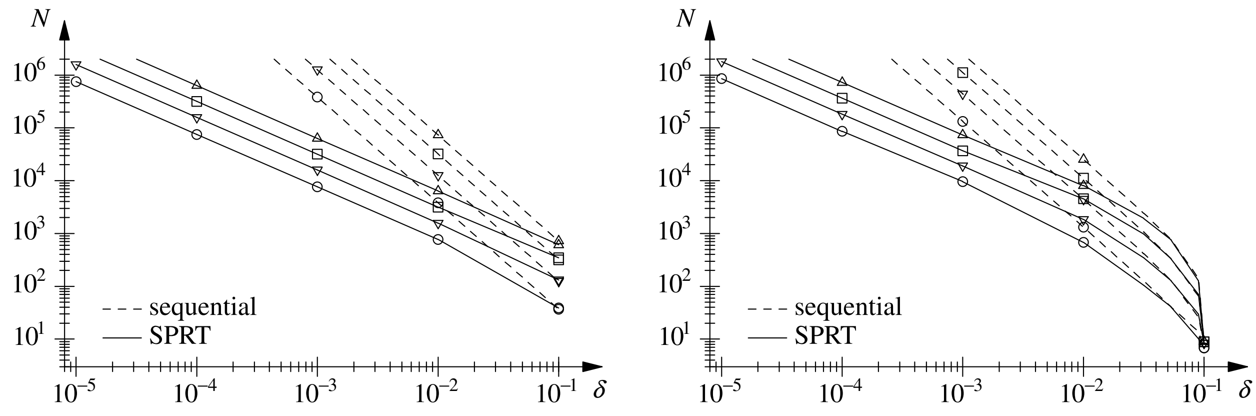
Figure 6.7: Sample size as a function of the formula time bound for the symmetric polling system ($n = 10$) in the vicinity of the indifference region for two different values of θ, using acceptance sampling with $2\delta = 10^{-2}$ and symmetric error bounds $\alpha = \beta$ equal to 10^{-8} ($\triangle$), 10^{-4} ($\square$), 10^{-2} ($\triangledown$), and 10^{-1} ($\circ$). The indifference region is indicated by a shaded area.

indifference region when using the simple sequential test rather than the SPRT. For example, we can have an indifference region of half-width 10^{-5} with the SPRT at essentially the same cost as 10^{-3} with the simple sequential test. For $\delta = 10^{-1}$ in the right plot, the upper border of the indifference region is 1, which means that both the SPRT and the simple sequential test become a curtailed single sampling plan. This explains the drop in expected sample size at this point.

6.2.2 "Five Nines"

For safety critical systems, we want to ensure that the probability of failure is very close to zero. While guaranteeing a zero probability of failure is usually unrealistic, it is not uncommon to require the failure probability of a safety critical system to be at most 10^{-4} or 10^{-5}. A failure probability of at most 10^{-5} means a success probability of $1 - 10^{-5} = 0.99999$, commonly referred to as "five nines." For such high accuracy requirements, it is typically best to use numerical solution techniques, but if the model is non-Markovian or has a large state space, this may not be a viable choice.

To use statistical hypothesis testing with a probability threshold $1 - 10^{-5}$, we need to use an indifference region with half-width at most 10^{-5}. An indifference region that narrow requires a large average sample size if the success probability is close to one, as we would expect it to be for a good system design. A possible



(a) $\theta = 0.5$, $n = 10$, $\tau = 10$

(b) $\theta = 0.9$, $n = 10$, $\tau = 14$

Figure 6.8: Sample size as a function of the half-width of the indifference region for the symmetric polling system, using acceptance sampling with symmetric error bounds $\alpha = \beta$ equal to 10^{-8} ($\triangle$), 10^{-4} ($\square$), 10^{-2} ($\triangledown$), and 10^{-1} ($\circ$).

solution is to set the indifference region to $(1-10^{-5}, 1)$ and use a curtailed single sampling plan. We need up to $n = \lceil \log \beta / \log(1 - 10^{-5}) \rceil$ observations for such a sampling plan, where β is the maximum probability that we accept the system as safe if the success probability is at most $1 - 10^{-5}$. We accept the system as safe if all n observations are positive, but reject the system as unsafe at the first negative observation. This means that if the success probability for the system is far below acceptable, we will quickly reject the system, but acceptance always requires n observations. Note, however, that we will never need more than n observations, so the maximum effort for verifying the system is known. Figure 6.9 plots the average verification time, as a function of the formula time bound, for the symmetric polling system ($n = 10$) with indifference regions $(0.99999, 1)$ and $(0.999985, 0.999995)$, of which the former leads us to use a curtailed single sampling plan. In the latter case (solid curves), the SPRT was used.

First, consider the indifference region with 1 as upper bound, which leads to a curtailed single sampling plan. We can see that for low values of τ, the average verification time is negligible, simply because we get a negative observation very quickly and reject the system design as unacceptable. As τ increases and the success probability approaches $1 - 10^{-5}$, the average sample size increases. As we pass the point at which the success probability exceeds $1 - 10^{-5}$ (roughly at $\tau = 29.57$), the sample size settles at around $2 \cdot 10^6$ for $\beta = 10^{-8}$. The verification time at this point is just under 11 minutes on our test machine (the average trajectory length is just over 23).

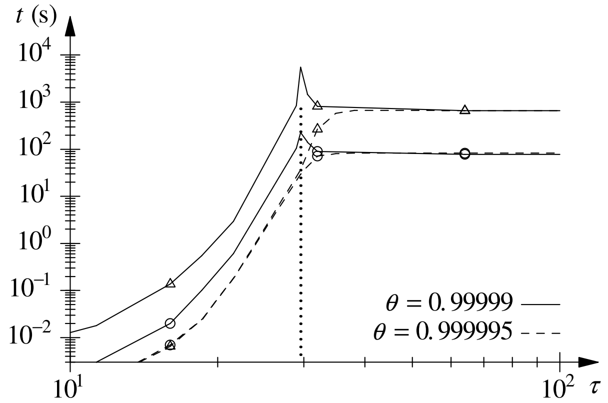


Figure 6.9: Verification time as a function of the formula time bound for the symmetric polling system ($n = 10$), using acceptance sampling with $2\delta = 10^{-5}$ and symmetric error bounds $\alpha = \beta$ equal to 10^{-8} ($\triangle$) and 10^{-1} ($\circ$).

β	$> .5$	$> .9$	$> .99$	$> .999$
10^{-8}	34.1	36.6	39.7	42.6
10^{-4}	33.2	35.8	38.9	41.8
10^{-2}	32.3	34.8	37.9	40.9
10^{-1}	31.3	33.9	37.0	40.0

Table 6.1: Minimum value of formula time bound τ, for the symmetric polling system ($\theta = 0.999995$ and $n = 10$), that leads to an acceptance probability of at least .5, .9, .99, and .999, respectively, for $2\delta = 10^{-5}$ and four different values of β.

We control the probability of error with the parameters α and β. By setting β low, we guarantee a low probability of accepting a poor system design, and by setting α low, we guarantee a low probability of rejecting a good system design. A curtailed single sampling plan is an efficient way of dealing with probability thresholds close to 1, but it gives us no control over the risk of rejecting a good system design, except that we will never reject a system design with success probability 1. This may lead us to reject many system designs that in practice are acceptable, or we may have to relax the system requirements. Table 6.1 shows the value of τ for the symmetric polling system that leads to acceptance with a certain probability for different values of β. For example, to guarantee that a poor system design is accepted with probability at most 10^{-8}, τ needs to be at least 42.6 for acceptance of the symmetric polling system with probability at least 0.999. In reality, the probability that $poll_1$ becomes true within τ time units is sufficiently high for $\tau = 29.57$, but using that time bound for verification would almost definitely lead us to reject the system.

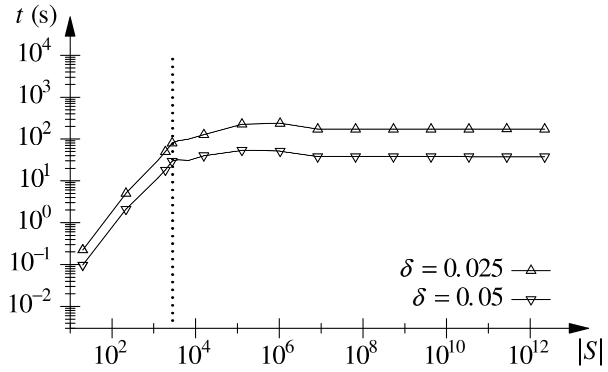
To ensure a non-trivial bound on the risk of rejecting an acceptable system design, we need to move the upper bound of the indifference region away from 1. Finding a single sampling plan for an indifference region as narrow as 10^{-5} is generally not feasible (cf. Figure 6.8), so we use only the SPRT in this case. This means that, in contrast to a curtailed single sampling plan, there is no upper bound on the sample size. The solid curves in Figure 6.9 show the average verification time for the SPRT with indifference region $(0.999985, 0.999995)$. We can see clear peaks in the verification time where the probability is close to $1 - 10^{-5}$. The price for moving the upper bound of the indifference region away from 1 is that verification can take over an hour on average instead of a few minutes. One of the 20 experiments for $\alpha = \beta = 10^{-8}$ required a sample size of over 35 million, which can be compared to a maximum sample size of just over 1.8 million for the curtailed single sampling plan with $\beta = 10^{-8}$.

6.2.3 Nested Probabilistic Operators

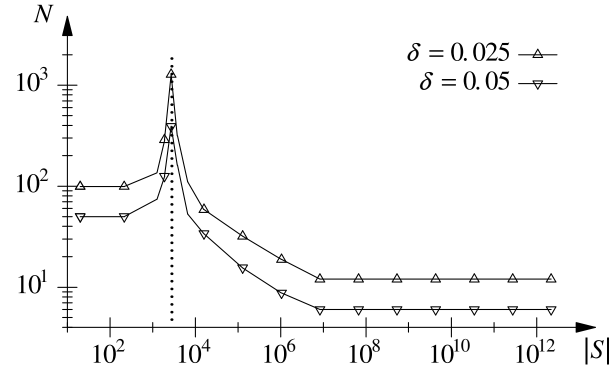
We use the robot grid world case study to show results of verification with nested probabilistic operators. We have proven that a statistical approach is possible even in the presence of nested probabilistic operators, with Theorem 5.8 being the key theoretical result. A practical concern, however, is that such verification could be costly, since each observation for the outer probabilistic operator involves an acceptance sampling test for the inner probabilistic operators. Nevertheless, our empirical results suggest that a statistical approach is, in fact, tractable.

Figure 6.10 shows empirical data for the robot grid world case study for verifying the UTSL formula $\mathcal{P}_{\geq 0.9}\big[\mathcal{P}_{\geq 0.5}\big[\Diamond^{[0,\tau_2]}\ c\big]\ \mathcal{U}^{[0,\tau_1]}\ x{=}n \wedge y{=}n\big]$. This formula asserts that the probability is high (at least 0.9) that the robot reaches the goal position while periodically communicating with the base station. The time bounds τ_1 and τ_2 were set to 100 and 9, respectively. We used the SPRT exclusively, with memoization enabled, and the heuristic proposed in Definition 5.3 to select the nested error bounds. It turns out that with $\tau_2 = 9$, the probability measure of the set of paths satisfying $\Diamond^{[0,\tau_2]}\ c$ is $1 - e^{-0.9} \approx 0.593$, independent of the start state. We used an indifference region with half-width δ independent of θ. For both values of δ that we used, $\delta = 0.05$ and $\delta = 0.025$, 0.593 is more than a δ-distance from the threshold 0.5 for the inner probabilistic operator, so we will have a low probability of erroneously verifying the path formula ($\mathcal{P}_{\geq 0.5}\big[\Diamond^{[0,\tau_2]}\ c\big]\ \mathcal{U}^{[0,\tau_1]}\ x{=}n \wedge y{=}n$) over sample trajectories. For the outer probabilistic operator, we used the symmetric error bounds $\alpha = \beta = 10^{-2}$. The heuristic gave us the symmetric nested error bounds 0.0153 and 0.00762 for $\delta = 0.05$ and $\delta = 0.025$, respectively.

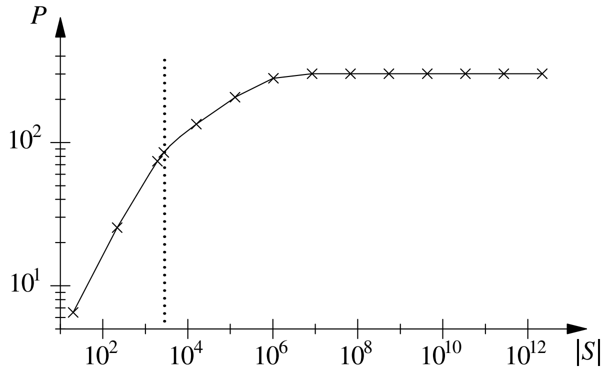
We can see in Figure 6.10(b) the familiar peak in the average sample where the value of the UTSL formula goes from true to false. Note, however, that the peak is not present in Figure 6.10(a), where the verification time is plotted as a function of the state space size. This is due to memoization. Figure 6.10(d) shows the fraction of unique states among all states visited along sample trajectories for the outer probabilistic operator. This graph is almost the mirror image of that for the average sample size. As we generate more sample trajectories, the probability increases that we visit states that have been visited before. With



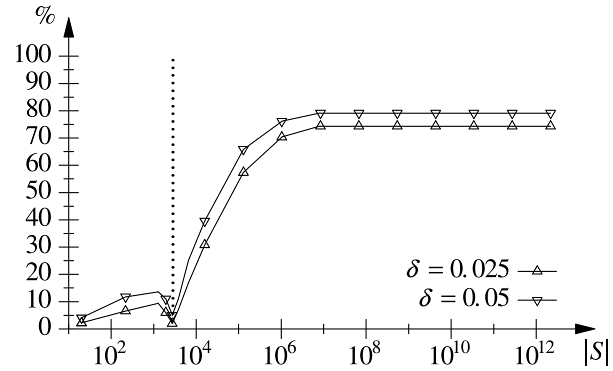
(a) Verification time as a function of state space size.



(b) Sample size as a function of state space size.



(c) Trajectory length as a function of state space size.



(d) Fraction of unique states among visited states.

Figure 6.10: Empirical results for the robot grid world ($\tau_1 = 100$ and $\tau_2 = 9$), using acceptance sampling with symmetric error bounds $\alpha = \beta = 10^{-2}$. The average trajectory length is the same for all values of δ. The dotted lines mark a change in the truth value of the formula being verified.

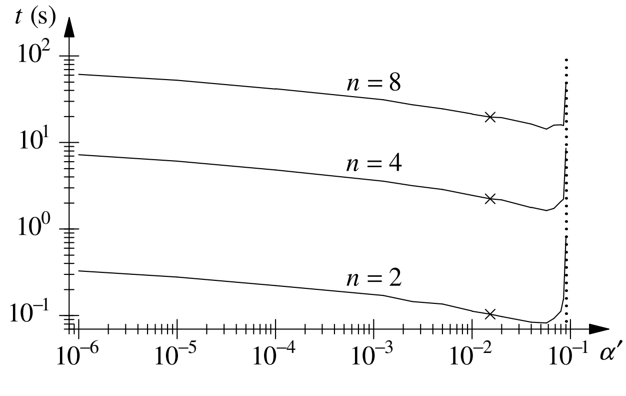


Figure 6.11: Verification time as a function of the nested error. The dotted line marks the maximum nested error.

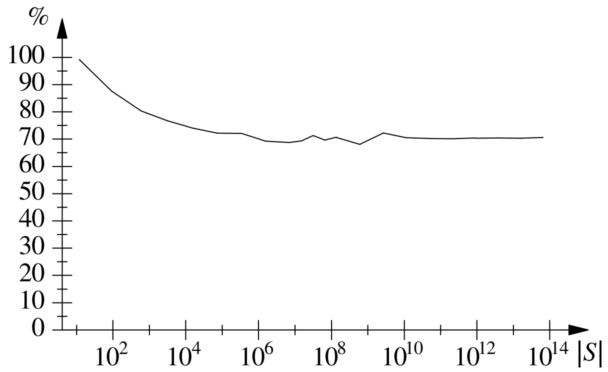


Figure 6.12: Fraction of verification time as a function of state space size for the symmetric polling system ($\tau = 20$) when using distributed acceptance sampling with two machines instead of one.

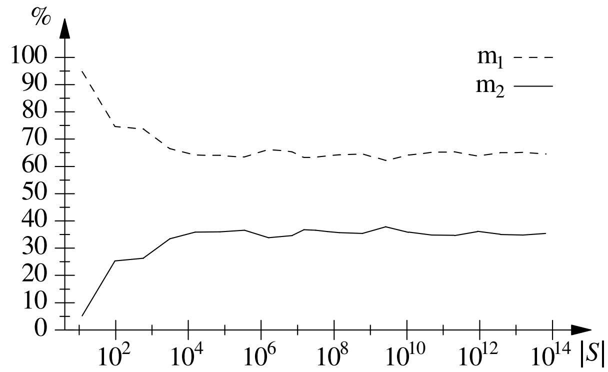


Figure 6.13: Distribution of workload as a function of state space size for the symmetric polling system ($\tau = 20$) when using distributed acceptance sampling with $m = 2$.

memoization, we do not need to verify nested probabilistic statements more than once in a visited state, so the cost per observation drops over time. The net effect is that total verification time is notably reduced. The price we pay for the improved efficiency is that we use more memory. However, the number of unique visited states is still only a tiny fraction of the total number of states for the robot grid world, resulting in modest memory requirements.

Figure 6.11 shows the effectiveness of our heuristic for selecting the nested error. We plot the verification time as a function of the symmetric nested error for $\delta = 0.05$ and three different values of n (the size of the grid). The cross on each curve marks the performance we get by using our heuristic. We do not obtain optimal performance, but we are only off by a factor of 1.3 to 1.4. Note that selecting a nested error that is too high or too low could easily result in a performance worse than optimal by orders of magnitude, so our heuristic does reasonably well.

6.2.4 Distributed Acceptance Sampling

Acceptance sampling may require millions of observations, but each observation represents an independent chance experiment. This means that we can carry out multiple experiments in parallel, which could result in a substantial reduction in verification time. When using a sequential sampling plan, we need to be careful not to introduce bias against observations that take a long time to generate. It is necessary to decide *a priori* on an order in which observations from nodes working in parallel will be taken into consideration, and not simply incorporate observations as they are generated. We addressed this problem in Section 5.3.1, where we proposed a way to schedule the processing of observations that dynamically adjusted to a heterogeneous environment (e.g. observations being generated on CPUs of different speed).

Figure 6.12 shows the reduction in verification time as a function of the state space size for the symmetric polling system when using two machines to generate observations. The first machine is equipped with a Pentium III 733 MHz processor. If we also generate observations, in parallel, on a machine with a Pentium III 500 MHz processor, we get the relative performance indicated by the solid curve. The verification time with two machines is roughly 70 percent of the verification time with a single machine. Figure 6.13 shows the fraction of observations used from each machine, with m_1 being the machine with a 733 MHz processor and m_2 being the machine with a 500 MHz processor. We can see that these fractions are in line with the relative performance of the two machines, and this is achieved without any explicit communication of performance characteristics.[1]

6.3 Comparison with Numerical Solution Method

To verify the UTSL formula $\mathcal{P}_{\bowtie\theta}\big[\Phi\ \mathcal{U}^{[0,\tau]}\ \Psi\big]$ in some state $s \in S$ of a model $\mathcal{M}$ with state space S, we can compute the probability $p = \mu(\{\sigma \in \mathit{Path}(\{\langle s, 0\rangle\}) \mid \mathcal{M}, \sigma, 0 \models \Phi\ \mathcal{U}^{[0,\tau]}\ \Psi\})$ numerically and test if $p \bowtie \theta$ holds.

First, as initially proposed by Baier et al. (2000), the problem is reduced to the computation of transient probabilities on a modified model $\mathcal{M}'$, where all states in $\mathcal{M}$ satisfying $\neg\Phi \vee \Psi$ have been made absorbing. The probability p is equal to the probability that we are in a state satisfying Ψ at time τ in model $\mathcal{M}'$. This probability can be computed using a technique called *uniformization* (also know as *randomization*), originally proposed by Jensen (1953). The computation of p is expressed as an infinite sum, with each term involving a matrix-vector multiplication. In practice, the infinite summation is truncated by using the techniques of Fox and Glynn (1988), so that the truncation error is bounded by an *a priori* error tolerance ϵ. The number of iterations required to achieve truncation error ϵ is R_ϵ. The value of R_ϵ is $q \cdot \tau + k\sqrt{2q \cdot \tau} + 3/2$,

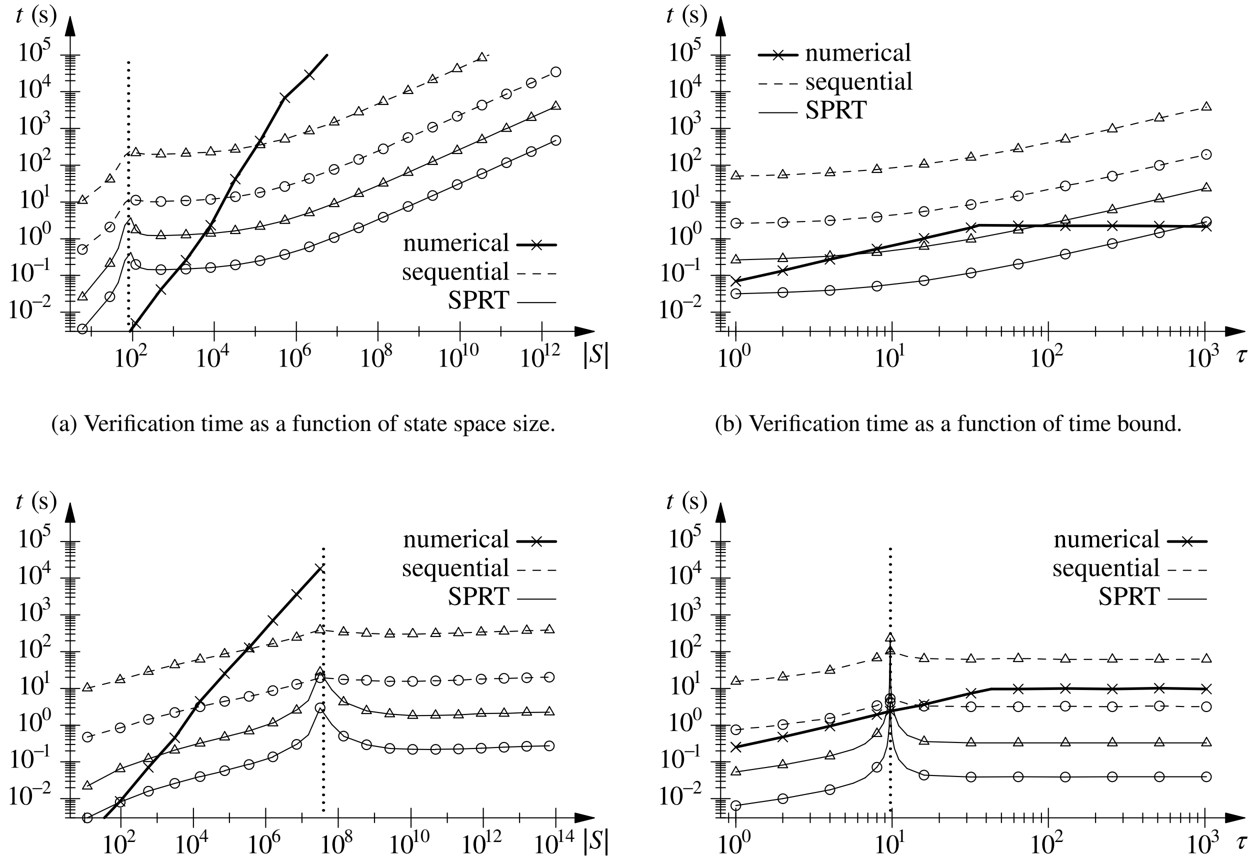
[1] Roughly 65 percent of the observations are generated by m_1. The CPU speed of m_1 (733 MHz) is just over 59 percent of the combined CPU speed for both m_1 and m_2 (1233 MHz), but this does not account for other factors (e.g. cache size) that also impact performance.

where q is the maximum exit rate for the model and k is $o\big(\sqrt{\log(1/\epsilon)}\big)$ (Fox and Glynn 1988). This means that the number of iterations grows very slowly as ϵ decreases. For large values of $q \cdot \tau$, the number of iterations is essentially $O(q \cdot \tau)$. Each iteration involves a matrix-vector multiplication and each such operation takes $O(M)$ time, where M is the number of non-zero entries in the rate matrix Q for the continuous-time Markov process $\mathcal{M}$. The time complexity for the numerical solution technique is therefore $O(q \cdot \tau \cdot M)$ (cf. Malhotra et al. 1994). This is in comparison to the theoretical time complexity $O(q \cdot \tau \cdot N_p)$ for our statistical solution method, where N_p is the expected sample size as a function of p. In the worst case M is $O(|S|^2)$, but is typically $O(|S|)$. N_p, on the other hand, is often much smaller than $|S|$ for large state spaces.

The number of iterations required by the numerical solution method can, in some cases, be reduced significantly through the use of steady-state detection (Reibman and Trivedi 1988; Malhotra et al. 1994; Younes et al. 2004). Further reduction is possible by using the sequential stopping rule described by Younes et al. (2004), although this does not reduce the asymptotic time complexity of the numerical solution method.

The limiting factor for the numerical solution method is typically memory. The space complexity for verifying the formula $\mathcal{P}_{\bowtie\theta}\big[\Phi\ \mathcal{U}^{[0,\tau]}\ \Psi\big]$ is $O(|S|)$ in most cases. For the results presented in this section, we use the hybrid approach proposed by Parker (2002), which uses flat representations of vectors and symbolic data structures, such as BDDs (Bryant 1986) and MTBDDs (Clarke et al. 1993; Bahar et al. 1993; Fujita et al. 1997), to represent matrices. With steady-state detection enabled, the hybrid approach requires storage of three double precision floating point vectors of size $|S|$, which for a memory limit of 800 MB means that systems with at most 35 million states can be analyzed. An alternative to symbolic data structures is sparse matrices. The space complexity is the same for both representations, and sparse matrices nearly always provide faster numerical computation, but symbolic representations of rate and probability matrices can exploit structure in the model and therefore require less memory in practice (Kwiatkowska et al. 2004).

Figure 6.14 compares the performance of the numerical and the statistical solution methods for the tandem queuing network and symmetric polling system case studies. The truncation error (ϵ) for the numerical solution method was set to 10^{-10}. This error bound cannot be compared directly with the error bounds for the statistical solution method, but the performance of the numerical method does not vary much with the choice of ϵ. We can see, clearly, that the numerical solution method is faster for small state spaces, but that the statistical solution method scales better with an increase in the size of the state space. For a fixed model size, and with increasing time bound, the numerical solution method compares much more favorably. The



(a) Verification time as a function of state space size.

(b) Verification time as a function of time bound.

(c) Verification time as a function of state space size.

(d) Verification time as a function of time bound.

Figure 6.14: Comparison of numerical and statistical probabilistic model checking for the tandem queuing network (top) and the symmetric polling system (bottom). For the statistical solution method, results are shown for symmetric error bounds $\alpha = \beta$ equal to 10^{-8} ($\triangle$) and 10^{-1} ($\circ$).

verification time remains constant once steady-state detection kicks in. Still, for the symmetric polling station, the verification time for the statistical solution remains constant for large time bounds as well, because all sample trajectories are terminated prematurely when reaching a state satisfying $poll_1$.

The numerical solution method has the same asymptotic time complexity for verifying a UTSL formula in a single state as in all states simultaneously (Katoen et al. 2001). This is a great benefit when dealing with nested probabilistic operators. Consider the UTSL formula $\mathcal{P}_{\geq 0.9}\big[\mathcal{P}_{\geq 0.5}\big[\Diamond^{[0,\tau_2]}\ c\big]\ \mathcal{U}^{[0,\tau_1]}\ x{=}n \wedge y{=}n\big]$ for the robot grid world. The time complexity for the numerical solution method is $O(q \cdot \tau_1 \cdot M + q \cdot \tau_2 \cdot M)$, which is essentially the same as $O(q \cdot \tau_1 \cdot M)$ for $\tau_2 < \tau_1$. The statistical solution method, on the other

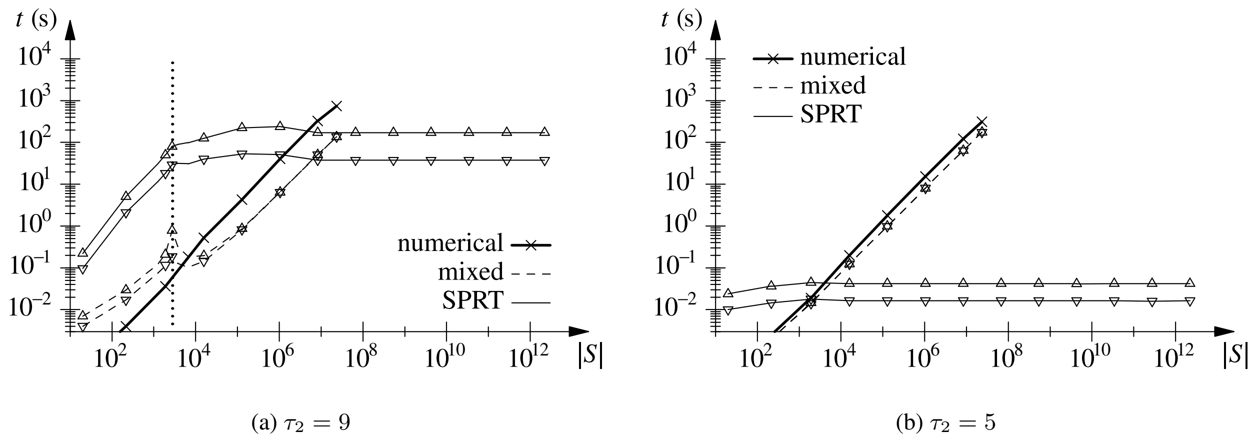


Figure 6.15: Comparison of numerical, mixed, and statistical solution methods for formulae with nested probabilistic operators. For the statistical and mixed solution methods, results are shown for δ equal to 0.025 ($\triangle$) and 0.05 ($\triangledown$).

hand, is definitely more costly in the presence of nested probabilistic operators. Younes et al. (2004) have suggested a mixed solution method, which uses the numerical approach for nested probabilistic operators and the statistical approach for top-most probabilistic operators. This mixed approach shares performance characteristics of both solution methods, but is limited by memory in the same way as the pure numerical solution method. We can see this in Figure 6.15, where we compare the three solution methods for the robot grid world case study using two different values of τ_2. For $\tau_2 = 9$, the statistical solution method is slower for state spaces up to 10^6 or 10^7, but handles much larger state spaces than the other two solution methods without running out of memory. For $\tau_2 = 5$, the nested probabilistic statement is false in all states. The pure statistical approach benefits from this fact because sample trajectories will typically not extend beyond the initial state. The numerical and mixed solution methods scale much worse in this case.

In summary, the empirical results presented in this chapter have shown that the performance of the statistical solution method depends on several factors, in particular the parameters δ, α, and β. The SPRT is generally orders of magnitude faster than a single sampling plan with the same strength, although there are exceptions to this rule. Memoization is important for making a pure statistical approach tractable in the presence of nested probabilistic operators. Numerical solution methods are faster than statistical methods for smaller state spaces, and can benefit greatly from the use of steady-state detection, but statistical methods scale better as the size of the state space increases.

Chapter 7

Probabilistic Verification for “Black-Box” Systems

So far, we have assumed that a model is available of the system that we want to verify. Given a model, we can apply either numerical or statistical solution methods for probabilistic model checking. Numerical techniques provide highly accurate results, but rely on strong assumptions regarding the dynamics of the systems they are used to analyze. Statistical techniques require only that the dynamics of a system can be simulated, and can therefore be used for a larger class of stochastic processes. The results produced by statistical methods are only probabilistic, however, and attaining high accuracy tends to be costly.

For some systems, it may not even be feasible to assume that we can simulate their behavior. Sen et al. (2004) consider the verification problem for such “black-box” systems. Here, “black-box” means that the system cannot be controlled to generate execution traces, or trajectories, on demand starting from arbitrary states. This is a reasonable assumption, for instance, for a system that has already been deployed and for which we are given only a set of trajectories generated during actual execution of the system. We are then asked to verify a probabilistic property of the system based on the information provided to us as a fixed set of trajectories. Statistical solution techniques are certainly required to solve this problem. The statistical method described in Chapter 4 cannot be used to verify “black-box” systems, however, because it depends on the ability to generate trajectories on demand.

Sen et al. (2004) present an alternative solution method for verification of “black-box” systems based on statistical hypothesis testing with fixed sample sizes. In this chapter, we improve upon their algorithm by making sure to always accept the most likely hypothesis, and we correct their procedure for verifying nested probabilistic properties. Differences between the two approaches are discussed in detail towards the end of this chapter.

The algorithm we present for verification of "black-box" systems can handle the full logic UTSL, including properties without finite time bounds, although the accuracy of the result for such properties may be poor. Our algorithm, like that of Sen et al. (2004), makes no guarantees regarding accuracy. Instead of respecting some *a priori* bounds on the probability of error, the algorithm computes a p-value for the result, which is a measure of confidence. This is really the best we can do, provided that we cannot generate trajectories for the system as we see fit and instead are restricted to using a predetermined set of trajectories.

The algorithm presented in this chapter is complementary to the statistical model checking algorithm presented in Chapter 5, and is useful under different assumptions. If we cannot generate trajectories for a system on demand, then the algorithm presented here still allows us to reach conclusions regarding the behavior of the system. If, however, we have a model of a system so that we can simulate its dynamics, then we are better off with the approach of Chapter 5 as it gives us full control over the probability of obtaining an incorrect result.

7.1 "Black-Box" Probabilistic Systems and Verification

Formally, we define a "black-box" probabilistic system in terms of what we know (or rather, do not know) regarding the probability measure over sets of trajectories.

Definition 7.1 ("Black-Box" Probabilistic System). A "black-box" probabilistic system is a stochastic discrete event system for which the probability measure μ over sets of trajectories with common prefix is not fully specified and cannot be sampled from.

We thus refer to a stochastic discrete event system $\mathcal{M}$ as a "black-box" system if we lack an exact definition of the probability measure μ over sets of trajectories of $\mathcal{M}$. We assume that we cannot even sample trajectories according to μ, as stated in Definition 7.1. Thus, in order to solve a verification problem $\langle \mathcal{M}, \mu_0, \Phi \rangle$ for a "black-box" system $\mathcal{M}$, we must rely on an external source to provide a sample set of n trajectories for $\mathcal{M}$ that is representative of the probability measure μ and the initial state distribution μ_0.

We further assume that we are provided only with *truncated* trajectories, because infinite trajectories would require infinite memory to store.

We will use statistical hypothesis testing to verify properties of a "black-box" system given a sample of n truncated trajectories. Since we rely on statistical techniques, we will typically not know with certainty if the result we produce is correct. The method we present for verification of "black-box" systems computes a p-value for a verification result, which is a value in the interval $[0, 1]$ with values closer to 0 representing higher confidence in the result and a p-value of 0 representing certainty (Hogg and Craig 1978, pp. 255–256). We start by assuming that Φ is free of nested probabilistic operators. Later on, we consider UTSL formulae with nested probabilistic operators, which just as with regular statistical probabilistic model checking cannot be handled in a meaningful way without making rather strong assumptions regarding the dynamics of the "black-box" system.

7.1.1 Verification without Nested Probabilistic Operators

Given a state s, verification of a UTSL formula $x \sim v$ is trivial. We can simply read the value assigned to x in state s and compare it to v. We consider the remaining three cases in more detail, starting with the probabilistic operator $\mathcal{P}_{\bowtie\theta}[\cdot]$. Recall that the objective is to produce a Boolean result annotated with a p-value.

Probabilistic Operator

Consider the problem of verifying the UTSL formula $\mathcal{P}_{\bowtie\theta}[\varphi]$ in state s of a stochastic discrete event system $\mathcal{M}$. As before, let X_i be a random variable representing the verification of the path formula φ over a trajectory for $\mathcal{M}$ drawn according to the probability measure $\mu(Path(\{\langle s, 0\rangle\}))$. If we choose $X_i = 1$ to represent the fact that φ holds over a random trajectory, and $X_i = 0$ to represent the opposite fact, then X_i is a Bernoulli variate with parameter $p = \mu(\{\sigma \in Path(\{\langle s, 0\rangle\}) \mid \sigma, 0 \models \varphi\})$, i.e. $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$. In order to verify $\mathcal{P}_{\bowtie\theta}[\varphi]$, we can make observations of X_i and use statistical hypothesis testing to determine if $p \bowtie \theta$ is likely to hold. An observation of X_i, denoted x_i, is the verification of φ over a specific trajectory σ_i. If σ_i satisfies the path formula φ, then $x_i = 1$, otherwise $x_i = 0$.

In our case, we are given n truncated trajectories for a "black-box" system that we can use to generate observations of X_i. Each observation is obtained by verifying the path formula φ over one of the truncated trajectories. This is straightforward given a truncated trajectory $\{\langle s_0, t_0\rangle, \ldots, \langle s_{k-1}, t_{k-1}\rangle, s_k\}$, provided that φ does not contain any probabilistic operators. For $\varphi = X^I\ \Phi$, we just check if $t_0 \in I$ and $s_1 \models \Phi$. For $\varphi = \Phi\ \mathcal{U}^I\ \Psi$, we traverse the trajectory until we find a state s_i such that one of the following conditions holds, with T_i defined as in (2.18) to be the time at which state s_i is entered:

1. $(s_i \models \neg\Phi) \wedge ((T_i \notin I) \vee (s_i \models \neg\Psi))$

2. $(T_i \in I) \wedge (s_i \models \Psi)$

3. $((T_i, T_{i+1}) \cap I \neq \emptyset) \wedge (s_i \models \Phi) \wedge (s_i \models \Psi)$

In the first case, $\Phi\ \mathcal{U}^I\ \Psi$ does not hold over the trajectory, while in the last two cases the time-bounded until formula does hold. This is the same procedure as was used in Chapter 5 for generating observations for the verification of probabilistic statements. Note, however, that in this case we may not always be able to determine the value of φ over all trajectories because the trajectories that are provided to us are assumed to be truncated. Previously, we assumed that we could always generate a sufficient prefix of a trajectory so that the truth value of a path formula could be determined.

We consider the case $\mathcal{P}_{\geq\theta}[\varphi]$ in detail, noting that $\mathcal{P}_{\leq\theta}[\varphi]$ can be handled in the same way simply by reversing the value of each observation. We want to test the hypothesis $H_0 : p \geq \theta$ against the alternative hypothesis $H_1 : p < \theta$ by using the n observations $x_1, \ldots, x_n$ of the Bernoulli variates $X_1, \ldots, X_n$. To do so, we specify a constant c. If $\sum_{i=1}^{n} x_i$ is greater than c, then hypothesis H_0 is accepted, i.e. $\mathcal{P}_{\geq\theta}[\varphi]$ is determined to hold. Otherwise, if the given sum is at most c, then hypothesis H_1 is accepted, meaning that $\mathcal{P}_{\geq\theta}[\varphi]$ is determined not to hold. The constant c should be chosen so that it becomes roughly equally likely to accept H_0 as H_1 if p equals θ. The pair $\langle n, c\rangle$ is a single sampling plan, as described in Section 2.2.

We know from before that by using a single sampling plan $\langle n, c\rangle$, we accept hypothesis H_1 with probability $F(c; n, p)$, and consequently hypothesis H_0 is accepted with probability $1 - F(c; n, p)$. Ideally, we should choose c such that $F(c; n, \theta) = 0.5$, but it is not always possible to attain equality because the binomial distribution is a discrete distribution. The best we can do is to choose c such that $|F(c; n, \theta) - 0.5|$ is minimized. We can readily compute the desired c using (2.3).

We now have a way to decide whether to accept or reject the hypothesis that $\mathcal{P}_{\geq\theta}[\varphi]$ holds, but we also want to report a value reflecting the confidence in our decision. For this purpose, we compute the p-value for a decision. The p-value is defined as the probability of the sum of observations being at least as extreme as the one obtained provided that the hypothesis that was not accepted holds. The p-value for accepting H_0 when $\sum_{i=1}^{n} x_i = d$ is $\Pr[\sum_{i=1}^{n} X_i \geq d \mid p < \theta] < F(n-d; n, 1-\theta) = 1 - F(d-1; n, \theta)$, while the p-value for accepting H_1 is $\Pr[\sum_{i=1}^{n} X_i \leq d \mid p \geq \theta] \leq F(d; n, \theta)$. The following theorem provides justification for our choice of the constant c.

Theorem 7.1 (Minimization of p-value). *By choosing c to minimize $|F(c; n, \theta) - 0.5|$ when testing $H_0 : p \geq \theta$ against $H_1 : p < \theta$ using a single sampling plan $\langle n, c \rangle$, the hypothesis with the lowest p-value is always accepted.*

Proof. Hypothesis H_1 is accepted only if $d \leq c$, which means that the p-value for H_1 under these circumstances is at most $F(c; n, \theta)$. The p-value for H_0 if $d \leq c$ would be at least $1 - F(c-1; n, \theta)$. We know that $F(c-1; n, \theta) < F(c; n, \theta)$ and by assumption that $|F(c-1; n, \theta) - 0.5| > |F(c; n, \theta) - 0.5|$. It follows that $F(c; n, \theta) < 1 - F(c-1; n, \theta)$ as required. For $d > c$, the p-value for acceptance of H_1 would be at least $F(c+1; n, \theta)$. The p-value for acceptance of H_0 when $d > c$, on the other hand, is at most $1 - F(c; n, \theta)$. We know that $F(c+1; n, \theta) > F(c; n, \theta)$ and by assumption that $|F(c+1; n, \theta) - 0.5| > |F(c; n, \theta) - 0.5|$. Consequently, $1 - F(c; n, \theta) < F(c+1; n, \theta)$ and our choice of c ensures that the hypothesis with the lowest p-value is always accepted. $\square$

In practice, it is unnecessary to compute c. It is more convenient simply to compute the p-value of each hypothesis and accept the hypothesis with the lowest p-value.

Example 7.1. Consider the problem of verifying the UTSL formula $\Phi = \mathcal{P}_{\geq 0.9}\big[\Diamond^{[0,100]}\ x{=}1\big]$ in a state satisfying $x{=}0$ for a "black-box" system that in reality is the continuous-time Markov process shown in Figure 7.1. The probability measure of trajectories starting in state $x{=}0$ and satisfying $\Diamond^{[0,100]}\ x{=}1$ is $1 - e^{-1} \approx 0.63$ for this system, so the UTSL formula does not hold, but we would of course not know this unless we had access to the model. Assume that we are given a set of 100 truncated trajectories, of which 63 satisfy the path formula $\Diamond^{[0,100]}\ x{=}1$ and 37 do not satisfy the given path formula. Thus, $n = 100$ and $d = 63$. The p-value for H_0 is $1 - F(62; 100, 0.9) \approx 1 - 10^{-13}$, while the p-value for H_1 is $F(63, 100, 0.9) \approx 5.48 \cdot 10^{-13}$. The hypothesis with the lowest p-value is H_1, so we conclude that Φ does not hold.

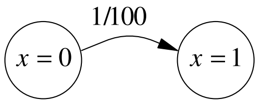


Figure 7.1: A simple two-state continuous-time Markov process.

In the analysis so far we have been assuming that the value of φ can be determined over all n truncated trajectories. Now, consider the case where we are unable to verify the path formula φ over some of the n truncated trajectories. This would happen if we are verifying $\Phi\ \mathcal{U}^I\ \Psi$ over a trajectory that has been truncated before either $\neg\Phi \vee \Psi$ is satisfied or time exceeds all values in I. We cannot simply ignore such trajectories: it is assumed that the *entire* set of n trajectories is representative of the measure μ, but the subset of truncated trajectories for which we can determine the value of φ is not guaranteed to be a representative sample for this measure.

Example 7.2. Consider the same problem as in Example 7.1. Assume that we are provided with a set of 100 truncated trajectories for the system, and that all trajectories have been truncated before time 50. Some of these trajectories, on average roughly 39 in every 100, will satisfy the path formula $\Diamond^{[0,100]}\ x{=}1$, while the remaining truncated trajectories will not contain sufficient information for us to determine the validity of the path formula over these trajectories. An analysis based solely on the trajectories over which the path formula can be decisively verified would be severely biased. If the number of positive observations is exactly 39, with 61 undetermined observations, we would wrongly conclude that Φ holds with p-value $1 - F(38; 39, 0.9) \approx 0.0164$, which implies a fairly high confidence in the result.

Let n' be the number of observations whose value we can determine and let d' be the sum of these n' observations. We then know that the sum of all observations, d, is at least d' and at most $d'{+}n{-}n'$. If $d' > c$, then hypothesis H_0 can be safely accepted. Instead of a single p-value, we associate an interval of possible p-values with the result: $[F(n' - d'; n, 1 - \theta), F(n - d'; n, 1 - \theta)]$. Conversely, if $d' + n - n' \leq c$, then hypothesis H_1 can be accepted with p-value in the interval $[F(d'; n, \theta), F(d' + n - n'; n, \theta)]$. If, however, $d' \leq c$ and $d' + n - n' > c$, then it is not clear which hypothesis should be accepted. We could in this case say that we do not have enough information to make an informed choice. Alternatively, we could accept one of the hypotheses with its associated p-value interval. We prefer to always make some choice, and we recommend choosing H_0 if $F(n - d'; n, 1 - \theta) \leq F(d' + n - n'; n, \theta)$ and H_1 otherwise. This strategy minimizes the maximum possible p-value. Alternatively, we could minimize the minimum possible p-value by instead choosing H_0 if $F(n'-d'; n, 1-\theta) \leq F(d; n, \theta)$ and H_1 otherwise. Note that this way of treating truncated trajectories makes our approach work even for unbounded until formulae $\Phi\ \mathcal{U}\ \Psi$, although we would typically expect the result to be highly uncertain for such formulae.

Example 7.3. Consider the same situation as in Example 7.2, with 39 positive and 61 undetermined observations. The p-value for accepting the UTSL formula $\Phi = \mathcal{P}_{\geq 0.9}\big[\Diamond^{[0,100]}\ x{=}1\big]$ as true lies in the interval $[F(0; 100, 0.1), F(61, 100, 0.1)] \approx [2.65 \cdot 10^{-5}, 1 - 3.77 \cdot 10^{-15}]$. For the opposite decision, we get the p-value interval $[F(39; 100, 0.9), F(100; 100, 0.9)] \approx [1.59 \cdot 10^{-35}, 1]$. Both intervals are almost equally uninformative, so no matter what decision we make, we will have a high uncertainty in the result. We would accept Φ as true if we prefer to minimize the maximum possible p-value, and we would reject Φ as false if we instead prefer to minimize the minimum possible p-value, but in both cases we have a maximum p-value well above 0.5. This is in sharp contrast to the faulty analysis suggested in Example 7.2, which led to an acceptance of Φ as true with a low p-value.

Composite State Formulae

To verify $\neg\Phi$, we first verify Φ. If we conclude that Φ has a certain truth value with p-value pv, then we conclude that $\neg\Phi$ has the opposite truth value with the same p-value. To motivate this, consider the case $\neg\mathcal{P}_{\geq\theta}[\varphi]$. To verify $\mathcal{P}_{\geq\theta}[\varphi]$, we test the hypothesis $H_0 : p \geq \theta$ against $H_1 : p < \theta$ as stated above. Note, however, that $\neg\mathcal{P}_{\geq\theta}[\varphi] \equiv \mathcal{P}_{<\theta}[\varphi]$, which could be posed as the problem of testing the hypothesis $H_0' : p < \theta$ against $H_1' : p \geq \theta$. Since $H_0' = H_1$ and $H_1' = H_0$, we can simply negate the result of verifying $\mathcal{P}_{\geq\theta}[\varphi]$ while maintaining the same p-value.

For a conjunction $\Phi \wedge \Psi$, we have to consider four cases. First, if we verify Φ to hold with p-value pv_Φ and Ψ to hold with p-value pv_Ψ, then we conclude that $\Phi \wedge \Psi$ holds with p-value $\max(pv_\Phi, pv_\Psi)$. Second, if we verify Φ not to hold with p-value pv_Φ, while verifying that Ψ holds, then we conclude that $\Phi \wedge \Psi$ does not hold with p-value pv_Φ. The third case is analogous to the second with Φ and Ψ interchanged. Finally, if we verify Φ not to hold with p-value pv_Φ and Ψ not to hold with p-value pv_Ψ, then we conclude that $\Phi \wedge \Psi$ does not hold with p-value $\min(pv_\Phi, pv_\Psi)$. This is similar to the result of Theorem 5.4, but for p-values instead of bounds on the type I and II error probabilities.

Before proving the results above, let us give an intuitive justification. In order for $\Phi \wedge \Psi$ to hold, both Φ

and Ψ must hold, so we cannot be any more confident in the result for $\Phi \wedge \Psi$ than we are in the result for the individual conjuncts, thus the maximum in the first case. To conclude that $\Phi \wedge \Psi$ does not hold, however, we only need to be convinced that one of the conjuncts does not hold. In case we think exactly one of the conjuncts holds, then the result for the conjunction will be based solely on this conviction and the p-value for the conjunct we think holds should not matter. This covers the second and third cases. In the fourth case, we have two sources (not necessarily independent) telling us that the conjunction is false. We therefore have no reason to be less confident in the result for the conjunction than in the result for each of the conjuncts, hence the minimum in this case.

For a mathematical derivation of the given expressions, we consider the formula $\mathcal{P}_{\geq \theta_1}[\varphi_1] \wedge \mathcal{P}_{\geq \theta_2}[\varphi_2]$. Let d_i denote the number of trajectories that satisfy φ_i. Provided we accept the conjunction as true, which means we accept each conjunct as true, the p-value for this result is

$$\Pr[\sum_{i=1}^{n} X_i^{(1)} \geq d_1 \wedge \sum_{i=1}^{n} X_i^{(2)} \geq d_2 \mid p_1 < \theta_1 \vee p_2 < \theta_2] \ . \tag{7.1}$$

To compute this p-value, consider the three ways in which $p_1 < \theta_1 \vee p_2 < \theta_2$ can be satisfied (cf. Sen et al. 2004). We know from elementary probability theory (Lemma 5.3) that

$$\Pr[A \wedge B] \leq \min(\Pr[A], \Pr[B]) \tag{7.2}$$

for arbitrary events A and B. From this fact, and assuming that pv_i is the p-value associated with the verification result for $\mathcal{P}_{\geq \theta_i}[\varphi_i]$, we derive the following:

1. $\Pr[\sum_{i=1}^{n} X_i^{(1)} \geq d_1 \wedge \sum_{i=1}^{n} X_i^{(2)} \geq d_2 \mid p_1 < \theta_1 \wedge p_2 < \theta_2] \leq \min(pv_1, pv_2)$
2. $\Pr[\sum_{i=1}^{n} X_i^{(1)} \geq d_1 \wedge \sum_{i=1}^{n} X_i^{(2)} \geq d_2 \mid p_1 < \theta_1 \wedge p_2 \geq \theta_2] \leq \min(pv_1, 1) = pv_1$
3. $\Pr[\sum_{i=1}^{n} X_i^{(1)} \geq d_1 \wedge \sum_{i=1}^{n} X_i^{(2)} \geq d_2 \mid p_1 \geq \theta_1 \wedge p_2 < \theta_2] \leq \min(1, pv_2) = pv_2$

We take the maximum over these three cases to obtain a bound for (7.1), which gives us $\max(pv_1, pv_2)$.

For the same formula, but now assuming we have verified both conjuncts to be false, we compute the p-value as

$$\Pr[\sum_{i=1}^{n} X_i^{(1)} \leq d_1 \wedge \sum_{i=1}^{n} X_i^{(2)} \leq d_2 \mid p_1 \geq \theta_1 \wedge p_2 \geq \theta_2] \ . \tag{7.3}$$

It follows immediately from (7.2) that $\min(pv_1, pv_2)$ is a bound for (7.3), which is the desired result.

7.1.2 Verification with Nested Probabilistic Operators

If we allow nested probabilistic operators, verification of UTSL formulae for "black-box" stochastic discrete event systems becomes much harder. Consider the formula $\mathcal{P}_{\geq \theta}\big[\Diamond^{[0,100]}\ \mathcal{P}_{\geq \theta'}[\varphi]\big]$. In order to verify this formula, we must test if $\mathcal{P}_{\geq \theta'}[\varphi]$ holds at some time $t \in [0, 100]$ along the set of trajectories that we are given. Unless the time domain T is such that there is a finite number of time points in a finite interval, then we potentially have to verify $\mathcal{P}_{\geq \theta'}[\varphi]$ at an infinite or even uncountable number of points along a trajectory, which clearly is infeasible. We made the same observation regarding verification of systems for which we can generate trajectories on demand. The situation is even worse, however, for "black-box" systems. Even if $T = \mathbb{Z}^*$, so that we only have to verify nested probabilistic formulae at a finite number of points, we still have to take the entire prefix of the trajectory into account at each time point. We are given a fixed set of trajectories, and we can use only the subset of trajectories with a matching prefix to verify a nested probabilistic formula. It is thus likely that we will have few trajectories available to use for verifying nested probabilistic formulae. In the worst case, there will be only a single matching prefix, in which case the uncertainty in the result will be overwhelming.

We can get around this problem by assuming that the "black-box" system is a Markov process. Under the Markov assumption, as mentioned earlier, we only have to take the last state along a trajectory prefix into consideration. Consequently, *any* suffix of a truncated trajectory starting at a specific state s can be regarded as representative of the probability measure $\mu(\{\langle s, 0\rangle\})$. This makes more trajectories available for the verification of nested probabilistic formulae.

Another complicating factor in the verification of $\mathcal{P}_{\geq \theta}[\varphi]$, where φ contains nested probabilistic operators, is that we cannot verify φ over trajectories without some uncertainty in the result. This means that we no longer obtain observations of the random variables X_i, as defined above, but instead we observe some other random variables Y_i, related to X_i through bounds on the observation error.

To compute a p-value for nested verification, we assume that $\Pr[Y_i = 0 \mid X_i = 1] \leq \alpha$ and $\Pr[Y_i = 1 \mid X_i = 0] \leq \beta$. We can make this assumption if we introduce indifference regions in the verification of nested probabilistic formulae and use the procedure described in Chapter 5 to verify path formulae over truncated trajectories. By Lemma 5.7, we have the following bounds: $p(1-\alpha) \leq \Pr[Y_i = 1] \leq 1-(1-p)(1-\beta)$. The p-value for accepting $\mathcal{P}_{\geq \theta}[\varphi]$ as true when the sum of the observations is d is $\Pr[\sum_{i=1}^{n} Y_i \geq d \mid p < \theta] < F(n-d; n, (1-\theta)(1-\beta))$. The p-value for the opposite decision is $\Pr[\sum_{i=1}^{n} Y_i \leq d \mid p \geq$

$\theta] \leq F(d; n, \theta(1-\alpha))$. Since $F(d; n, p)$ increases as p decreases, we see that the p-value increases as the error bounds α and β increase, which makes perfect sense. As was suggested earlier, we can minimize the p-value of the verification result by computing the p-values of both hypotheses and accept the one with the lowest p-value.

We can let the user specify a parameter δ_0 that controls the relative width of the indifference regions. A nested probabilistic formula $\mathcal{P}_{\geq\theta}[\varphi]$ is verified with indifference region of half-width $\delta = \delta_0\theta$ if $\theta \leq 0.5$ and $\delta = \delta_0(1-\theta)$ otherwise. The verification is carried out using acceptance sampling as before, but with hypotheses $H_0 : p \geq \theta + \delta$ and $H_1 : p \leq \theta - \delta$. Instead of reporting a p-value, as is done for top-level probabilistic operators, we report bounds for the type I error probability of the sampling plan in use if H_1 is accepted and the type II error probability if H_0 is accepted. In our case, assuming a sampling plan $\langle n, c\rangle$ is used, the type I error bound is $1 - F(c; n, \theta + \delta)$ and the type II error bound is $F(c; n, \theta - \delta)$. The difference from the procedure described in Chapter 5 is that we compute the error bounds that we can achieve for subformulae with a fixed sample size instead of computing the sample size required to achieve certain error bounds. We can then use Theorem 5.4 to compute error bounds for composite UTSL formulae and path formulae with an until operator. As error bounds for the computation of the p-value for a top-level probabilistic operator, we simply take the maximum error bounds for the verification of the path formula over all trajectories.

7.2 Comparison with Related Work

The idea of using statistical hypothesis testing for verification of “black-box” systems was first proposed by Sen et al. (2004). This section highlights the differences between their approach and the approach presented in this chapter.

First, consider the verification of a probabilistic formula $\mathcal{P}_{\geq\theta}[\varphi]$. Our approach is essentially the same as theirs: given a constant c, accept if $\sum_{i=1}^{n} X_i > c$ and reject otherwise. Their choice of c is different, however, and is essentially based on De Moivre’s (1738) normal approximation for the binomial distribution. Their acceptance condition is $\sum_{i=1}^{n} X_i \geq n\theta$, which corresponds to choosing c to be $\lceil n\theta \rceil - 1$. The mean of the binomial distribution $B(n, \theta)$ is $n\theta$, so this would be the right thing to do if $\sum_{i=1}^{n} X_i$ can be assumed to have a normal distribution. De Moivre showed that this is approximately the case for large n if X_i

are Bernoulli variates, but the approximation is poor for moderate values of n or if θ is not close to 0.5. Their algorithm, as a consequence, will under some circumstances accept a hypothesis with a larger p-value than the alternative hypothesis. By choosing c as we do, without relying on the normal approximation, we guarantee that the hypothesis with the smallest p-value is always accepted (Theorem 7.1). Consider the formula $\mathcal{P}_{\geq 0.01}[\varphi]$, for example, with $n = 501$ and $d = 5$. Our procedure would accept the formula as true with p-value 0.562, while the the algorithm of Sen et al. would reject the formula as false with p-value 0.614. The difference is not of great significance, but it is still worth pointing out because it demonstrates the danger of using the normal approximation for the binomial distribution. With today's fast digital computers, it is hard to motivate using this assumption.

The second improvement over the method presented by Sen et al. is in the calculation of the p-value for the verification of a conjunction $\Phi \wedge \Psi$ when both conjuncts have been verified to be false. They state that the p-value is $pv_\Phi + pv_\Psi$, but this is too conservative. There is no reason to believe that the confidence in the result for $\Phi \wedge \Psi$ would be *lower* (i.e. the p-value *higher*) if we are convinced that both conjuncts are false. We have shown that the p-value in this case is bounded by $\min(pv_\Phi, pv_\Psi)$, which intuitively makes more sense.

Sen et al., in their handling of nested probabilistic operators, confuse the p-value with the probability of accepting a false hypothesis (generally referred to as the type I or type II error of a sampling plan). The p-value is *not* a bound on the probability of a certain test procedure accepting a false hypothesis. In fact, the test that both they and we use does not provide a useful bound on the probability of accepting a false hypothesis. Their analysis relies heavily on the ability to bound the probability of accepting a false hypothesis, and we have presented a way to provide such bounds by introducing indifference regions for nested probabilistic operators.

In addition, Sen et al. are vague regarding the assumptions needed for their approach to produce reliable answers. The fact that they treat any portion of a trajectory starting in s, regardless of the portion preceding s, as a sample from the same distribution, hides a rather strong assumption regarding the dynamics of their "black-box" systems. As we have pointed out, this is not a valid assumption unless we know that the system is a Markov process. It also appears as if they consider only truncated trajectories over which they can fully verify a path formula, and this can introduce a bias that very well may invalidate the conclusion reached regarding the truth value of a probabilistic formula. We have made this clear in our exposition, and we have presented a sound procedure for handling the fact that the value of a path formula may not be determined over all the truncated trajectories.

Finally, the empirical analysis offered by Sen et al. gives the reader the impression that a certain p-value can be guaranteed for a verification result simply by increasing the sample size. This violates the premise of a “black-box” system stated by the authors themselves earlier in their paper, namely that trajectories cannot be generated on demand. More important, though, is the fact that a certain p-value can *never* be guaranteed. The p-value is not a property of a test, but simply a function of a specific set of observations. If we are unlucky, we may make observations that give us a large p-value even in cases when this is unlikely. It is therefore misleading to say that an algorithm for “black-box” verification is “faster” than the statistical model checking algorithm described in Chapter 5, as the latter algorithm is designed to realize certain *a priori* performance characteristics. The empirical results of Sen et al. cannot, in fact, be replicated reliably because there is no fixed procedure by which one can determine the sample size required to achieve a certain p-value. Their results give the false impression that their procedure is sequential, i.e. that the sample size automatically adjusts to the difficulty of attaining a certain p-value, when in reality they selected the reported sample sizes *manually* based on prior empirical testing (K. Sen, personal communication, May 20, 2004).

Part II

Planning

Chapter 8

Goal Directed Planning

We now turn to the problem of planning for stochastic systems with asynchronous events and actions. In this chapter, we consider goal directed planning problems. We propose the use of UTSL as a formalism for specifying plan objectives, and we present a general planning framework based on the Generate, Test and Debug (GTD) paradigm (Simmons 1988). The goal is to generate a stationary policy, i.e. a mapping from states to actions, that satisfies a UTSL goal condition. To handle the complexity of asynchronous events with general delay distributions, we resort to statistical techniques. We use the statistical approach for UTSL model checking, presented in Chapter 5, to verify policies. During the verification phase, sample trajectories are generated, which can then be analyzed to find reasons for why a policy fails to satisfy the goal condition. The result of this analysis is used to guide policy debugging.

We use a deterministic temporal planner to help generate the policies. A probabilistic planning problem is transformed into a deterministic problem by making every possible outcome of events and actions available to the planning system. The solution is a deterministic plan, from which a policy is generated through decision tree learning. This policy is typically overly optimistic, and the sample trajectories obtained during policy verification are used to restrict the subsequent choices that the planning system can make.

8.1 Planning Framework

We present a general framework for goal directed probabilistic planning with asynchronous events, based on the Generate, Test and Debug (GTD) paradigm proposed by Simmons (1988). The domain model is a

```
FIND-POLICY(M, s0, φ)
  π0 ⇐ GENERATE-INITIAL-POLICY(M, s0, φ)
  if TEST-POLICY(M, s0, φ, π0) then
    return π0
  else
    π ⇐ π0
    loop    ▷ return π on interrupt
      π′ ⇐ DEBUG-POLICY(M, s0, φ, π)
      if TEST-POLICY(M, s0, φ, π′) then
        return π′
      π ⇐ BETTER-POLICY(M, s0, φ, π, π′)
```

Algorithm 8.1: Generic planning algorithm for probabilistic planning based on the GTD paradigm.

continuous-time stochastic discrete event system, and policies are generated to satisfy properties specified as UTSL formulae (Chapter 4). The approach resembles that of Drummond and Bresina (1990) for probabilistic planning in discrete-time domains. Both approaches use temporal logic to express goal conditions, and goal conformance is achieved through incremental plan modification.

At the core of the framework is a generic hill-climbing procedure, FIND-POLICY, shown as Algorithm 8.1. The input to the procedure is a model $\mathcal{M}$ of a stochastic discrete event system, an initial state s_0, and a UTSL goal condition ϕ. The result is a policy π such that the stochastic process $\mathcal{M}[\pi]$ (i.e. $\mathcal{M}$ controlled by π) satisfies ϕ when execution starts in a state s_0.

The procedure GENERATE-INITIAL-POLICY returns a seed policy for the policy search algorithm. In Section 8.2, we describe in detail how to implement this procedure using an existing deterministic temporal planner. TEST-POLICY returns true if the current policy satisfies the goal condition, and returns false if the goal condition is violated. This amounts to solving the UTSL model checking problem $\langle \mathcal{M}[\pi], s_0, \phi \rangle$, which can be done using existing numerical solution methods or the statistical solution technique presented in Chapter 5. DEBUG-POLICY is responsible for debugging the current policy and returning a new policy. If the new policy still does not satisfy the goal condition, then we retain the better of the two policies, as determined by BETTER-POLICY, and continue until a satisfactory policy is found or the search is interrupted.

In the work presented here, it is essential that TEST-POLICY uses a statistical approach, because our implementation of DEBUG-POLICY relies on the sample trajectories that are produced during policy verification for its failure analysis. DEBUG-POLICY analyses the sample trajectories to find reasons why the goal

Goal description	UTSL Formula
reach office with probability at least 0.9	$\mathcal{P}_{\geq 0.9}[\Diamond\ \mathit{office}]$
reach office within 17 time units with probability at least 0.9	$\mathcal{P}_{\geq 0.9}\big[\Diamond^{[0,17]}\ \mathit{office}\big]$
reach office within 17 time units with probability at least 0.9 while not spilling coffee	$\mathcal{P}_{\geq 0.9}\big[\neg \mathit{coffee\text{-}spilled}\ \mathcal{U}^{[0,17]}\ \mathit{office}\big]$
reach office within 17 time units with probability at least 0.9 while maintaining at least a 0.5 probability of eventually recharging	$\mathcal{P}_{\geq 0.9}\big[\mathcal{P}_{\geq 0.5}[\Diamond\ \mathit{recharging}]\ \mathcal{U}^{[0,17]}\ \mathit{office}\big]$
remain stable for at least 8.2 time units with probability at least 0.7	$\mathcal{P}_{\geq 0.7}\big[\Box^{[0,8.2]}\ \mathit{stable}\big]$

Table 8.1: Examples of goals expressible as UTSL formulae.

condition is violated, attempts to debug the current policy based on the outcome of the failure analysis, and returns a new policy.

The model $\mathcal{M}$ is assumed to be a stochastic discrete event system with state space S and event set E. We associate an enabling condition, ϕ_e, with each event $e \in E$. In state s, events $E_s = \{e \in E \mid s \models \phi_s\}$ are enabled and race to trigger. The event that triggers first causes a state transition to occur. For most of this chapter, we will assume that the model is a GSMP (Section 2.3.3). Algorithm 8.1 does not rely on this assumption—it can be made to work for arbitrary stochastic discrete event systems, but we will exploit the probability structure imposed by a GSMP model to guide the generation of an initial policy and the subsequent debugging of unsatisfactory policies.

A decision dimension is added to the domain model by identifying a set $A \subset E$ of actions (controllable events) that can be disabled at will. A policy π is used to determine which actions should be enabled in any given situation. We restrict our attention to stationary policies, which are mappings from states to actions. A model $\mathcal{M}$ controlled by a stationary policy π is a stochastic discrete event system $\mathcal{M}[\pi]$ with events $\big\{e \in E \mid \big(s \models \phi_e\big) \wedge \big(e \in A \rightarrow e = \pi(s)\big)\big\}$ enabled in state s. We can choose to be idle (i.e. have no action enabled) in a state. A special action, a_ϵ, is used to represent idleness and has an enabling condition that is always true.

We use a subset of UTSL to express plan objectives, consisting of formulae of the form $\mathcal{P}_{\bowtie\theta}\big[\Phi\ \mathcal{U}^I\ \Psi\big]$ and formulae that can be transformed to this form, such as $\mathcal{P}_{\bowtie\theta}\big[\Diamond^I\ \Phi\big]$. A wide variety of goals can be expressed with this subset of UTSL. Table 8.1 shows examples of achievement goals, goals with safety constraints on execution paths, and maintenance/prevention goals. We limit our attention to goal formulae with finite time bounds.

8.2 Initial Policy Generation

Given a planning problem $\langle \mathcal{M}, s_0, \phi \rangle$, we want to find a stationary policy $\pi : S \rightarrow E_{\mathrm{a}}$ such that $\mathcal{M}[\pi], s_0 \models \phi$. Algorithm 8.1 outlines a procedure for finding such a policy by means of local search. The efficiency of the procedure will depend on the quality of the initial policy returned by GENERATE-INITIAL-POLICY. A quick solution would be to simply return the null-policy mapping every state to the idle action a_ϵ, but this ignores the goal condition of the planning problem. If we can make a more informed choice for an initial policy, it is likely to have fewer bugs than the null-policy, thus requiring fewer repairs.

We present an implementation of GENERATE-INITIAL-POLICY that relaxes the original planning problem by ignoring uncertainty and solves the resulting deterministic planning problem using an existing temporal planner. Our implementation uses a slightly modified version of VHPOP (Younes and Simmons 2003), a heuristic partial order causal link (POCL) planner with support for PDDL2.1 durative actions (Fox and Long 2003).

8.2.1 Conversion to Deterministic Planning Problem

We assume a GSMP model. This means that a distribution G_e is associated with each event e governing the time from when e becomes enabled until it triggers, provided e remains continuously enabled during that time period. At the triggering of event e in state s, the next state is determined by a probability distribution $p_e(\cdot; s)$. If we have a factored representation of the state space, with Boolean state variables V, then the distribution $p_e(\cdot; s)$ can be represented implicitly by an effect formula eff_e using the formalism presented by Rintanen (2003). Effects are recursively defined as follows:

1. $\top$ is the null-effect.
2. b and $\neg b$ are effects if $b \in V$ is a Boolean state variable.
3. $\mathit{eff}_1 \wedge \cdots \wedge \mathit{eff}_n$ is an effect if eff_1 through eff_n are effects.
4. $c \rhd \mathit{eff}$ is an effect if c is a formula over V and eff is an effect.
5. $p_1 \mathit{eff}_1 | \ldots | p_n \mathit{eff}_n$ is an effect if eff_1 through eff_n are effects, $p_i \geq 0$ for all $i \in \{1, \ldots, n\}$, and $\sum_{i=1}^{n} p_i = 1$.

The language PPDDL+, described in Appendix B, uses this representation. Younes and Littman (2004) describe how to compute an explicit representation of $p_e(\cdot; s)$ from an effect formula.

We relax a temporal probabilistic planning problem by treating all events of a model equally, ignoring the fact that some events are not controllable. In other words, all events are considered to be actions that the deterministic planner can choose to include in a plan. We eliminate probabilistic effects by splitting events with probabilistic effects into multiple events with deterministic effects. Each new event has the same enabling condition as the original event and an effect representing a separate outcome of the original event's probabilistic effect. An event with probabilistic effect $p_1\,\mathit{eff}_1|\ldots|p_n\,\mathit{eff}_n$ is split into n events, the ith event having deterministic effect eff_i.[1] Furthermore, instead of a probability distribution over possible event durations, we associate an interval with each event representing the possible durations for the event. This interval is simply the support of the probability distribution for the event delay. The deterministic temporal planner is permitted to select any duration within the given interval for an event that is part of a plan. In the next section, when we discuss policy debugging, we consider ways of constraining the choice of action and event durations based on information gathered during the verification phase.

With these transformations, each event can be represented as one or more PDDL2.1 durative actions with interval constraints on the duration, with the enabling condition of the event as a condition that must hold over the entire duration of the action, and with the effect associated with the end of the durative action. Figure 8.1 shows a stochastic event with delay distribution $U(0, 10)$ and a probabilistic effect with two outcomes, and the two durative actions with deterministic effects that are used to represent the stochastic event. The purpose of the transformation is to make every possible outcome of a stochastic event available to the deterministic planner.

A UTSL goal condition of the form $\mathcal{P}_{\geq p}\big[\Phi \ \mathcal{U}^{[\tau,\tau']} \ \Psi\big]$ is converted into a goal for the deterministic planning problem as follows. We make Ψ a goal condition that must become true some time between τ and τ' time units after the start of the plan, while Φ becomes an invariant condition that must hold until Ψ is satisfied. We can represent this goal in the temporal POCL framework as a durative action with no effects, with an invariant condition Φ that must hold over the duration of the action, and a condition Ψ associated

[1] Nested probabilistic effects may require further splitting. Any effect formula can be transformed to the form $p_1\,\mathit{eff}_1|\ldots|p_n\,\mathit{eff}_n$, where eff_i is a deterministic effect, although this may result in an exponential increase in the size of the effect formula (Rintanen 2003).

```
(:delayed-event crash
        :delay (uniform 0 10)
        :condition (up)
        :effect (probabilistic 0.4 (down) 0.6 (broken)))

(:durative-action crash1
        :duration (and (>= ?duration 0) (<= ?duration 10))
        :condition (and (at start (up)) (over all (up)) (at end (up)))
        :effect (at end (down)))
(:durative-action crash2
        :duration (and (>= ?duration 0) (<= ?duration 10))
        :condition (and (at start (up)) (over all (up)) (at end (up)))
        :effect (at end (broken)))
```

Figure 8.1: A stochastic event (top) and two durative deterministic actions (bottom) representing the stochastic event.

with the end of the action. We add the temporal constraints that the start of the goal action must be scheduled at time 0 and that the end of the action must be scheduled in the interval $[\tau, \tau']$. VHPOP records all such temporal constraints in a *simple temporal network* (Dechter et al. 1991) allowing for efficient temporal inference during planning.

For UTSL goals of the form $\mathcal{P}_{\leq p}\big[\Phi\ \mathcal{U}^{[\tau,\tau']}\ \Psi\big]$, we instead want to find plans representing executions *not* satisfying the path formula $\Phi\ \mathcal{U}^{[\tau,\tau']}\ \Psi$. We then use $\neg\Psi$ as an invariant condition that must hold in the interval $[\tau, \tau']$. This can be represented by a durative action scheduled to start at time τ and end at time τ' with invariant condition $\neg\Psi$ and no effect. Note that it is not necessary to achieve $\neg\Phi$ in order for $\Phi\ \mathcal{U}^{[\tau,\tau']}\ \Psi$ to be false, so we do not include Φ in the deterministic planning problem. This means that an empty plan will satisfy the goal condition, unless τ is zero and Ψ holds in the initial state in which case the problem lacks solution. We therefore return the null-policy as an initial policy for such goals.

There are a few additional constraints that we enforce in the modified version of VHPOP. The first is that we do not allow concurrent actions. This is due to the restriction on policies being mappings from states to single actions. The restriction is not severe, however, since an "action" with extended delay can be modeled as a controllable event with short delay to start the action and an exogenous event to end the action, allowing for additional actions to be executed before the temporally extended action completes. For example, a "drive" action with extended duration can be represented by a "start" action and an "arrive" event. The second constraint is that separate instances of the same exogenous event cannot overlap in time.

For example, if one instance of the "crash" event is enabled at time τ and scheduled to trigger at time τ', then no other instances of "crash" can be scheduled to be enabled or trigger in the interval $[\tau, \tau']$. This constraint follows from the GSMP domain model. Both constraints are of the same nature and are represented in the planner as a new flaw type, associated with two events e_1 and e_2, that can be resolved in ways analogous to promotion and demotion for regular POCL threat resolution: either the end of e_1 must come before the start of e_2, or the start of e_1 must come after the end of e_2.

The state of a GSMP can change only at the triggering of an event. At this point, other events can be enabled. It is not possible, however, that an event becomes enabled between state transitions. A plan is adjusted, before it is returned by GENERATE-INITIAL-POLICY, to ensure that events are scheduled to become enabled at the triggering of some other event, and not at an arbitrary point in time. A plan now represents an execution of actions and exogenous events satisfying the path formula $\Phi \; \mathcal{U}^{[\tau,\tau']} \; \Psi$, possibly ignoring the adverse effects of other exogenous events, which is left for the debugging phase to discover.

8.2.2 From Plan to Policy

A plan returned by VHPOP is a set of triples $\langle t_i, e_i, d_i \rangle$, where e_i is an event, t_i is the time that e_i is scheduled to become enabled, and d_i is the delay of e_i (i.e. e_i is scheduled to trigger at time $t_i + d_i$). Given a plan, we now want to generate a policy. We represent a policy using a *decision tree* (cf. Boutilier et al. 1995), and generate it by converting a plan into a set of training examples composed of state-action pairs $\langle s_i, e_i \rangle$, $s_i \in S$ and $e_i \in A \cup \{a_\epsilon\}$, and then generating a decision tree from these training examples. The training examples are obtained by serializing the plan returned by VHPOP and executing the sequence of events, starting in the initial state. A decision tree policy can be compiled into a set of *test-action pairs*, the policy representation used by CIRCA (Musliner et al. 1995), to facilitate efficient and predictable execution behavior.

We serialize a plan by sorting the events in ascending order based on their trigger time, breaking ties nondeterministically. The first event to trigger, call it e_0, is applied to the initial state s_0, resulting in a state s_1. If e_0 is an action, then this gives rise to a training example $\langle s_0, e_0 \rangle$. Otherwise, the first event gives rise to the training example $\langle s_0, a_\epsilon \rangle$, signifying that we are waiting for something beyond our control to happen in state s_0. We continue to generate training examples in this fashion until there are no unprocessed events left in the plan. Given a set of training examples for the initial plan, we use regular decision tree induction (Quinlan 1986) to generate an initial policy. The policy will assign actions even to states that are not included in the training set. It is left to the debugging phase to identify overgeneralization.

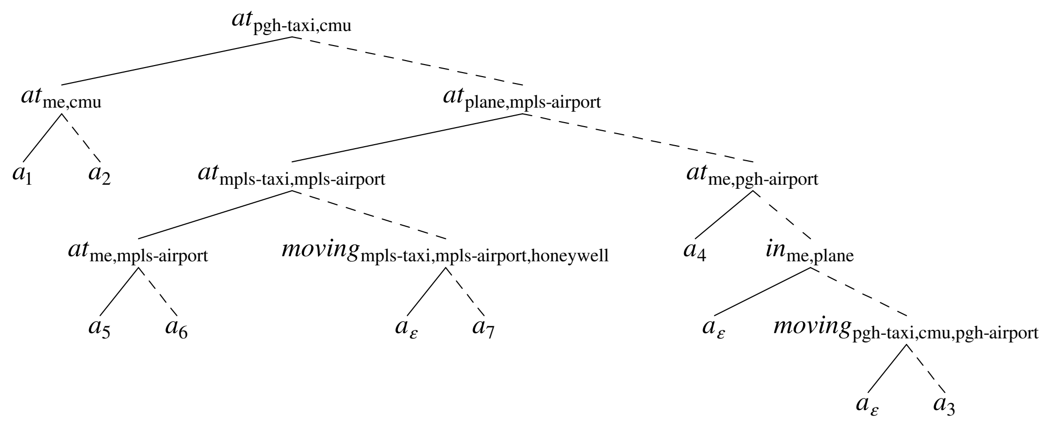
To illustrate the process of generating an initial policy, consider the planning problem described by Younes et al. (2003), which is a continuous-time variation of a problem developed by Blythe (1994). In this problem, the goal is to have a person transport a package from CMU in Pittsburgh to Honeywell in Minneapolis in at most 300 time units with probability at least 0.9, without losing it on the way. In UTSL, this goal can be expressed as $\mathcal{P}_{\geq 0.9}\left[\neg lost_{\mathrm{pkg}}\ \mathcal{U}^{[0,300]}\ at_{\mathrm{me,honeywell}} \wedge carrying_{\mathrm{me,pkg}}\right]$. The package can be transported between the two cities by airplane and between two locations within the same city by taxi. There is one taxi in each city. The Pittsburgh taxi is initially at CMU, while the Minneapolis taxi is at the airport. There is one airplane available, and it is initially at the Pittsburgh airport. The airplane can get filled if we do not have a reservation, preventing us to board it when arriving at the Pittsburgh airport. A reservation can be made from CMU. Taxis located at airports serve other customers periodically, which means that we may have to wait for a taxi when we arrive at the Minneapolis airport. If we stay for too long at an airport, the package can get lost, although this can be prevented by putting the package in storage. The departure of the airplane from an airport is controlled by an exogenous event, which means that we can miss the departure if it takes too long to get to the airport.

Figure 8.2(a) shows the plan generated by the deterministic temporal planner. The plan schedules two events to become enabled at time zero, one being the action to enter a taxi at CMU, and the other being the exogenous event causing the plane to depart from Pittsburgh to Minneapolis (actions are identified by an entry in the second column of the table in Figure 8.2(a)). The "enter-taxi" action is scheduled to trigger first, resulting in a training example mapping the initial state to this action. The next state is mapped to the first "depart-taxi" action, while the state following the triggering of that action is mapped to the idle action. This is because the next event ("arrive-taxi") is not an action. Eight additional training examples can be extracted from the plan, and the decision tree representation of the policy learned from the eleven training examples is shown in Figure 8.2(b). This policy, for example, maps all states satisfying $at_{\text{pgh-taxi,cmu}} \wedge at_{\mathrm{me,cmu}}$ to the action labeled a_1 (the first "enter-taxi" action in the plan), while states where $at_{\text{pgh-taxi,cmu}}$, $at_{\text{plane,mpls-airport}}$, and $at_{\text{me,pgh-airport}}$ are all false and $in_{\mathrm{me,plane}}$ is true are mapped to the idle action a_ϵ.

Additional training examples can be obtained from plans with multiple events scheduled to trigger at the same time by considering different trigger orderings of the simultaneous events. If two events e_1 and e_2 are

t_i:$e_i[d_i]$	act.
0:(enter-taxi me pgh-taxi cmu)[1]	a_1
0:(depart-plane plane pgh-airport mpls-airport)[60]	
1:(depart-taxi me pgh-taxi cmu pgh-airport)[1]	a_2
2:(arrive-taxi pgh-taxi cmu pgh-airport)[20]	
22:(leave-taxi me pgh-taxi pgh-airport)[1]	a_3
23:(check-in me plane pgh-airport)[1]	a_4
60:(arrive-plane plane pgh-airport mpls-airport)[90]	
150:(enter-taxi me mpls-taxi mpls-airport)[1]	a_5
151:(depart-taxi me mpls-taxi mpls-airport honeywell)[1]	a_6
152:(arrive-taxi mpls-taxi mpls-airport honeywell)[20]	
172:(leave-taxi me mpls-taxi honeywell)[1]	a_7

(a) Plan for simplified deterministic planning problem.



(b) Policy generated from plan in (a).

Figure 8.2: (a) Initial plan and (b) policy for transportation problem. Leaves in the decision tree are labeled by actions, with labels taken from the table in (a). To find the action selected by the policy for a state s, start at the root of the decision tree. Traverse the tree until a leaf node is reached by following the left branch of a decision node if s satisfies the test at the node and following the right branch otherwise.

DEBUG-POLICY$(\mathcal{M}, s_0, \phi, \pi)$
  $S_\perp \Leftarrow$ set of states occurring in $\vec{\sigma}_\perp$
  $s \Leftarrow$ some state in $S_\perp$
  $a \Leftarrow$ some action in $\{a \in A \cup \{a_\epsilon\} \mid s \models \phi_a\} \setminus \{\pi(s)\}$
  $\pi' \Leftarrow \pi$, but with the mapping of s to a
  return π'

Algorithm 8.2: Generic nondeterministic procedure for debugging a policy.

both scheduled to trigger at time t, we would get one set of training example by applying e_1 before e_2, and a second set by applying e_2 before e_1. This can result in different training examples if one of the events is an action.

8.3 Policy Debugging

During verification of a policy π for a planning problem $\langle \mathcal{M}, s_0, \phi \rangle$, a set of sample trajectories $\vec{\sigma} = \{\sigma_1, \cdots, \sigma_n\}$ is generated for the stochastic process $\mathcal{M}[\pi]$ with initial state s_0. If the policy π does not satisfy the goal condition ϕ, then these sample trajectories can help us understand the "bugs" of π and provide us with valuable information on how to debug the policy.

Let $\vec{\sigma}_\perp$ denote the set of trajectories over which φ is verified not to hold. This set of sample trajectories provides information on how a policy can fail to satisfy the specified goal condition. We can use this information to guide policy debugging, without relying on model specific knowledge.

To debug a policy for goal condition $\mathcal{P}_{\geq \theta}[\varphi]$, we must lower the probability measure of the set of trajectories not satisfying φ. Each member $\sigma_i \in \vec{\sigma}_\perp$ is a trajectory prefix $\{\langle s_0, t_0 \rangle, \ldots \langle s_k, t_k \rangle\}$ providing evidence on how a policy can fail to achieve the goal condition. We could, conceivably, improve a policy by modifying it so that the sequence of states appearing along a sample trajectory $\sigma_i \in \vec{\sigma}_\perp$ is interrupted. Algorithm 8.2 shows a generic procedure for debugging a policy based on this simple principle. A state is nondeterministically selected from the set of states that occur along some failure trajectory and an alternative action is assigned to that state, resulting in a modified policy.

The sample trajectories can help us focus the debug effort on the relevant parts of the state space, in particular if failure occurs early along a trajectory. There is little, however, to guide the state and action choice in the model independent approach. We next present model dependent techniques for analyzing sample trajectories that can lead to a more efficient implementation of the DEBUG-POLICY procedure. The result of the analysis is a set of ranked *failure scenarios*. A failure scenario can be fed to the deterministic temporal planner, which will try to generate a plan that takes the failure scenario into account. The resulting plan, if one exists, can be used to debug the current policy.

8.3.1 Analysis of Sample Trajectories

Policy verification generates a set of trajectory prefixes $\vec{\sigma} = \{\sigma_1, \ldots, \sigma_n\}$, with each trajectory prefix being of the form

$$\sigma_i = \{\langle s_{i0}, t_{i0}\rangle, e_{i0}, \ldots, \langle s_{i,k_i-1}, t_{i,k_i-1}\rangle, e_{i,k_i-1}, \langle s_{ik_i}, t_{ik_i}\rangle\} \enspace .$$

This form differs slightly from our previous representation of sample trajectories in that it includes the triggering events. Knowing which events cause state transitions, and not only the time at which the transitions occur, is essential in our analysis. The goal of the analysis is to produce a set of failure scenarios that summarizes the information in the sample trajectories. A failure scenario is a sequence $\langle e_1@t_1, \ldots, e_n@t_n\rangle$ of events and trigger times, and is constructed with a specific event e_k, $1 \leq k \leq n$, in mind. A failure scenario for e_k is meant to represent an average trajectory that does not satisfy the goal condition while including a state transition caused by e_k. Each failure scenario is assigned a score, with a lower score indicating higher severity.

We start the construction of failure scenarios by computing a value, relative to a UTSL goal formula $\mathcal{P}_{\geq\theta}\big[\Phi\ \mathcal{U}^{[\tau,\tau']}\ \Psi\big]$, for each state occurring along a sample trajectory. The value of a state is between -1 and 1, and signifies the closeness to success or failure, ignoring timing information and counting only the number of transitions. A large positive value indicates closeness to success, while a large negative value indicates closeness to failure. State values are computed by constructing a discrete-time Markov reward process representing an abstract view of the sample trajectories (cf. Riley and Veloso 2004). The state space for this Markov reward process is the set of states that occur along some sample trajectory. The transition probabilities $p(s'; s)$ are defined as the number of times s' is immediately followed by s along the sample trajectories divided by the total number of occurrences of s. Let k_s be the number of trajectory prefixes that end in state s and satisfy the path formula $\Phi\ \mathcal{U}^{[\tau,\tau']}\ \Psi$, and let l_s be the number of trajectory prefixes that end in s and do not satisfy the path formula. Then, the *immediate reward* associated with state s is

$(k_s - l_s)/(k_s + l_s)$, or 0 if no trajectory ends in s.[2] The values of states are computed using the recurrence

$$V(s) = \gamma \sum_{s' \in S} p(s'; s) V(s') \ ,$$

where $\gamma < 1$ is a discount factor. The discount factor permits us to control the influence a success or failure has on the value of states at some distance from the point along a trajectory at which success or failure is determined to occur. State values can be computed iteratively, with the initial value of a state being equal its immediate reward.

The next step is to assign a value to each event that occurs along some sample trajectory. Each triple $s \xrightarrow{e} s'$, meaning that e causes a transition from s to s', is given the value $V(s') - V(s)$, which can be seen as the value contribution of e. The value $V(e)$ of an event e is the sum of the values of all triples that e is part of. This way, an event that occurs often but early on the path to failure can have a lower value than an event that leads directly to failure but only rarely. For later use, the mean μ_e and standard deviation σ_e over triples involving e is also computed. The event with the largest negative value can be thought of as the "bug" contributing the most to failure, and we want to plan to avoid this event or to prevent it from having negative effects. The event, by itself, may not be sufficient to understand why failure occurs. A failure scenario provides the context in which the event leads to failure.

We construct a failure scenario for each event e by combining the information from all failure trajectories σ_i containing a triple $s \xrightarrow{e} s'$ such that $V(s') - V(s) < \mu_e + \sigma_e$. The reason for the cutoff is to not include information from failure trajectories where an event contributes to failure significantly less than on average so that the aggregate information is representative for the "bug" being considered. For example, we fail to deliver the package to Honeywell in Minneapolis if the airplane is filled before we have a chance to board it. However, every occurrence of a "fill-plane" event along a failure trajectory does not represent the same "bug". If the airplane is filled while we are on our way to the Pittsburgh airport, but we also arrive at the airport after the airplane has departed, the "fill-plane" event would be less responsible for failure than if we had arrived at the airport in time for departure.

A failure scenario is constructed from a set of trajectories by averaging the trigger times of events. Figure 8.3 gives an example of how two failure trajectories are combined into a single failure scenario. Event e_1 occurs twice along both failure trajectories and therefore occurs twice in the failure scenario. The

[2] For a goal formula $\mathcal{P}_{\leq\,\theta}[\varphi]$, the immediate rewards are negated.

Trajectory 1	Trajectory 2	Failure Scenario
e_1 @ 1.2	e_1 @ 1.6	e_1 @ 1.4
e_2 @ 3.0	e_2 @ 3.2	e_2 @ 3.1
e_1 @ 4.5	e_3 @ 4.4	e_1 @ 4.5
e_3 @ 4.8	e_1 @ 4.5	e_3 @ 4.6
e_4 @ 6.8	e_5 @ 6.4	e_5 @ 6.7
e_5 @ 7.0	-	-

Figure 8.3: Example of failure scenario construction from two failure trajectories.

e_i @ t_i	Label
(enter-taxi me pgh-taxi cmu) @ 0.909091	a_1
(depart-taxi me pgh-taxi cmu pgh-airport) @ 1.81818	a_2
(fill-plane plane pgh-airport) @ 13.284	e_3
(arrive-taxi pgh-taxi cmu pgh-airport) @ 30.0722	e_4
(leave-taxi me pgh-taxi pgh-airport) @ 30.9813	a_5
(lose-package me pkg pgh-airport) @ 44.0285	e_6

Figure 8.4: Failure scenario for the policy in Figure 8.2(b) associated with the "fill-plane" event.

trigger time for the ith occurrence of e_1 in the failure scenario is the average of the trigger times of the ith occurrences of e_1 in the two trajectories. Event e_4 only appears along the first trajectory and is thus excluded from the scenario (it is assumed that e_4 has trigger time ∞ in the second trajectory, which makes the average trigger time ∞ as well). Figure 8.4 shows an actual failure scenario for the transportation problem.

8.3.2 Planning with Failure Scenarios

We select the failure scenario for the event with the lowest value and try to generate a plan for the selected scenario that achieves the goal. If this fails, we try planning for the next worst failure scenario, and continue in this manner until we find a promising repair, or run out of failure scenarios.

We plan to neutralize a failure scenario by incorporating the events and timing information of the scenario into the planning problem that is then passed to the temporal deterministic planner. Given a failure scenario $\langle e_1@t_1, \ldots, e_k@t_k, \ldots, e_n@t_n \rangle$ associated with the event e_k, we generate a sequence of states $s_0, \ldots, s_n$, where s_0 is the initial state of the original planning problem and s_i for $i > 0$ is the state obtained by applying e_i to state s_{i-1}. We can plan to avoid the bad event e_k by generating a planning problem with initial state s_i for $i < k$. By choosing i closer to k, we can potentially avoid planning for situations that the current policy already handles well. By choosing i closer to 0, we allow the planner more time to neutralize

e_k. Our implementation iterates over the possible start states from $i = k - 1$ to $i = 0$. If a solution is found for some i, then we do not investigate other possible initial states. For each planning problem generated, we limit the number of search nodes explored by VHPOP. This is necessary because VHPOP takes too long time to recognize that a problem lacks solution, but often finds a solution quickly if one exists. In case the search limit is reached, we attempt to plan given an earlier initial state, or try to plan for the next worst failure scenario if we already are at $i = 0$.

Given an initial state s_i, the events following s_i in the failure scenario are incorporated into the planning problem in the form of a set of *event dependency trees* $\mathcal{T}_i$ and a set of *untriggered* events $\mathcal{U}_i$. The purpose of these two sets is to force the deterministic planner to schedule events in a way consistent with the failure scenario. Each node in an event dependency tree stores an event and a trigger time for the event relative to the parent node (or relative to the initial state for root nodes). The children of a node for an event e represent events that depend on the triggering of e to become enabled. If the deterministic planner schedules the event e, then the events that depend on e should be scheduled to follow e. The set $\mathcal{U}_i$ represents events that are enabled in all states s_j but differ from all events e_j for $j \geq i$, and these events should not be allowed to trigger between time 0 and t_n in the deterministic planning problem.

We define the sets $\mathcal{T}_i$ and $\mathcal{U}_i$ for state s_i recursively. The base case is $\mathcal{T}_n = \emptyset$, with $\mathcal{U}_n$ containing all events enabled in s_n (a failure scenario imposes no scheduling constraints after the last event of the scenario). For $i < n$, let $\delta = t_{i+1} - t_i$ (or simply t_1 for $i = 0$) and construct a tree T_i consisting of a single node with event e_{i+1} and trigger time δ. For each tree $T \in \mathcal{T}_{i+1}$:

- if the event at the root of T is an action, then add T to $\mathcal{T}_i$ (there is no reason to force an action to follow the triggering of an event, because actions are truly under the control of the planner).
- if the event at the root of T is enabled in s_i, then add δ to the trigger time of the root node and add the resulting tree to $\mathcal{T}_i$.
- if the event at the root of T is disabled in s_i, then add T to the children of T_i (if the root event of T is disabled in s_i, then it is enabled by e_{i+1} according to the failure scenario).

Let $\mathcal{U}$ be the set of events $e \in \mathcal{U}_{i+1}$ not enabled in s_i. Then $\mathcal{U}_i = \mathcal{U}_{i+1} \setminus \big(\mathcal{U} \cup \{e_{i+1}\}\big)$. Finally, add T_i to $\mathcal{T}_i$.

For the scenario shown in Figure 8.4 and the state right before the “fill-plane” event ($i = 2$), there are three event trees: one with e_4@28.254 as the sole node, one with e_3@11.4658 as the sole node, and a final tree with a_5 at the root and e_6@13.0472 as a child node. The set $\mathcal{U}_2$ contains the following two events:

(depart-plane plane pgh-airport mpls-airport)

(move-taxi mpls-taxi mpls-airport)

This means that if we start planning from state s_2, we are not allowed to schedule either of these two events until after the trigger time for the last event in the failure scenario.

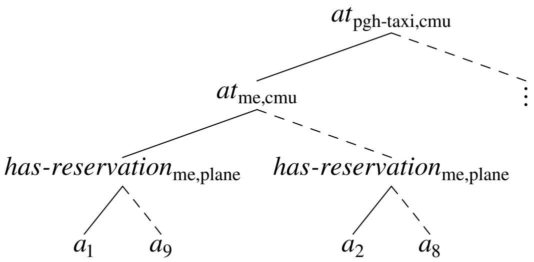
We incorporate the event trees in $\mathcal{T}_i$ that have an exogenous event at the root into the deterministic planning problem by forcing all the events in these trees to be part of the plan. Events at root nodes are scheduled to become enabled at time 0 and to trigger at the time stored at the node, and events at non-root nodes are scheduled to become enabled at the time the parent event triggers and scheduled to trigger t time units after the parent event triggers (t being the time stored at the node). The deterministic planner is allowed to disable the effects of a forced event by disabling its enabling condition. This can easily be handled in a POCL framework by treating the enabling condition as an effect condition that can be disabled by means of *confrontation* (Weld 1994). The sets $\mathcal{U}_i$ impose further scheduling constraints for the deterministic planner.

Once a plan is found for a failure scenario, we extract a set of training examples from the plan as described in Section 8.2. We update the current policy by incorporating the additional training examples into the decision tree using incremental decision tree induction (Utgoff et al. 1997). This requires that we store the old training examples in the leaf nodes of the decision tree, and some additional information in the decision nodes, but we avoid having to generate the entire decision tree from scratch. We adapt the algorithm of Utgoff et al. to our particular situation by always giving precedence to new training examples over old ones in case of inconsistencies, and by restructuring the decision tree only after incorporating all new training examples (the latter is done for efficiency and does not change the outcome).

Figure 8.5(a) shows a plan for the failure scenario in Figure 8.4, with the state after the "enter-taxi" action as the initial state for the planning problem. Note, in particular, the "fill-plane" event, which the deterministic planner has been forced to schedule at time 12.3749. The planner uses the "make-reservation" action to counter the adverse effects of the "fill-plane" event. The policy after incorporating the training examples generated from the plan is shown in Figure 8.5(b). The entire right subtree for the repaired policy is the same as for the initial policy, so it does not have to be regenerated.

t_i:$e_i[d_i]$	act.
0:(leave-taxi me pgh-taxi cmu)[1]	a_8
0:(depart-plane plane pgh-airport mpls-airport)[60]	
0:(fill-plane plane pgh-airport)[12.3749]	
1:(make-reservation me plane cmu)[1]	a_9
2:(enter-taxi me pgh-taxi cmu)[1]	a_1
3:(depart-taxi me pgh-taxi cmu pgh-airport)[1]	a_2
4:(arrive-taxi pgh-taxi cmu pgh-airport)[20]	
24:(leave-taxi me pgh-taxi pgh-airport)[1]	a_3
25:(check-in me plane pgh-airport)[1]	a_4
60:(arrive-plane plane pgh-airport mpls-airport)[90]	
150:(enter-taxi me mpls-taxi mpls-airport)[1]	a_5
151:(depart-taxi me mpls-taxi mpls-airport honeywell)[1]	a_6
152:(arrive-taxi mpls-taxi mpls-airport honeywell)[20]	
172:(leave-taxi me mpls-taxi honeywell)[1]	a_7

(a) Plan for failure scenario.



(b) Repaired policy.

Figure 8.5: (a) Plan for failure scenario in Figure 8.4 using the second state as initial state, and (b) the policy after incorporating the training examples from the plan in (a). If the taxi is at CMU but we are not, then it is assumed that we are in the taxi. In that case, we leave the taxi (a_8) if we do not have a reservation. The right subtree of the root node is identical to that of the initial policy in Figure 8.2(b), and is only indicated by three vertical dots.

8.4 Statistical Policy Comparison

The procedure BETTER-POLICY is supposed to compare the policies π and π', returning the better of the two. Given a UTSL goal condition $\mathcal{P}_{\geq\theta}[\varphi]$, let p be the probability measure of the set of trajectories that satisfy φ for model $\mathcal{M}[\pi]$ and let p' be the probability measure of the set of trajectories that satisfy φ for model $\mathcal{M}[\pi']$. We can use a statistical approach to implementing BETTER-POLICY such that it returns π with high probability if p is significantly greater than p', π' with high probability if p is significantly less than p', and either of the two policies with roughly equal probability if p is close to p'.

The problem of comparing two policies can be posed as a hypothesis testing problem. We want to test the hypothesis $H : p \geq p'$ against the alternative hypothesis $K : p < p'$. Acceptance of H should result in us choosing π over π', while acceptance of K would lead us to prefer π'. We can use a technique described by Wald (1945, pp. 165) to transform this into a hypothesis testing problem that can be solved using techniques described in previous chapters. The basic idea is to pair the observations made for the two model checking problems. Let $x_1, \ldots, x_m$ be the observations obtained by verifying φ over sample trajectories for $\mathcal{M}[\pi]$ and let $x'_1, \ldots, x'_{m'}$ be the observations obtained by verifying φ over sample trajectories for $\mathcal{M}[\pi']$. We create

```
BETTER-POLICY(𝓜, s₀, φ, π, π′)
  k ⇐ min(|x⃗|, |x⃗′|)    ▷ x⃗ are observations for π and x⃗′ are observations for π′
  d ⇐ 0, n ⇐ 0
  for i ⇐ 1 to k do
    if x_i = 1 ∧ x′_i = 0 then
      d ⇐ d + 1, n ⇐ n + 1
    else if x_i = 0 ∧ x′_i = 1 then
      n ⇐ n + 1
  if 2d ≥ n then
    return π    ▷ p-value F(n − d; n, 0.5)
  else
    return π′    ▷ p-value F(d; n, 0.5)
```

Algorithm 8.3: Statistical comparison of two policies.

pairs $\langle x_i, x'_i \rangle$ of the first $\min(m, m')$ observations. Each pair $\langle 1, 0 \rangle$ is counted as an observation $y_i = 1$ of a Bernoulli variate Y_i for a new hypothesis testing problem, and a pair $\langle 0, 1 \rangle$ is counted as an observation $y_i = 0$. Pairs with matching observations are discarded. It is easy to verify that if π and π' are equally good, then $\Pr[Y_i = 1] = 0.5$ (cf. Wald 1945, p. 166). Let $\tilde{p} = \Pr[Y_i = 1]$. We test H against K by testing the hypothesis $\tilde{H} : \tilde{p} \geq 0.5$ against the alternative hypothesis $\tilde{K} : \tilde{p} < 0.5$ using the observations y_i.

For efficiency, we can reuse the observations already generated by TEST-POLICY. This gives us a predetermined sample of size n, where n is the number of paired observations that differ in value. We can use the same approach as described in Chapter 7 for "black-box" probabilistic verification to test $\tilde{H} : \tilde{p} \geq 0.5$ against $\tilde{K} : \tilde{p} < 0.5$ using a predetermined sample. This gives us a p-value for the decision we make. With $\sum_{i=1}^{n} y_i = d$, the p-value for $\tilde{H}$ is $F(n - d; n, 0.5)$, while the p-value for $\tilde{K}$ is $F(d; n, 0.5)$. Because the threshold is 0.5, the lower p-value is obtained by accepting $\tilde{H}$ if and only if at least half of the observations are positive. Algorithm 8.3 shows code for implementing the procedure BETTER-POLICY in this way.

8.5 Formal Properties of Planning Algorithm

When describing a new planning algorithm, it is common to consider *soundness* and *completeness* of the algorithm. A planning algorithm is sound if every plan that it generates is a valid solution to the planning problem it is given. The algorithm is complete if it generates a plan for every problem that has a solution. A planning algorithm that is both sound and complete is guaranteed to produce a valid plan whenever a solution exists, and it is guaranteed *not* to produce a plan for problems that lack solutions.

Our proposed planning algorithm is sound, so long as TEST-POLICY never accepts a policy that does not satisfy the goal condition. Since we rely on statistical techniques, our planner can give only probabilistic guarantees regarding soundness. For a given policy π, our statistical model checking algorithm guarantees that $\Pr\big[\mathcal{M}[\pi], s_0 \vdash \phi \mid \mathcal{M}[\pi], s_0 \models_{\approx_\perp} \phi\big] \leq \beta$. This means that, in each iteration of the algorithm, we are guaranteed that a policy π is accepted with probability at most β if π is not satisfactory (i.e. $\mathcal{M}[\pi], s_0 \models_{\approx_\perp} \phi$). Since the algorithm halts once we accept a policy, we get an overall bound of β on the probability that FIND-POLICY returns an unsatisfactory policy. We say that the planning algorithm is *β-sound*.

By adopting hill-climbing for policy search, we sacrifice completeness. Even with exhaustive search of the policy space, however, we may still not be able to guarantee completeness. This is because the statistical model checking algorithm could fail to identify a satisfactory policy. We are guaranteed that $\Pr\big[\mathcal{M}[\pi], s_0 \vdash \phi \mid \mathcal{M}[\pi], s_0 \models_{\approx_\top} \phi\big] \geq 1 - \alpha$. If we consider each policy at least once, and there are k satisfactory policies, then the probability is at least $1 - \alpha^k$ that some policy is accepted as a solution. This does *not* mean that the accepted policy is *satisfactory* (that is a matter of soundness rather than completeness). We can increase the probability of producing a policy by visiting policies multiple times during the search. If, for example, we could guarantee that a satisfactory policy was visited an infinite number of times, then the algorithm would produce a policy with probability 1, which in the limit would give us a complete algorithm, assuming that each policy verification is carried out independently. Without an independence assumption, we could guarantee only a $1 - \alpha$ probability of accepting some policy (cf. Theorem 5.4). For instance, the independence assumption would be violated if we reused sample trajectories for the verification of multiple policies.[3] This leads to a $(1 - \alpha)$-*complete* planning algorithm.

8.6 Experimental Results

The results in this section were generated on a PC with a 650 MHz Pentium III processor running Linux. A search limit of 10,000 explored nodes was set for the deterministic planner VHPOP. We used the additive heuristic described by Younes and Simmons (2002a, 2003), which is an adaptation for POCL planning of

[3] Younes and Musliner (2002) describe a probabilistic extension of CIRCA where policies are constructed incrementally. While reuse of sample trajectories is not mentioned explicitly by Younes and Musliner, it is present in the implementation of their approach.

	Event	Rank	Value	$\mu_e + \sigma_e$	Trajectories
first policy	(fill-plane plane pgh-airport)	1.0	-24.1	-0.36	41.8
	(lose-package me pkg mpls-airport)	2.0	-14.7	-0.76	15.0
	(lose-package me pkg pgh-airport)	3.2	-6.8	-0.15	36.4
second policy	(lose-package me pkg mpls-airport)	1.0	-94.3	-0.70	101.6
	(arrive-plane plane pgh-airport mpls-airport)	2.4	-19.9	0.04	99.4
	(move-taxi mpls-taxi mpls-airport)	2.6	-18.2	0.06	107.4

Table 8.2: Top ranking "bugs" for the first two policies of the transportation problem. All numbers are averages over five runs. A rank of 1.0 means that a "bug" was determined to be the worst in all five runs.

the additive heuristic for state space planning first proposed by Bonet et al. (1997).

Consider the transportation problem described earlier in this chapter. There are several things that can go wrong with the initial policy in Figure 8.2(b): the plane can become full or depart before we get to the Pittsburgh airport to check in, the Minneapolis taxi can be serving other customers when we arrive at the Minneapolis airport, and the package can get lost if we stand with it at an airport for too long. The top part of Table 8.2 shows the worst three "bugs" for the initial policy as determined by the sample trajectory analysis. The numbers in the table are averages over five runs with different random seeds, and we used the parameters $\alpha = \beta = 0.01$ (error probability) and $\delta = 0.005$ (half-width of indifference region) with the verification algorithm. By a wide margin, the worst bug is that the plane becomes full before we have a chance to check in. Losing the package at Minneapolis airport comes in second place. Note that the package is more often lost at Pittsburgh airport than at Minneapolis airport, but this bug is not ranked as high because it tends to happen only when the plane already has been filled. The value of the state where the "lose-package" event at Pittsburgh airport occurs is already close to -1 due to an earlier "fill-plane" event, resulting in a mean value of only -0.15 for the "lose-package" event at Pittsburgh airport.

The "fill-plane" bug is repaired by making a reservation before leaving CMU, resulting in the policy shown in Figure 8.5(b). The top three bugs for this policy are shown in the bottom part of Table 8.2. Now, losing the package at Minneapolis airport appears to be the only severe bug left. Note that losing the package at Pittsburgh airport no longer ranks in the top three because the repair for the "fill-plane" bug fortuitously took care of this bug as well. The package is lost at Minneapolis airport because the taxi is not there when we arrive, and the repair found by the planner is to store the package in a safety box until the taxi returns. The policy resulting from this repair satisfies the goal condition, so we are done.

Table 8.3 shows running times for the different parts of the planning algorithm on two variations of the

	$\alpha = \beta$	*first policy* **Verify**	**Analyze**	**Repair**	*second policy* **Verify**	**Analyze**	**Repair**	*third policy* **Verify**
problem 1	10^{-1}	0.044	0.008	0.642	0.232	0.012	0.014	0.176
	10^{-2}	0.084	0.014	0.640	0.470	0.012	0.018	0.344
	10^{-4}	0.160	0.004	0.646	0.974	0.020	0.022	0.698
problem 2	10^{-1}	0.072	0.006	0.666	2.372	0.036	2.468	0.606
	10^{-2}	0.140	0.006	0.670	5.490	0.074	2.496	1.318
	10^{-4}	0.272	0.010	0.682	10.036	0.128	2.568	2.494

Table 8.3: Running times, in seconds, for different stages of the planning algorithm for the original transportation problem (problem 1) and the modified transportation problem (problem 2) with varying error bounds (α and β). All numbers are averages over five runs.

transportation problem. The first problem uses the original transportation domain, while the second problem replaces the possibility of storing a package with an action for reserving a taxi and uses the probability threshold 0.85 instead of 0.9. We can see that the sample trajectory analysis takes very little time. The time for the first repair is about the same for both problems, which is not surprising as exactly the same repair applies in both situations. The second repair takes longer for the second problem because we have to go further back in the failure scenario in order to find a state where we can apply the taxi reservation action so that it has desired effects. The planner tries each initial state for a failure scenario before considering a lower ranked scenario. The search limit determines the amount of effort that is spent on finding a solution for a specific initial state before proceeding with the next alternative. We observe that the sample trajectory analysis finds the same major bugs despite random variation in the sample trajectories across runs and varying error bounds. Verification takes longer for the second policy for problem 2 because the policy is close to satisfactory. In all other cases, the policy is either clearly satisfactory or clearly unsatisfactory.

There is no guarantee that each repair step takes us any closer to a solution. We currently only take the most recent trajectories into account in the failure analysis, which makes it possible to reintroduce a previous bug in an attempt to address a new bug. It is also not clear when to give up on a failure scenario, and imposing a fixed search limit per attempt appears arbitrary. We believe that the failure analysis could be more useful as an aid to human system analysts and engineers designing stochastic systems, as the failure scenarios represent a convenient summary of a large number of trajectories.

Chapter 9

Decision Theoretic Planning

In decision theoretic planning, rewards are introduced that represent positive or negative value to a decision maker, who has to decide on a course of action in light of uncertainty. For example, there is a small chance that we win $1,000,000 on the lottery, but each ticket costs $1. The objective for the decision maker is, roughly speaking, to maximize expected reward.

We introduce the *generalized semi-Markov decision process* (GSMDP), based on the GSMP model of discrete event systems, as a model for decision theoretic planning with asynchronous events and actions. To solve a GSMDP, we present an approximation technique that transforms an arbitrary GSMDP into a continuous-time Markov decision process (MDP). Each non-exponential delay distribution in the GSMDP is approximated by a continuous *phase-type distribution* (see Section 2.1.3). The resulting continuous-time MDP can then be solved using standard solution techniques such as value iteration. We demonstrate our approximation technique on models of different size and complexity and we show that the introduction of phases indirectly allows us to take into account the time spent in a state when selecting actions, which can lead to policies with higher expected reward than if we make selections based only on the current state.

9.1 Generalized Semi-Markov Decision Processes

The generalized semi-Markov process (GSMP), described in Section 2.3.3, is an established formalism in queuing theory for modeling continuous-time stochastic discrete event systems. We add a decision dimension to the formalism by distinguishing a subset of the events as controllable and introducing rewards, thereby obtaining the generalized semi-Markov *decision* process (GSMDP). We limit our attention to *time homogeneous* models with finite state and event sets. For simplicity, we assume that event trigger time distributions are state independent.

9.1.1 Actions, Policies, and Rewards

As in Chapter 8, we associate an enabling condition ϕ_e with each event e and identify a set $A \subset E$ of controllable events, or *actions*. The remaining events $E \setminus A$ are referred to as *exogenous events*. An arbitrary event e, which can be either an action or an exogenous event, is disabled in any state s such that the enabling condition ϕ_e does not hold in s. An exogenous event e is always enabled in a state s if the event's enabling condition ϕ_e holds in s. For an action a, on the other hand, satisfaction of the enabling condition is only a necessary condition for a to be enabled, but a can be kept disabled in s even if ϕ_a holds. A decision maker, or *agent*, can influence system behavior during execution by enabling and disabling actions at will.

A *control policy*, denoted π, determines which action or set of actions should be enabled in any given situation during execution. We allow the action choice to depend on the current state of the process, as well as its entire execution history. The execution history can be captured by a vector $\vec{u}$, with an element u_e for each event e recording the time that e has remained enabled without triggering. The situation space for a process with state space S and event set E is therefore the set $O = S \times [0, \infty)^{|E|}$. A policy is a mapping from situations to sets of actions: $\pi : O \to 2^A$. In situation $o = \langle s, \vec{u} \rangle$, events $E_o^\pi = E_s \setminus \big(A \setminus \pi(o)\big)$ are enabled, i.e. actions not in $\pi(o)$ are disabled. The choice $\pi(o) = \emptyset$ represents idleness. Note that the current situation changes continuously as time progresses, which means that the action choice could change continuously as well. In practice, it can be useful to restrict the size of the action sets that a policy can keep enabled. For example, in a single agent system, we would typically allow at most one action to be enabled at any time.

While in theory it could be beneficial to change the action choice continuously in certain cases, it can hardly be considered practically feasible to do so. We will limit out attention to *piecewise constant* policies, where the action choice is required to remain constant for a duration of time before it can be changed. We can represent such a policy with a mapping τ from situations to positive distribution functions, in addition to the mapping π. At the triggering of an event, we find ourselves in situation o. We enable actions $\pi(o)$ at this point, and keep this choice for a duration of time governed by $\tau(o)$ if no event triggers first. The pair

$\langle \pi, \tau \rangle$ represents a piecewise constant policy. In some situations, as we will see, it is sufficient to consider *stationary* policies, where the action choice is permitted to depend only on the current state and not in any other way on the execution history of the process.

In addition to actions, we specify a reward structure to obtain a GSMDP. We assume a traditional reward structure with a lump sum reward $k_e(s, s')$ associated with the transition from state s to s' caused by the triggering of event e, and a continuous reward rate $c_{A'}(s)$ associated with the set of actions $A' \subset A$ being enabled in s.

Example 9.1. Consider a network of two computers that each can be either up or down. With each computer we associate a crash event c_i, enabled when computer i is up, and a reboot action r_i, enabled when computer i is down. The decision maker plays the role of a system administrator in this example. We can associate an action independent reward rate of $c \in \{0, 1, 2\}$ with states where c machines are up. A reasonable policy for this GSMDP would be to enable reboot action r_i whenever machine i is down. If we can reboot only one machine at a time, due to resource constraints, we could choose to reboot a machine as soon as it crashes. This is reasonable if the reboot time distribution for each computer is memoryless. If reboot time distributions are not memoryless and one machine crashes while we are rebooting another machine, then it may be better to complete the current reboot action before switching to reboot the machine that just crashed.

9.1.2 Optimality Criteria

We will now derive the "Bellman equation" for GSMDPs with piecewise constant policies. The general case leads to a recurrence that we do not expect can be solved exactly. If all delay distributions are exponential, however, a GSMDP is simply a continuous-time MDP. We show how the recurrence equation for such models can be solved using value iteration. This result is relevant for Section 9.2, where we present a technique for approximating a GSMDP that has general delay distributions with one where all delays are exponentially distributed.

We consider two optimality criteria—*expected finite-horizon total reward* and *expected infinite-horizon discounted reward*—both of which can be represented by a universally enabled event that terminates execution in the GSMDP framework. A finite planning horizon can be represented by an event with a deterministic distribution. In the infinite-horizon case, reward earned t time units into the future is discounted by a factor γ^t. This is equivalent to having a termination event with delay distribution $Exp(\alpha)$, such that $\gamma = e^{-\alpha}$

(Howard 1960, p. 114). We thus represent termination by an event $e_\perp$ that always leads to an absorbing state $s_\perp$. No reward is earned after termination, so $c_{A'}(s_\perp)$ is zero for all action sets.

To express the expected future reward for a situation $o = \langle s, \vec{u} \rangle$, given a fixed piecewise constant policy $\langle \pi, \tau \rangle$, we consider all possible schedules (assignments of trigger times) of enabled events that are consistent with the situation at hand. To the enabled events in situation o under policy $\langle \pi, \tau \rangle$, denoted $E_o^{\langle \pi, \tau \rangle}$, we count $e_\perp$ of course, but also another virtual event e_τ with delay distribution $\tau(o)$. The event e_τ represents the point in time when a change of action choice is scheduled by the piecewise constant policy $\langle \pi, \tau \rangle$ without a state transition occuring. A schedule for the events is a vector $\vec{t}$ of size $|E| + 2$. We can define a probability density function over possible schedules as follows:

$$f^{\langle \pi, \tau \rangle}(\vec{t}; \langle s, \vec{u} \rangle) = \prod_{e \in E_o^{\langle \pi, \tau \rangle}} h_e(t_e; u_e) \cdot \prod_{e \in E \setminus E_o^{\langle \pi, \tau \rangle}} \delta(t_e - \infty) \tag{9.1}$$

Here, $\delta(t - t_0)$ is the Dirac delta function (Dirac 1927, p. 625) with the property that $\int_{-\infty}^{x} \delta(t - t_0)\, dt$ is 0 for $x < t_0$ and 1 for $x \geq t_0$. In particular, $\int_{-\infty}^{x} \delta(t - \infty)\, dt$ is 0 for any finite x and 1 for $x = \infty$. We use it in (9.1) to assign zero weight to schedules with a finite trigger time for disabled events. Let $t^* = \min \vec{t}$ and let $e^* = \arg\min \vec{t}$. The expected future reward for a non-terminal situation $o = \langle s, \vec{u} \rangle$ can now be defined using the recurrence

$$\begin{aligned} v^{\langle \pi, \tau \rangle}(o) &= \int\limits_{[0,\infty)^{|\vec{t}|}} \int\limits_0^{t^*} c_{\pi(o)}(s)\, dt + \sum_{s' \in S} p_{e^*}(s'; s) \big(k_{e^*}(s, s') + v^{\langle \pi, \tau \rangle}(O(o, \vec{t}, s')) \big)\, df^{\langle \pi, \tau \rangle}(\vec{t}; o) \\ &= \int\limits_{[0,\infty)^{|\vec{t}|}} t^* c_{\pi(o)}(s) + \bar{k}_{e^*}(s) + \sum_{s' \in S} p_{e^*}(s'; s) v^{\langle \pi, \tau \rangle}(O(o, \vec{t}, s'))\, df^{\langle \pi, \tau \rangle}(\vec{t}; o) \ , \end{aligned} \tag{9.2}$$

where $\bar{k}_e(s) = \sum_{s' \in S} p_e(s'; s) k_e(s', s)$ is the expected transition reward in s when a transition is caused by e, and O is a function providing the next situation. The next situation is $\langle s', \vec{u}' \rangle$, with u'_e increased by t^* if e remains enabled without triggering and otherwise reset to zero. Equation 9.2 is the "Bellman equation" for GSMDPs with piecewise constant policies.

Equation 9.2 involves a high-dimensional probability integral, which suggests that finding an optimal piecewise linear policy for a GSMDP may be hard in the general case. If all delay distributions are exponential, however, then the GSMDP is just a continuous-time MDP, and the recurrence becomes more manageable. We call this a Markovian GSMDP to stress the event structure. The requirement on all delay distributions to be exponential rules out the finite-horizon criterion, which requires a deterministic distribution, but we can handle the infinite-horizon discounted criterion.

The exponential distribution is memoryless, and this means that $h_e(t; u_e) = h_e(t)$ for an event e with an exponential delay distribution. As a consequence, it is of no value to know for how long events have been enabled, as the future behavior of the system depends only on the current state s. A policy for a Markovian GSMDP can be a mapping from states to sets of actions, and there is no need for a τ-component since the relevant situation does not change as time progresses in a state. This means that we need to consider only the class of *stationary* policies in order to act optimally.

In state s, with actions A' chosen to be enabled, the events $E_s(A') = E_s \setminus \big(A \setminus A'\big)$ are enabled, not counting the termination event $e_\perp$ which is enabled in all states. Let λ_e denote the rate of the exponential delay distribution associated with event e, and let α be the rate of the termination event. The time we spend in state s before an event triggers is exponentially distributed with rate

$$\lambda_{A'}(s) = \alpha + \sum_{e \in E_s(A')} \lambda_e \ .$$

The probability that event e triggers first is $\lambda_e/\lambda_{A'}(s)$, and the probability that termination occurs before any event has time to trigger is $\alpha/\lambda_{A'}(s)$. These conditions are easily derived for the exponential distribution, permitting us to write the recurrence for a Markovian GSMDP, defining the expected future reward of a state s under a given policy π, as follows:

$$\begin{aligned} v^\pi(s) &= \int_0^\infty \lambda_{\pi(s)}(s) e^{-\lambda_{\pi(s)}(s)t} \left(tc_{\pi(s)}(s) + \sum_{e \in E_s(\pi(s))} \frac{\lambda_e}{\lambda_{\pi(s)}(s)} \sum_{s' \in S} p_e(s'; s) \Big(k_e(s, s') + v^\pi(s') \Big) \right) dt \\ &= \frac{1}{\lambda_{\pi(s)}(s)} \left(\bar{r}_{\pi(s)}(s) + \sum_{e \in E_s(\pi(s))} \lambda_e \sum_{s' \in S} p_e(s'; s) v^\pi(s') \right) \end{aligned}$$

In the above equation, $\bar{r}_{A'}(s)$ denotes the quantity $c_{A'}(s) + \sum_{e \in E_s(A')} \lambda_e \sum_{s' \in S} p_e(s'; s) k_e(s, s')$, which essentially is the expected reward per time unit in state s until the next state is reached. We can swap the order of the two summations to obtain

$$\begin{aligned} v^\pi(s) &= \frac{1}{\lambda_{\pi(s)}(s)} \left(\bar{r}_{\pi(s)}(s) + \sum_{s' \in S} \sum_{e \in E_s(\pi(s))} \lambda_e p_e(s'; s) v^\pi(s') \right) \\ &= \frac{1}{\lambda_{\pi(s)}(s)} \left(\bar{r}_{\pi(s)}(s) + \sum_{s' \in S} w_{\pi(s)}(s'; s) v^\pi(s') \right) \ , \end{aligned} \tag{9.3}$$

where $w_{A'} = \sum_{e \in E_s(A')} \lambda_e p_e(s'; s)$.

The maximum expected reward is obtained by choosing the set of actions that maximizes the reward in the current state and act optimally in subsequent states. We can express this with the recurrence

$$v^*(s) = \max_{A' \subset A} \frac{1}{\lambda_{A'}(s)} \left(\bar{r}_{A'}(s) + \sum_{s' \in S} w_{A'}(s'; s) v^*(s') \right) , \tag{9.4}$$

derived from (9.3). Equation 9.4 forms the basis for value iteration for Markovian GSMDPs. Note the striking resemblance with (2.24) for discrete-time MDPs. Remember that the discount factor, $\gamma = e^{-\alpha}$, is present in $\lambda_{A'}(s)$. We can also write (9.4) using matrix notation:

$$V^* = \max_{\vec{A}' \subset A^{|S|}} H_{\vec{A}'} \circ \left(\bar{R}_{\vec{A}'} + W_{\vec{A}'} V^* \right) \tag{9.5}$$

The operator $\circ$ represents *Hadamard product* (element-wise matrix multiplication). This form is convenient, for example, when implementing value iteration using MTBDDs. The row vector $H_{\vec{A}'}$ represents the expected holding time in each state.

9.2 Approximate Solution Technique

The previous section provided a dynamic programming formulation of optimal GSMDP planning. We noted that the general case involves a high-dimensional probability integral, which limits the practical use of the formulation. If all delay distributions are exponential, however, a GSMDP is simply a continuous-time MDP, and the dynamic programming formulation becomes manageable as shown in (9.4). We now take advantage of this fact, presenting an approximate solution technique for GSMDPs that uses phase-type distributions.

To find a policy for a GSMDP, we first approximate it with a continuous-time MDP by approximating each non-exponential delay distribution with a phase-type distribution. Recall that phase-type distributions (Section 2.1.3) represent the time from entry until absorption in a Markov process with n transient states (phases) and a single absorbing state. The continuous-time MDP can be solved exactly, for example by using value iteration. We can also use uniformization to obtain a discrete-time MDP, in case we want to use an existing solver for discrete-time models. The resulting policy, in either case, may be phase-dependent. Phase transitions do not occur in the actual model, so in order to execute the policy in the real world, we simulate phase transitions. Our solution method is summarized in Figure 9.1.

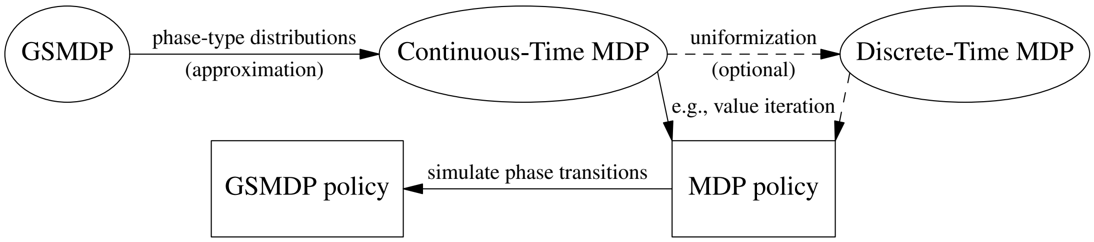


Figure 9.1: Schematic view of solution technique for GSMDPs.

9.2.1 From GSMDP to MDP

We first present our method for approximating a GSMDP with an MDP. We have noted that if all events of a GSMDP have exponential delay distributions, then the GSMDP is simply a continuous-time MDP with a factored transition model. By using phase-type distributions, we can replace each non-Markovian event in the GSMDP with one or more Markovian events, thereby obtaining a continuous-time MDP that approximates the original GSMDP.

A GSMDP event is represented by a triple $\langle \phi_e, G_e, p_e \rangle$, and we assume a factored representation of the state space with state variables V. We also assume that p_e is implicitly represented by an effect formula eff_e, using the effect formalism described in Section 8.2.1 with the addition of numeric state variables.

For each non-Markovian event e with delay distribution G_e, we find a phase-type distribution of order n_e approximating G_e. We add a phase variable ph_e to V for each event e with $n_e > 1$ and replace e with one or more Markovian events. A phase-type distribution consists of a set of phase transitions. Each phase transition can be represented by a Markovian event. We assume that the initial phase is always $ph_e = 1$, as this will simplify the handling of interacting events. A phase transition from phase i to phase j with rate λ_{ij} is represented by an event with enabling condition $\phi_e \wedge ph_e{=}i$ and delay distribution $Exp(\lambda_{ij})$. The effect formula for the phase transition event, ignoring for the moment possible event interactions, is $ph_e \leftarrow j$ if $j \leq n_e$ and $\mathit{eff}_e \wedge ph_e \leftarrow 1$ otherwise (a transition to phase $n_e + 1$ represents the triggering of the original event e and resets the phase to its initial value). We associate a transition reward of zero with pure phase transitions and $k_e(s, s')$ with phase transitions representing the triggering of event e.

The triggering of an event e in state s can cause another event e', enabled in s, to become disabled in the state following the triggering of e. When e disables a non-Markovian event e', we should reset the phase of the phase-type distribution for e' (i.e. set $ph_{e'}$ to one). We can think of the phases as a partitioning into random-length intervals of the time, u_e, that an event e has remained continuously enabled without triggering. Resetting the phase of an event corresponds to resetting u_e to zero. To account for this sort of interaction between events, we need to modify the effects of events that do not simply change the value of a phase variable. Let ϕ'_e represent the condition of an event e, but evaluated in the next state rather than the current state. The final effect formula for an event e is obtained by adding the effect $(\phi_{e'} \wedge \neg\phi'_{e'}) \rhd ph_{e'} \leftarrow 1$ to eff_e for all non-Markovian events $e' \neq e$.

We now have a method for approximating a GSMDP with a continuous-time MDP. If there is a close match between the delay distributions of the GSMDP and the phase-type distributions used in the MDP, then we expect the approximation to be close, although we have no quantitative measure for how good the approximation is. Section 9.3 provides evidence that the approximation technique works well in practice.

9.2.2 Policy Execution

The execution history of a GSMDP can be represented by a set of real-valued variables, one for each event $e \in E$ representing the time e has been continuously enabled without triggering. The phases introduced when approximating a GSMDP with a continuous-time MDP can be thought of as a randomized discretization of the time events have remained enabled. For example, approximating G with an n-phase Erlang distribution with parameters p and λ represents a discretization of the time G has been enabled into n random-length intervals. The length of each interval is a random variable with distribution $Exp(\lambda)$. A policy for the continuous-time MDP with phase transitions is therefore approximately a mapping from states and the times events have been enabled to actions for the original GSMDP. We can also think of phase transitions as a factored representation of the distribution $\tau(o)$, which governs the time to spend in a state before considering a change of action choice.

Phase transitions are not part of the original model, so we have to simulate them when executing the policy obtained for the approximate model. When a GSMDP event or action e becomes enabled during execution, we sample a first phase transition time t_1 for the phase-type distribution used to approximate G_e. If e remains enabled for t_1 time units without triggering, we increment the phase associated with e and sample a second phase transition time t_2. This continues until e triggers or is disabled, in which case the phase is reset to one, or we reach the last phase, in which case the phase does not change until e triggers or is disabled. The action choice can change every time a simulated phase transition occurs, although phase transitions do not change the actual state of the process. This allows us to take into account the time spent in a state when selecting which action to enable. We will see in the next section that this can produce better policies than if actions are chosen only at actual state transitions.

The phases can also be thought of as partially observable state variables. We cannot observe the phase of a trigger time distribution. We can observe actual state transitions, however, so we know how much time we have spent in a state. At any given time, we can compute a probability distribution over phases, which can be used to select the action to enable at that time. This is analogous to the QMDP solution technique for partially observable MDPs (Littman et al. 1995), and could result in higher expected value during execution than if phase transitions are simulated. One disadvantage, however, is that time is continuous, which means that the belief distribution over phase assignments changes continuously. In practice, we could select a frequency at which to update the belief distribution and reconsider the current action choice, but there is no clear choice for such an update frequency. It may be wasteful to update the belief state with high frequency, and we risk missing important phase changes if the update frequency is too low. Belief tracking may be computationally expensive as well. We leave it to future research to explore this, and other alternative ways, of executing a phase-dependent policy.

9.3 Experimental Results

We have implemented a basic GSMDP planner based on the solution procedure outlined in Figure 9.1. Our implementation uses MTBDDs to represent matrices and vectors, similar to the approach proposed by Hoey et al. (1999) for discrete-time MDPs. MTBDDs use Boolean state variables, and we need $\lceil \log s \rceil$ bits to represent the phase of a phase-type distribution with s phases. The experimental results were generated on a 3 GHz Pentium 4 PC running Linux, and with an 800 MB memory limit set per process.

9.3.1 Preventive Maintenance ("The Foreman's Dilemma")

Our first test case is a variation of Howard's "the Foreman's Dilemma" (Howard 1960), where we have a machine that can be working (s_0), failed (s_1), or serviced (s_2). This example is meant to show that it can be beneficial to delay the enabling of an action in a state, and phases allow us to do so. A failure event with delay distribution G is enabled in s_0 and causes a transition to s_1. Once in s_1, the repair time for the machine has distribution $Exp(1/100)$. At any time in s_0, the foreman can choose to enable a service action with delay distribution $Exp(10)$. If this action triggers before the failure event, the system enters s_2 where the machine is being serviced with service time distribution $Exp(1)$. Given reward rates $c(s_0) = 1$, $c(s_1) = 0$, and $c(s_2) = 1/2$, independent of action choice, and no transition rewards, the problem is to produce a service policy that maximizes the expected infinite-horizon discounted reward in s_0.

Depending on the failure time distribution G, it may be beneficial to enable the service action at some point in s_0. This is because it takes a long time to recover from failure, while the return to s_0 from the service state is quick. Still, the reward rate is highest in s_0, so there is an incentive to delay the enabling of the service action. The optimal policy in this case is to enable the service action after spending t_0 time units in s_0, where the best choice for t_0 depends on the shape of G. The lower the probability is that failure occurs early, the later we can schedule to enable the service action.

We can model this problem as an SMDP, noting that the probability of the service action triggering before the failure event is $p_{02} = 1 - \int_{t_0}^{\infty} 10e^{-10(t-t_0)} F(t)\, dt$ (where $F(t)$ is the cumulative distribution function for G) if we enable the service action after t_0 time units in s_0. We can solve the SMDP using the techniques described by Howard (1971b), but then we can choose to enable the action in s_0 only immediately ($t_0 = 0$) or not at all ($t_0 = \infty$). Alternatively, we can express the expected reward in s_0 as a function of t_0 and use numerical solution techniques to find the value for t_0 that maximizes the expected reward. Depending on the shape of $F(t)$, both approaches may require numerical integration over semi-infinite intervals.

Figure 9.2 plots the expected discounted reward, as a percentage of optimal, for policies obtained using standard SMDP solution techniques as well as our technique for approximating a (G)SMDP with an MDP using phase-type distributions. A uniform failure time distribution over the interval $(5, b)$ was used. The optimal value and the value for the SMDP solution were computed numerically using MATLAB, while the other values were computed by simulating execution of the phase-dependent policies and taking the average discounted reward over 5000 sample trajectories. We used $\gamma = 0.95$ as the discount factor.

Note that the SMDP solution is well below the optimal solution because it has to enable the service action either immediately, or not at all, in s_0. For small values of b, the optimal SMDP policy is to enable the action in s_0, but as b increases so does the expected failure time, so for larger b it is better not to enable the action because it allows us to spend more time in s_0 where the reward rate is highest. The performance of the policy obtained by matching a single moment of G is almost identical to that of the SMDP solution.

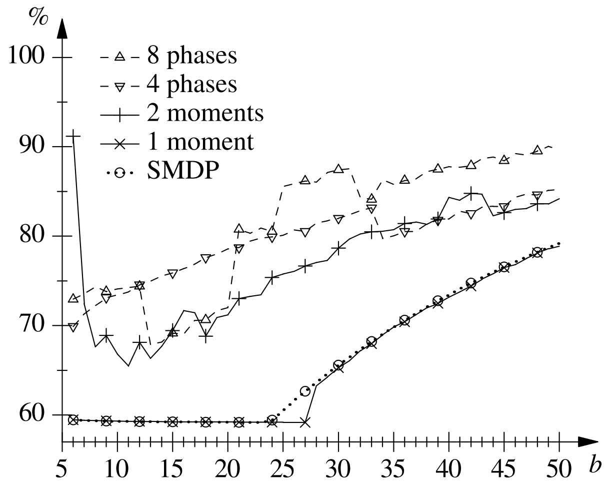


Figure 9.2: Policy value, as a percentage of the optimal value, for the Foreman's Dilemma with the failure time distribution G being $U(5, b)$.

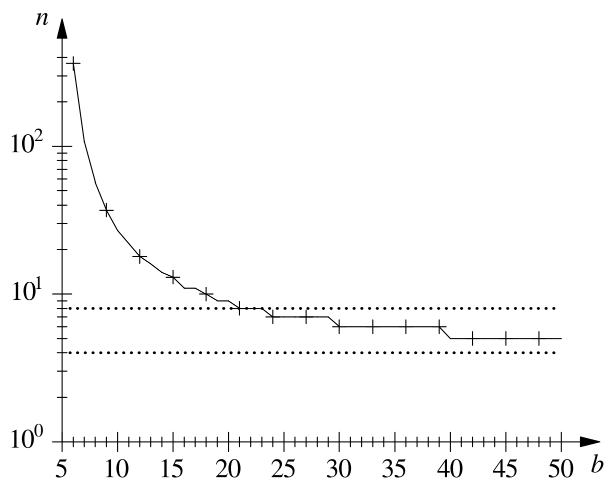


Figure 9.3: Number of phases required to match two moments of a uniform distribution over the interval $(5, b)$. The dotted lines indicate 4 and 8 phases.

This policy is also restricted to enabling the action either immediately or nor at all in s_0, since there is just one phase in the approximation. Due to the approximation of G, the performance is slightly worse around the point where the optimal SMDP policy changes. We can see that by matching two moments (with a generalized Erlang distribution), the quality of the policy can be increased significantly. Note that the number of phases required to match two moments of $U(5, b)$ varies with b, as is shown in Figure 9.3. For $b = 6$, over 300 phases are needed, which helps to explain the high quality at this point for the policy obtained by matching two moments. We also show the value for policies obtained by fixing the number of phases and using the EM algorithm to find a phase-type distribution with good fit. Note that using 8 phases instead of 4 actually hurts the quality of the policy for some values of b. In these cases, the 8-phase distribution causes the enabling of the service action to be delayed for too long.

Figure 9.4 shows the performance of policies for a different failure time distribution—a Weibull distribution with parameters $1.6a$ and 4.5. In this case, a 16-phase generalized Erlang distribution is sufficient to match two moments for all values of a. We can see the policy obtained by using 8 phases and EM fitting actually outperforms the policy obtained by matching two moments, if only slightly, and we can get even better performance by using 24 phases. For $a = 10$, the solution obtained with 8 phases gives us a 34 percent increase in value compared to the SMDP solution, and the value increase is 50 percent with 24 phases. The SMDP and single moment solutions again have almost identical performance, and are for the

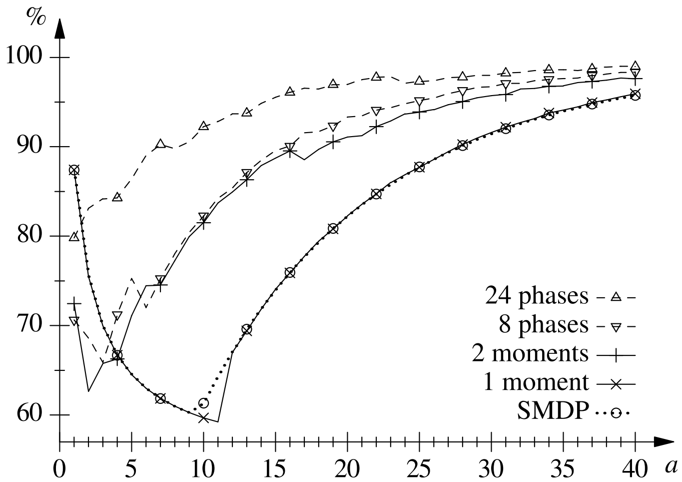


Figure 9.4: Policy value, as a percentage of the optimal value, for the Foreman's Dilemma with the failure time distribution G being $W(1.6a, 4.5)$.

most part significantly worse than the other solutions. The only exception is for low values of a, in which case the phase-type distributions underestimate the probability that failure will occur at a very early stage, so the enabling of the service action comes later than needed to perform well. In most situations, however, using more phases gives better policies, mainly because the additional phases allow us to better account for the fact that G is not memoryless. For the Foreman's Dilemma, this is crucial as it allows us to delay the enabling of the service action in s_0, taking into account the fact that failure is unlikely to occur early on.

9.3.2 System Administration Problem

Our second test case is a system administration problem, loosely based on a similar problem described by (Guestrin et al. 2003). While the first test case illustrated that phases can result in better policies by delaying the enabling of an action in a state, this test case illustrates that phases can help by keeping an action enabled if it has already been enabled for some time. In both cases, phases introduce memory into the state space.

In the system administration problem, there is a network of n computers, with each computer being either up or down. There is a crash event for each computer that can cause a computer that is currently up to go down at a random point in time. The delay of the crash event is governed by an exponential distribution with unit rate. To make this a decision problem, we add a reboot action for each machine that can be enabled whenever a machine is down. The delay distribution for this action is $U(0, 1)$. The reward

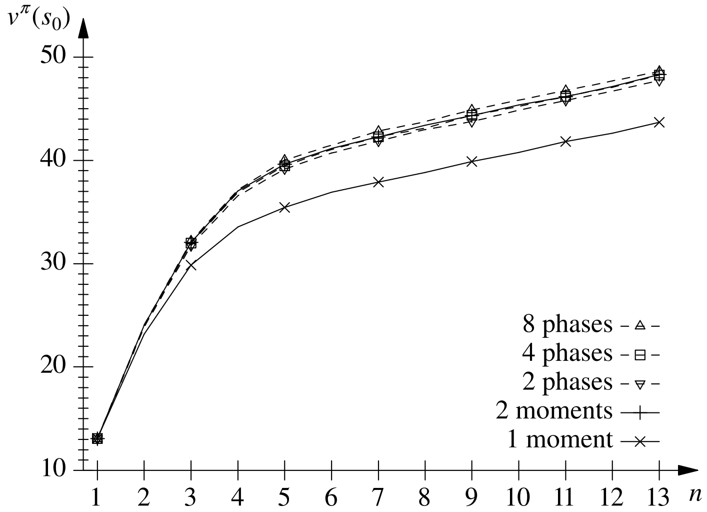


Figure 9.5: Expected discounted reward for the system administration problem with n machines. The expected reward is reported for state s_0 with all n machines up.

rate for a state is equal to the number of machines that are up, so in a state with all machines up we earn a reward of n per time unit. We assume that there is a single system administrator managing the network, so only a single reboot action can be enabled at any point in time. Unlike the previous test case, this is not an SMDP, except for $n = 1$, because a reboot action may remain enabled across state transitions (caused by a crash event). We therefore cannot solve this problem using existing SMDP solution techiques. The obvious solution is to reboot a machine whenever it goes down, and wait until rebooting is finished before going on to reboot another machine. The problem is that in a Markov formulation we would not know that we have been rebooting a machine when another machine goes down. The introduction of phases gives us that information and therefore enables us to obtain better policies.

Figure 9.5 plots the expected discounted reward ($\gamma = 0.95$) of the policy obtained by our GSMDP planner when approximating each uniform distribution with a phase-type distribution. We report the values obtained when matching one and two moments (using a three phase Erlang distribution), and when fixing the number of phases per uniform distribution to 2, 4, and 8. By using the EM algorithm with at least two phases, we can increase the expected reward by up to 10 percent compared with the solution obtained by matching only a single moment. When matching a single moment, we can enable a reboot action based only on which machines are currently down, and the resulting policy reboots machine i before machine j if $i < j$. In contrast, the policy obtained when using multiple phases keeps a reboot action enabled if it is in a phase

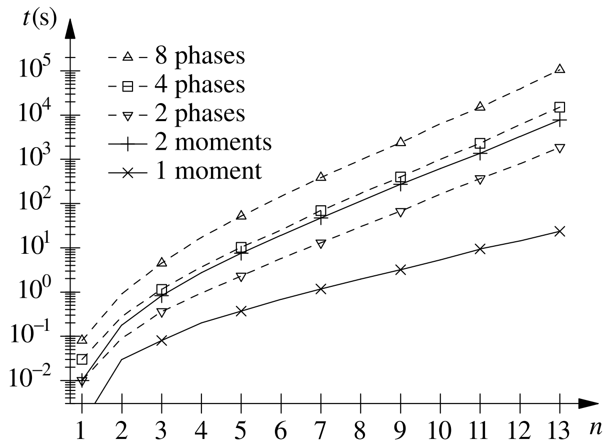


Figure 9.6: Planning time for the system administration problem.

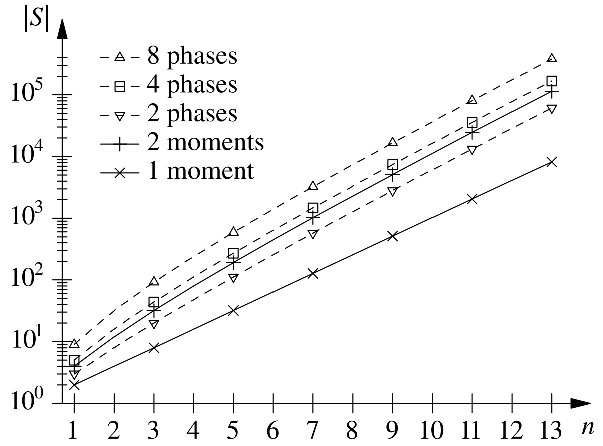


Figure 9.7: Size of potentially reachable state space for the system administration problem.

other than one, because it is expected to trigger soon. By using more than two phases, we can increase the expected reward even further, although the increase is not as significant.

By increasing the number of phases used to represent a non-exponential distribution, we increase the accuracy of the approximation, but we also increase the state space. In terms of planning time, a larger state space means that a solution will take longer to obtain. Thus, as one can expect, in general, better policies are obtained at the price of longer solution times. The solution time for the system administration problem, not including phase-type fitting[1] and model construction, is shown in Figure 9.6. Figure 9.7 plots the size of the potentially reachable state space (from the state with all machines up) as a function of the number of machines, n. If we use s phases to represent a non-exponential distribution, then the size of the reachable state space is at most $((s-1)n/2+1) \cdot 2^n$. Note that $d = (\lceil \log s \rceil + 1)n$ Boolean state variables are used for a problem with n machines, but the reachable state space is significantly smaller than 2^d for $s > 1$. For $n = 13$ and $s = 8$, we have $d = 52$, while the size of the state space is under $4 \cdot 10^5$ ($< 2^{19}$).

9.3.3 State Filtering and Uniformization

We conclude the empirical evaluation of our planning approach with a discussion of techniques for reducing planning time. The first technique is related to the use of Boolean state variables to encode the phase of a distribution. If the number of phases is not a power of 2, then we are potentially introducing spurious states

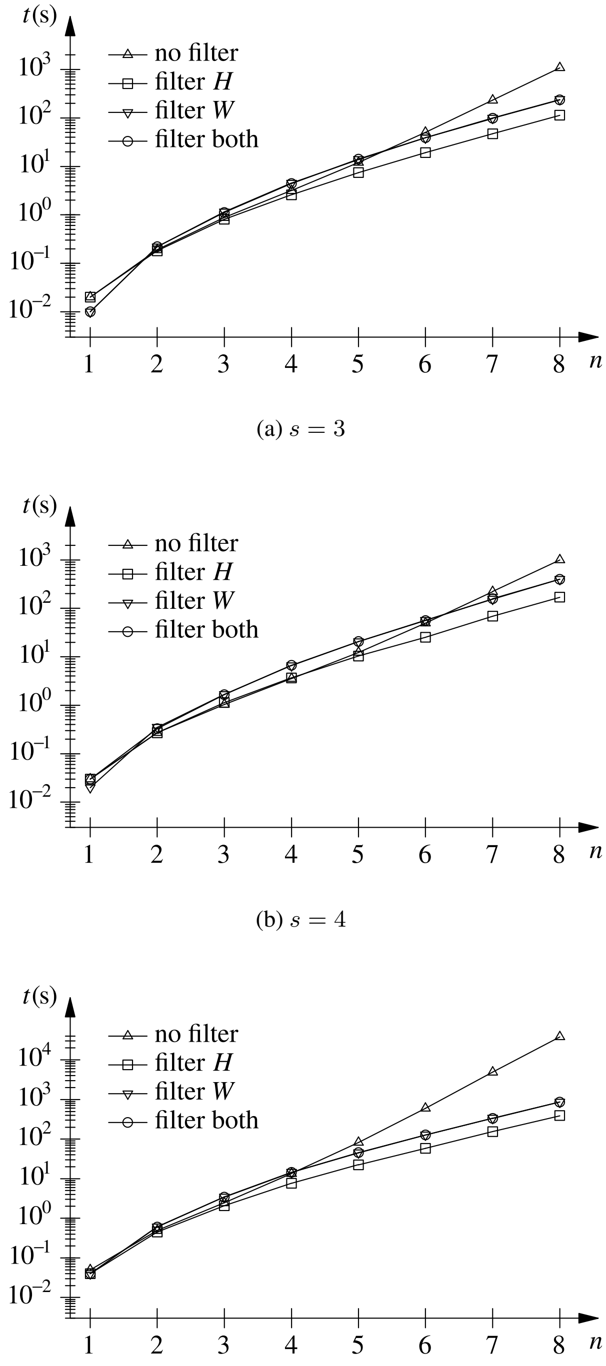
[1]The time for phase-type fitting ranges from a few milliseconds (2 phases) to a few minutes (8 phases) for the EM approach.

into the model. This could mean that we are wasting time computing the optimal action choice for irrelevant states. It is also the case that the phase associated with the delay distribution for an action or event is not significant when the action or event is disabled. By convention we set the phase to one for disabled actions and events, so a different phase assignment for a diabled action or event corresponds to a spurious state.

Consider the recurrence in (9.5), which we use in our implementation of value iteration for continuous-time MDPs. To avoid computing the optimal action choice for evidently spurious states, we can apply a filter to the vectors and/or the matrix involved in the computation. For example, we can set all row elements of $H_{\vec{A}'}$ to zero for spurious states, or we could set to zero all entries of $W_{\vec{A}'}$ corresponding to such states. Applying a filter to a vector or matrix represented by an MTBDD could result in a larger representation, which could result in increased planning times. Figure 9.8 shows the effect of filtering for the system administration problem, with different choices of s (the number of phases to use for each non-exponential distribution). We can see that filtering just the $H_{\vec{A}}$ vectors results in the best performance, while using no filter at all leads to a noticeable performance degradations as n increases. Filtering helps even when s is a power of 2, because the phase is forced to be one for reboot actions that are not enabled.

We can solve a continuous-time MDP directly, using the recurrence in (9.4). Alternatively, we can use uniformization to transform the continuous-time MDP into a discrete-time MDP, and solve the resulting problem. Uniformization is a technique by which we transform a continuous-time MDP with state-dependent exit rates into an equivalent continuous-time MDP with the same (uniform) exit rate for all states. The uniform continuous-time MDP can then be treated as a discrete-time MDP resulting from observing the original continuous-time MDP at a constant rate. Uniformization introduces self-transitions not present in the original model, because it is possible that we remain in the same state from one observation to another. While uniformization seems to be promoted as the standard solution technique for continuous-time MDPs (cf. Puterman 1994), it is not clear what the benefit is of using uniformization rather than solving the continuous-time MDP directly. In fact, as Figure 9.9 indicates, uniformization can actually hurt performance. The introduction of virtual self-transitions increases the complexity of the transition matrix, which makes each iteration of value iteration take longer time.

In conclusion, we have shown that phase-type distributions are useful for solving decision theoretic planning problems with asynchronous events and actions. Using more phases often results in better policies, but also increased planning times. State filtering can help to reduce planning times.

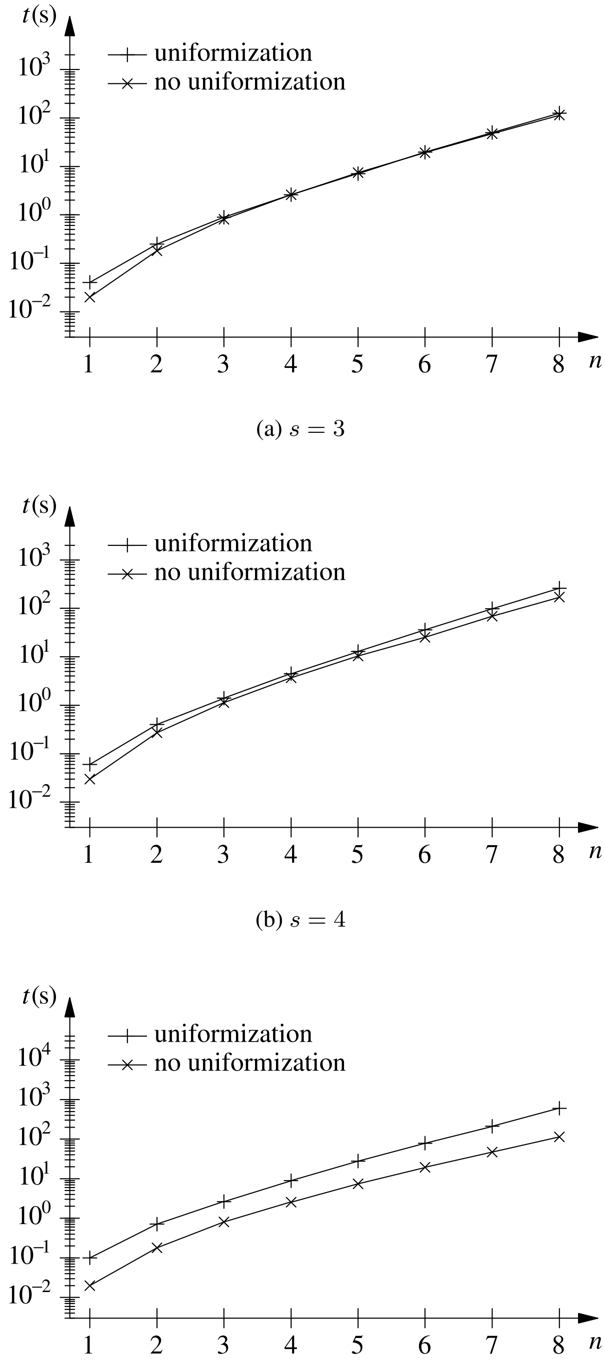


(a) $s = 3$

(b) $s = 4$

(c) $s = 5$

Figure 9.8: The effect of state filtering for the system administration problem.



(a) $s = 3$

(b) $s = 4$

(c) $s = 5$

Figure 9.9: Performance with and without uniformization for the system administration problem.

Chapter 10

Conclusion and Future Work

At the outset of this thesis, we embarked on an ambitious endeavor to develop algorithms for *both* planning *and* verification with asynchronous events. We believe our research effort to be a good start in the direction towards practical solution techniques for asynchronous stochastic systems, but we most certainly acknowledge that we have only scraped the surface of this vast area of research.

In verification, we have established the foundations of statistical probabilistic model checking. A key observation is that probabilistic model checking can be modeled as a hypothesis testing problem. We can therefore use well-established and efficient statistical hypothesis testing techniques, in particular sequential acceptance sampling, for probabilistic model checking. Our model checking approach is not tied to any specific statistical test. The only requirement is that we can bound the probability of an incorrect answer (either a false positive or a false negative). A potential benefit of statistical techniques is that they tend to be highly amenable to parallelization. We show this to be the case for statistical model checking, although some care must be take so as not to introduce bias in the sampling process. Our solution to this problem results in a distributed algorithm for probabilistic model checking that can take full advantage of a heterogeneous computing environment without the need for any explicit communication of performance characteristics.

We have considered only transient properties of stochastic systems. The logic CSL, as described by Baier et al. (2003), can also express steady-state properties. Statistical techniques for steady-state analysis exist, including *batch means analysis* and *regenerative simulation* (Bratley et al. 1987). Although these techniques have been used for statistical *estimation*, we are confident that they could be adapted for hypothesis testing, as well. Extending our work on statistical probabilistic model checking to steady-state properties is therefore a prime candidate for future work. To more efficiently handle probability thresholds close to zero and one, the use of *importance sampling* (Heidelberger 1995) may also be possible. It would moreover be worthwhile exploring Bayesian techniques for acceptance sampling, in particular the test developed by Lai (1988). It is well-known that the sequential probability ratio test, while generally very efficient, tends to require a large sample size if the true probability lies in the indifference region of the test, which is unfortunate because we spend the most effort where we are indifferent of the outcome. This shortcoming is addressed by Bayesian hypothesis testing. The challenge would be to devise a Bayesian test for conjunctive and nested probabilistic operators. A final topic for future work, which we have not discussed much in this thesis, is to improve the efficiency of discrete event simulation for our representation of stochastic discrete event systems. A bottleneck in our current implementation is the determination of enabled events in a state. Our solution is to scan through the list of all events and evaluate the enabling condition for each event. This is not efficient for models with many events. We think that perhaps the use of symbolic data structures, such as BDDs and MTBDDs, could speed up the generation of sample trajectories.

Our contribution to the artificial intelligence community is a formalism for planning with asynchronous events in stochastic environments. We base this formalism on an established model in queuing theory, the generalized semi-Markov process. Asynchronous stochastic systems have been largely absent in AI research on planning. We hope that we can inspire further research on this topic with the establishment of a formal model for stochastic decision processes with asynchronous events. We have presented two approaches to planning with asynchronous events, both with merits and limitations.

For goal directed planning, we have developed an approach based on the Generate, Test and Debug paradigm. Statistical model checking is used to verify policies, and the simulation traces generated during verification are used to guide policy repair. We have demonstrated that this approach can be used for automated policy repair. However, there is no guarantee that a repair step takes us closer to a solution, and the selection of repair steps is hard to automate for more complex bugs. We believe that the analysis techniques would be more useful as an aid to human system analysts and engineers. To make this work, we need to develop tools for visualizing the information gathered from the simulation traces. The failure scenarios that we extract could be valuable information to a system analyst trying to debug a faulty system design.

To solve decision theoretic planning problems with asynchronous events, we have used phase-type distributions. We have experimented with different methods for approximating a general distribution with a phase-type distribution, and we have shown that the introduction of phases makes it possible to generate policies of higher quality than if we simply assume that all events have exponential delay distributions. A limitation of our approach is that we cannot guarantee that the approximate solution is approximately optimal, although using more phases generally results in better policies. It is not even clear what the shape of an optimal policy for a GSMDP is, nor is it evident that optimal GSMDP planning is decidable in the general case. We take a pragmatic approach by at least generating a policy that almost always is better than the one obtained by simply ignoring history dependence. A thorough theoretical analysis of the GSMDP formalism is currently lacking, and is a clear candidate for future research. We would also like to explore alternative approximate solution techniques for GSMDPs, including value function approximation.

It is clear that there are systems in the real world for which the Markov assumption is inappropriate. This is, in particular, the case for many systems with asynchronous events. We have provided practical techniques for verification and planning for such systems. We have presented a statistical approach to probabilistic verification, which is applicable to any stochastic discrete event system. The user is given only probabilistic correctness guarantees, but the alternative is to use an approximate model amenable to numerical verification techniques and it is generally hard to quantify the effect that a model approximation has on the validity of the verification result. For planning, we have demonstrated that the use of phase-type distributions can allow us to generate control policies with greater expected value than if we ignored history dependence. Models with phase information are more complex and therefore take longer time to solve. In many situations, however, we need to generate a control policy only once for a system and the same policy can be used repeatedly. Even a small increase in efficiency of a manufacturing process, for instance, can lead to a large profit increase for a business. In future research, we plan to identify several real-world applications for the techniques we have developed.

Appendix A

Input Language for Model Checker

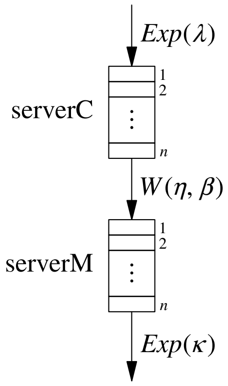
The experimental results presented in Chapter 6 were generated by the probabilistic model checker YMER.[1] The input language used by YMER is based on the PRISM language (Parker 2002), which takes inspiration from Alur and Henzinger's (1999) Reactive Modules formalism.

A.1 Modular Specification of Stochastic Discrete Event Systems

The model of a stochastic discrete event system is specified as a set of asynchronous modules. Figure A.1 shows a GSMP model of a tandem queuing network and its representation in the YMER input language. The model has two modules: serverC and serverM. A set of local state variables SV_m and a set of events E_m is associated with each module m. The state variables sc and sm in our simple example are used to record the number of items currently stored in each of the queues. A model can also have a set of global state variables SV_g. For the tandem queuing network, SV_g is empty. The set of all state variables, $SV = SV_g \cup \bigcup_m SV_m$, constitutes a factored representation of the state space for the model.

Each event e has an enabling condition ϕ_e, which is a logic formula over the state variables SV. An event e is enabled in a state s if and only if $s \models \phi_e$. The enabled events in a state race to trigger first. The trigger time for each event e is determined by a positive distribution G_e. YMER currently supports the exponential, Weibull, lognormal, and uniform distributions. Only continuous distributions are permitted in order to avoid complications arising from the simultaneous triggering of multiple events, which could be a

[1] YMER web site: http://www.cs.cmu.edu/~lorens/ymer.html




```
gsmp

const n = 63;

rate lambda = 252; // 4*n
rate eta = 1;
rate beta = 1/2;
rate kappa = 4;

module serverC
    sc : [0..n];

    [] (sc<n) -> lambda : sc'=sc+1;
    [route] (sc>0) -> W(eta,beta) : sc'=sc-1;
endmodule

module serverM
    sm : [0..n];

    [route] (sm<n) -> 1 : sm'=sm+1;
    [] (sm>0) -> kappa : sm'=sm-1;
endmodule
```

Figure A.1: A tandem queuing network (left) and its representation in the input language used by YMER (right).

source of nondeterminism. The triggering event in a state updates the values of state variables local to the module that the event is associated with. An event is also permitted to update global state variables, but cannot change the value of state variables that belong to a different module.

It is possible to synchronize the update of state variables from different modules. The event with a Weibull distribution that routes messages from serverC to serverM is an example of this in the specification of Figure A.1. There is one event in each module with a synchronization label "route", and these two events are paired into a single event. The condition for the composite event is the conjunction of the individual event conditions, and the update list for the composite event is the concatenation of the update lists for the individual events. All but one of the individual events must have an exponential trigger time distribution with unit rate, specified as 1. The trigger time distribution for the composite event is taken from the individual event that has a different trigger time distribution. In the tandem queuing network model, the trigger time distribution for the composite event is taken from the event in the serverC module. Synchronizing events are not permitted to update the same global variable in an inconsistent manner, as this would lead to an under specified model.

A.2 BNF Grammar

This section presents the full syntax for YMER's input language using an extended BNF notation with the following conventions:

- Each rule is of the form ⟨*non-terminal*⟩ ::= *expansion*.
- Alternative expansions are separated by a vertical bar ("|").
- An asterisk ("*") following a syntactic element *x* means zero or more occurrences of *x*.
- Terminals are written using `typewriter` font.
- Case *is* significant. For example, `X` and `x` are separate identifiers.
- Parentheses and square brackets are an essential part of the syntax and have no semantic meaning in the extended BNF notation.
- Any number of whitespace characters (space, newline, tab, etc.) may occur between tokens.

There are two top-level syntactic elements that may occur in an input file: ⟨*model*⟩ and ⟨*property*⟩. A ⟨*name*⟩ is a string of characters starting with an alphabetic character followed by a possibly empty sequence of alphanumeric characters, hyphens ("−"), and underscore characters ("_"). A ⟨*pname*⟩ is a name immediately followed by a prime symbol ("`'`"). An ⟨*integer*⟩ is a non-empty sequence of digits. A ⟨*number*⟩ is a sequence of numeric characters, possibly with a single decimal point ("`.`") at any position in the sequence, or two integers separated by a slash "`/`". A ⟨*probability*⟩ is a number with a value in the interval $[0, 1]$.

⟨*model*⟩ ::= ⟨*model-type*⟩ ⟨*declaration*⟩* ⟨*module*⟩*
⟨*model-type*⟩ ::= `stochastic` | `ctmc` | `gsmp`
⟨*declaration*⟩ ::= `const` ⟨*name*⟩ `=` ⟨*integer*⟩ `;`
  | `rate` ⟨*name*⟩ `=` ⟨*number*⟩ `;`
  | `global` ⟨*name*⟩ `:` ⟨*range*⟩ `;`
  | `global` ⟨*name*⟩ `:` ⟨*range*⟩ `init` ⟨*expr*⟩ `;`
⟨*range*⟩ ::= `[` ⟨*expr*⟩ `..` ⟨*expr*⟩ `]`
⟨*module*⟩ ::= `module` ⟨*name*⟩ ⟨*variable-decl*⟩* ⟨*command*⟩* `endmodule`

| `module` ⟨*name*⟩ = ⟨*name*⟩ `[` ⟨*substitution-list*⟩ `]` `endmodule`

⟨*substitution-list*⟩ ::= ⟨*name*⟩ = ⟨*name*⟩ | ⟨*name*⟩ = ⟨*name*⟩ `,` ⟨*substitution-list*⟩

⟨*variable-decl*⟩ ::= ⟨*name*⟩ `:` ⟨*range*⟩ `;`

| ⟨*name*⟩ `:` ⟨*range*⟩ `init` ⟨*expr*⟩ `;`

⟨*command*⟩ ::= ⟨*synchronization*⟩ ⟨*formula*⟩ `->` ⟨*distribution*⟩ `:` ⟨*update*⟩ `;`

⟨*synchronization*⟩ ::= `[ ]` | `[` ⟨*name*⟩ `]`

⟨*formula*⟩ ::= ⟨*formula*⟩ `&` ⟨*formula*⟩ | ⟨*formula*⟩ | ⟨*formula*⟩ | `!` ⟨*formula*⟩

| ⟨*expr*⟩ ⟨*binary-comp*⟩ ⟨*expr*⟩ | `(` ⟨*formula*⟩ `)`

⟨*binary-comp*⟩ ::= `<` | `<=` | `>=` | `>` | `=` | `!=`

⟨*distribution*⟩ ::= ⟨*rate-expr*⟩ | `Exp (` ⟨*rate-expr*⟩ `)` | `W (` ⟨*rate-expr*⟩ `,` ⟨*rate-expr*⟩ `)`

| `L (` ⟨*rate-expr*⟩ `,` ⟨*rate-expr*⟩ `)` | `U (` ⟨*rate-expr*⟩ `,` ⟨*rate-expr*⟩ `)`

⟨*update*⟩ ::= ⟨*pname*⟩ = ⟨*expr*⟩ | ⟨*update*⟩ `&` ⟨*update*⟩ | `(` ⟨*update*⟩ `)`

⟨*expr*⟩ ::= ⟨*integer*⟩ | ⟨*name*⟩ | ⟨*expr*⟩ ⟨*binary-op*⟩ ⟨*expr*⟩ | `(` ⟨*expr*⟩ `)`

⟨*binary-op*⟩ ::= `+` | `-` | `*`

⟨*rate-expr*⟩ ::= ⟨*integer*⟩ | ⟨*name*⟩ | ⟨*rate-expr*⟩ ⟨*rate-op*⟩ ⟨*rate-expr*⟩ | `(` ⟨*rate-expr*⟩ `)`

⟨*rate-op*⟩ ::= `*` | `/`

⟨*property*⟩ ::= `true` | `false` | `P` ⟨*pr-comp*⟩ ⟨*probability*⟩ `[` ⟨*path-formula*⟩ `]`

| ⟨*property*⟩ ⟨*logic-op*⟩ ⟨*property*⟩ | `!` ⟨*property*⟩ | ⟨*expr*⟩ | `(` ⟨*property*⟩ `)`

⟨*pr-comp*⟩ ::= `<` | `<=` | `>=` | `>`

⟨*logic-op*⟩ ::= `=>` | `&` | `|`

⟨*path-formula*⟩ ::= ⟨*property*⟩ `U` ⟨*property*⟩ | `X` ⟨*property*⟩

| ⟨*property*⟩ `U <=` ⟨*number*⟩ ⟨*property*⟩

| ⟨*property*⟩ `U [` ⟨*number*⟩ `,` ⟨*number*⟩ `]` ⟨*property*⟩

| `X <=` ⟨*number*⟩ ⟨*property*⟩ | `X [` ⟨*number*⟩ `,` ⟨*number*⟩ `]` ⟨*property*⟩

Appendix B

PPDDL+: An Extension to PDDL for Modeling Stochastic Decision Processes

PDDL (Ghallab et al. 1998; McDermott 2000; Fox and Long 2003) is an established formalism for expressing deterministic planning domains and problems. We present PPDDL+, based on PDDL extensions proposed by Younes (2003) and PPDDL (Younes and Littman 2004). The latter was developed for the probabilistic track of the 2004 International Planning Competition. PPDDL+ extends PPDDL with facilities for modeling actions and events with delayed effects.

B.1 Delayed Actions, Reward Rates, and UTSL Goals

In PPDDL, time is measured in discrete steps, with each time step corresponding to the execution of an action. Rewards are associated with state transitions. This is sufficient for modeling discrete-time MDPs, but not continuous-time MDPs or GSMDPs. PPDDL+ introduces delayed actions for this purpose.

A delayed action defines a transition probability matrix P_a and a reward vector R_a in the same way as a regular PPDDL action. P_a and R_a can be computed from the effect formula for a as described by Younes and Littman (2004). $P_a(i, j)$ is the probability of transitioning to state j when a triggers in state i and $R_a(i)$ is the *expected* reward for a state transition caused by a in i. A positive distribution G_a is also associated with each delayed action a. Let $F_a(t)$ be the cumulative distribution function of G_a. If a becomes enabled at time t_0 and remains continuously enabled until a triggers, then $F_a(t-t_0)$ is the probability that the triggering of a occurs in the interval $(t_0, t]$. In addition to delayed actions, PPDDL+ supports delayed events, which have the same semantics as delayed actions except that they cannot be controlled by a decision maker.

With delayed actions and events, we are quantitatively measuring the time that is spent in a state before a state transition occurs. Therefore, PPDDL+ permits the specification of state-dependent reward rates in problem definitions. The PPDDL+ statement `(:reward-rate` ϕ k`)` specifies that a reward of k is awarded for every time unit that is spent in a state satisfying the formula ϕ.

A final extension of PPDDL facilitates the specification of temporally extended goals in the form of UTSL goal conditions. The statement `(:pctl-goal (pr 0.9 (until 100` Φ Ψ`)))`, for example, corresponds to the UTSL formula $\mathcal{P}_{\geq 0.9}\big[\Phi\ \mathcal{U}^{[0,100]}\ \Psi\big]$. We can use this language feature to express the plan objective $\mathcal{P}_{\geq\theta}[\Diamond\ \phi]$, i.e. that ϕ is eventually achieved with probability at least θ, commonly used by probabilistic planners (cf. Farley 1983; Blythe 1994; Goldman and Boddy 1994b; Kushmerick et al. 1995; Lesh et al. 1998). A regular PDDL goal condition `(:goal` ϕ`)` corresponds to the UTSL formula $\mathcal{P}_{\geq 1}[\Diamond\ \phi]$.

B.2 BNF Grammar

We provide the full syntax for PPDDL+ using an extended BNF notation with the following conventions:

- Each rule is of the form ⟨*non-terminal*⟩ ::= *expansion*.
- Alternative expansions are separated by a vertical bar ("|").
- A syntactic element surrounded by square brackets ("[" and "]") is optional.
- Expansions and optional syntactic elements with a superscripted requirements flag are available only if the requirements flag is specified for the domain or problem currently being defined. For example, [⟨*types-def*⟩]$^{\texttt{:typing}}$ in the syntax for domain definitions means that ⟨*types-def*⟩ may occur only in domain definitions that include the `:typing` flag in the requirements declaration.
- An asterisk ("*") following a syntactic element *x* means zero or more occurrences of *x*; a plus ("$^+$") following *x* means at least one occurrence of *x*.

- Parameterized non-terminals, for example ⟨*typed list* (*x*)⟩, represent separate rules for each instantiation of the parameter.
- Terminals are written using `typewriter` font.
- The syntax is Lisp-like. In particular this means that case is not significant (e.g. `?x` and `?X` are equivalent), parentheses are an essential part of the syntax and have no semantic meaning in the extended BNF notation, and any number of whitespace characters (space, newline, tab, etc.) may occur between tokens.

B.2.1 Domains

The syntax for domain definitions is the same as for PDDL2.1, except that durative actions have been replaced by delayed actions. Declarations of constants, predicates, and functions are allowed in any order with respect to one another, but they must all come after any type declarations and precede any action declarations. A ⟨*name*⟩ is a string of characters starting with an alphabetic character followed by a possibly empty sequence of alphanumeric characters, hyphens ("-"), and underscore characters ("_"). A ⟨*variable*⟩ is a ⟨*name*⟩ immediately preceded by a question mark ("?"). For example, `in-office` and `ball_2` are names, and `?gripper` is a variable.

⟨*domain*⟩	::=	`( define ( domain` ⟨*name*⟩ `)`
		[⟨*require-def*⟩]
		[⟨*types-def*⟩]`:typing`
		[⟨*constants-def*⟩]
		[⟨*predicates-def*⟩]
		[⟨*functions-def*⟩]`:fluents`
		⟨*structure-def*⟩* `)`
⟨*require-def*⟩	::=	`( :requirements` ⟨*require-key*⟩* `)`
⟨*require-key*⟩	::=	*See Section B.2.4*
⟨*types-def*⟩	::=	`( :types` ⟨*typed list* (*name*)⟩ `)`
⟨*constants-def*⟩	::=	`( :constants` ⟨*typed list* (*name*)⟩ `)`
⟨*predicates-def*⟩	::=	`( :predicates` ⟨*atomic formula skeleton*⟩* `)`

⟨*atomic formula skeleton*⟩ ::= `(` ⟨*predicate*⟩ ⟨*typed list* (*variable*)⟩ `)`

⟨*predicate*⟩ ::= ⟨*name*⟩

⟨*functions-def*⟩ ::= `( :functions` ⟨*function typed list* (*function skeleton*)⟩ `)`

⟨*function skeleton*⟩ ::= `(` ⟨*function symbol*⟩ ⟨*typed list* (*variable*)⟩ `)`

⟨*function symbol*⟩ ::= ⟨*name*⟩

⟨*structure-def*⟩ ::= *See Section B.2.2*

⟨*typed list* (*x*)⟩ ::= ⟨*x*⟩* $|^{\texttt{:typing}}$ ⟨*x*⟩$^+$ – ⟨*type*⟩ ⟨*typed list* (*x*)⟩

⟨*type*⟩ ::= `( either` ⟨*primitive type*⟩$^+$ `)` | ⟨*primitive type*⟩

⟨*primitive type*⟩ ::= ⟨*name*⟩

⟨*function typed list* (*x*)⟩ ::= ⟨*x*⟩* $|^{\texttt{:typing}}$ ⟨*x*⟩$^+$ – ⟨*function type*⟩ ⟨*function typed list* (*x*)⟩

⟨*function type*⟩ ::= `number`

B.2.2 Actions

Action definitions and goal descriptions have the same syntax as in PDDL2.1, with the addition of delayed actions and events. A ⟨*number*⟩ is a sequence of numeric characters, possibly with a single decimal point ("`.`") at any position in the sequence. Negative numbers are written as `(-` ⟨*number*⟩`)`, i.e. is using negation.

⟨*structure-def*⟩ ::= ⟨*action-def*⟩
 $|^{\texttt{:delayed-actions}}$ ⟨*delayed-action-def*⟩
 $|^{\texttt{:exogenous-events}}$ ⟨*delayed-event-def*⟩

⟨*action-def*⟩ ::= `( :action` ⟨*name*⟩
 [`:parameters (` ⟨*typed list* (*variable*)⟩ `)`]
 [`:precondition` ⟨*GD*⟩]
 [`:effect` ⟨*effect*⟩] `)`

⟨*delayed-action-def*⟩ ::= `( :delayed-action` ⟨*name*⟩
 [`:parameters (` ⟨*typed list* (*variable*)⟩ `)`]
 `:delay` ⟨*delay-distribution*⟩
 [`:condition` ⟨*GD*⟩]
 [`:effect` ⟨*effect*⟩] `)`

⟨*delayed-event-def*⟩ ::= (:delayed-event ⟨*name*⟩
[:parameters (⟨*typed list* (*variable*)⟩)]
:delay ⟨*delay-distribution*⟩
[:condition ⟨*GD*⟩]
[:effect ⟨*effect*⟩])

⟨*GD*⟩ ::= ⟨*atomic formula* (*term*)⟩ | (and ⟨*GD*⟩*)
$\mid^{\texttt{:equality}}$ (= ⟨*term*⟩ ⟨*term*⟩)
$\mid^{\texttt{:equality}}$ (not (= ⟨*term*⟩ ⟨*term*⟩))
$\mid^{\texttt{:negative-preconditions}}$ (not ⟨*atomic formula* (*term*)⟩)
$\mid^{\texttt{:disjunctive-preconditions}}$ (not ⟨*GD*⟩)
$\mid^{\texttt{:disjunctive-preconditions}}$ (or ⟨*GD*⟩*)
$\mid^{\texttt{:disjunctive-preconditions}}$ (imply ⟨*GD*⟩ ⟨*GD*⟩)
$\mid^{\texttt{:existential-preconditions}}$ (exists (⟨*typed list* (*variable*)⟩) ⟨*GD*⟩)
$\mid^{\texttt{:universal-preconditions}}$ (forall (⟨*typed list* (*variable*)⟩) ⟨*GD*⟩)
$\mid^{\texttt{:fluents}}$ ⟨*f-comp*⟩

⟨*atomic formula* (*x*)⟩ ::= (⟨*predicate*⟩ ⟨*x*⟩*) | ⟨*predicate*⟩

⟨*term*⟩ ::= ⟨*name*⟩ | ⟨*variable*⟩

⟨*f-comp*⟩ ::= (⟨*binary-comp*⟩ ⟨*f-expr*⟩ ⟨*f-expr*⟩)

⟨*binary-comp*⟩ ::= < | <= | = | >= | >

⟨*f-expr*⟩ ::= ⟨*number*⟩ | ⟨*f-head* (*term*)⟩
| (⟨*binary-op*⟩ ⟨*f-expr*⟩ ⟨*f-expr*⟩) | (− ⟨*f-expr*⟩)

⟨*f-head* (*x*)⟩ ::= (⟨*function symbol*⟩ ⟨*x*⟩*) | ⟨*function symbol*⟩

⟨*binary-op*⟩ ::= + | − | * | /

The syntax for effects has been extended to allow for probabilistic effects, which can be arbitrarily interleaved with conditional effects and universal quantification. A ⟨*probability*⟩ is a ⟨*number*⟩ with a value in the interval $[0,1]$. Reward updates are limited to constant increments and decrements.

⟨*effect*⟩ ::= ⟨*p-effect*⟩ | (and ⟨*effect*⟩*)
$\mid^{\texttt{:conditional-effects}}$ (forall (⟨*typed list* (*variable*)⟩) ⟨*effect*⟩)

$|^{\texttt{:conditional-effects}}$ (when ⟨*GD*⟩ ⟨*effect*⟩)
$|^{\texttt{:probabilistic-effects}}$ (probabilistic ⟨*prob-effect*⟩$^{+}$)
⟨*p-effect*⟩ ::= ⟨*atomic formula (term)*⟩ | (not ⟨*atomic formula* (*term*)⟩)
$|^{\texttt{:fluents}}$ (⟨*assign-op*⟩ ⟨*f-head* (*term*)⟩ ⟨*f-expr*⟩)
$|^{\texttt{:rewards}}$ (⟨*additive-op*⟩ ⟨*reward fluent*⟩ ⟨*f-expr*⟩)
⟨*prob-effect*⟩ ::= ⟨*probability*⟩ ⟨*effect*⟩
⟨*assign-op*⟩ ::= assign | scale-up | scale-down | ⟨*additive-op*⟩
⟨*additive-op*⟩ ::= increase | decrease
⟨*reward fluent*⟩ ::= (reward) | reward

Five families of parametric distributions are supported by PPDDL+. A delay distribution that is simply a constant expression corresponds to a deterministic distribution. Implementations may not support all the distributions, and should report an error if they encounter an unsupported distribution in a domain definition. For example, a planning system for continuous-time MDPs would support only the one-parameter exponential distribution. Furthermore, support for deterministic distributions should be implemented with care. If two events with deterministic delay can be enabled simultaneously, there could be a non-zero probability that both events trigger at the same time.

⟨*delay-distribution*⟩ ::= ⟨*const-expr*⟩
| (exponential ⟨*const-expr*⟩ [⟨*const-expr*⟩])
| (weibull ⟨*const-expr*⟩ [⟨*const-expr*⟩] [⟨*const-expr*⟩])
| (lognormal ⟨*const-expr*⟩ ⟨*const-expr*⟩)
| (uniform ⟨*const-expr*⟩ ⟨*const-expr*⟩)
⟨*const-expr*⟩ ::= ⟨*number*⟩
| (⟨*binary-op*⟩ ⟨*const-expr*⟩ ⟨*const-expr*⟩) | (- ⟨*const-expr*⟩)

B.2.3 Problems

The syntax for problem definitions includes the extensions of PPDDL to PDDL2.1 that allow for the specification of a probability distribution over initial states, and also permit the association of a one-time reward with entering a goal state. In PPDDL+, it is also possible to specify a goal condition as a UTSL formula.

⟨*problem*⟩	::= `(define (problem` ⟨*name*⟩ `)`
	`(:domain` ⟨*name*⟩ `)`
	[⟨*require-def*⟩]
	[⟨*objects-def*⟩]
	[⟨*init*⟩]
	[⟨*reward-rate-spec*⟩]$^{\texttt{:rewards}}$
	⟨*goal*⟩ `)`
⟨*objects-def*⟩	::= `(:objects` ⟨*typed list* (*name*)⟩ `)`
⟨*init*⟩	::= `(:init` ⟨*init-el*⟩* `)`
⟨*init-el*⟩	::= ⟨*p-init-el*⟩ $\mid^{\texttt{:probabilistic-effects}}$ `(probabilistic` ⟨*prob-init-el*⟩* `)`
⟨*p-init-el*⟩	::= ⟨*atomic formula* (*name*)⟩ $\mid^{\texttt{:fluents}}$ `(=` ⟨*f-head* (*name*)⟩ ⟨*number*⟩ `)`
⟨*prob-init-el*⟩	::= ⟨*probability*⟩ ⟨*a-init-el*⟩
⟨*a-init-el*⟩	::= ⟨*p-init-el*⟩ | `(and` ⟨*p-init-el*⟩* `)`
⟨*reward-rate-spec*⟩	::= `(:reward-rate` ⟨*state-reward*⟩* `)`
⟨*state-reward*⟩	::= ⟨*GD*⟩ ⟨*const-expr*⟩
⟨*goal*⟩	::= ⟨*goal-spec*⟩ [⟨*metric-spec*⟩] | ⟨*metric-spec*⟩
⟨*goal-spec*⟩	::= `(:goal` ⟨*GD*⟩ `)` [`(:goal-reward` ⟨*ground-f-expr*⟩ `)`]$^{\texttt{:rewards}}$
	$\mid^{\texttt{:utsl-goals}}$ `(:utsl-goal` ⟨*pctl-formula*⟩ `)`
⟨*metric-spec*⟩	::= `(:metric` ⟨*optimization*⟩ ⟨*ground-f-expr*⟩ `)`
⟨*optimization*⟩	::= `minimize` | `maximize`
⟨*ground-f-expr*⟩	::= ⟨*number*⟩ | ⟨*f-head* (*name*)⟩
	| `(` ⟨*binary-op*⟩ ⟨*ground-f-expr*⟩ ⟨*ground-f-expr*⟩ `)`
	| `(-` ⟨*ground-f-expr*⟩ `)`
	| `(total-time)` | `total-time`
	| `(goal-achieved)` | `goal-achieved`
	$\mid^{\texttt{:rewards}}$ ⟨*reward fluent*⟩
⟨*pctl-formula*⟩	::= `(pr` ⟨*probability*⟩ ⟨*path-formula*⟩ `)`
	| `(not (pr` ⟨*probability*⟩ ⟨*path-formula*⟩ `))`

$\langle$*path-formula*$\rangle$ ::= (`until` [$\langle$*number*$\rangle$] [$\langle$*number*$\rangle$] $\langle$*GD*$\rangle$ $\langle$*GD*$\rangle$)
| (`weak-until` [$\langle$*number*$\rangle$] [$\langle$*number*$\rangle$] $\langle$*GD*$\rangle$ $\langle$*GD*$\rangle$)
| (`eventually` [$\langle$*number*$\rangle$] [$\langle$*number*$\rangle$] $\langle$*GD*$\rangle$ $\langle$*GD*$\rangle$)
| (`continuously` [$\langle$*number*$\rangle$] [$\langle$*number*$\rangle$] $\langle$*GD*$\rangle$ $\langle$*GD*$\rangle$)

B.2.4 Requirements

Below is a table of all requirements in PPDDL+. Some requirements imply others; some are abbreviations for common sets of requirements. If a domain stipulates no requirements, it is assumed to declare a requirement for `:strips`.

Requirement	**Description**
`:strips`	Basic STRIPS-style adds and deletes
`:typing`	Allow type names in declarations of variables
`:equality`	Support = as built-in predicate
`:negative-preconditions`	Allow negated atoms in goal descriptions
`:disjunctive-preconditions`	Allow disjunctive goal descriptions
`:existential-preconditions`	Allow `exists` in goal descriptions
`:universal-preconditions`	Allow `forall` in goal descriptions
`:quantified-preconditions`	= `:existential-preconditions` + `:universal-preconditions`
`:conditional-effects`	Allow `when` and `forall` in action effects
`:probabilistic-effects`	Allow `probabilistic` in action effects
`:rewards`	Allow reward fluent in action effects and optimization metric
`:fluents`	Allow numeric state variables
`:utsl-goals`	Allow UTSL goal conditions
`:delayed-actions`	Allow actions with random delay
`:exogenous-events`	Allow uncontrollable events with random delay
`:adl`	= `:strips` + `:typing` + `:equality`

	+ `:negative-preconditions`
	+ `:disjunctive-preconditions`
	+ `:quantified-preconditions`
	+ `:conditional-effects`
`:mdp`	= `:probabilistic-effects` + `:rewards`
`:gsmdp`	= `:mdp` + `:delayed-actions`
	+ `:exogenous-events`

Bibliography

Agresti, Alan and Brent A. Coull. 1998. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician* 52, no. 2: 119–126.

Aldous, David and Larry Shepp. 1987. The least variable phase type distribution is Erlang. *Communications in Statistics—Stochastic Models* 3, no. 3: 467–473.

Altiok, Tayfur. 1985. On the phase-type approximations of general distributions. *IIE Transactions* 17, no. 2: 110–116.

Alur, Rajeev, Costas Courcoubetis, and David L. Dill. 1990. Model-checking for real-time systems. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, 414–425, Philadelphia, Pennsylvania. IEEE Computer Society.

———. 1991. Model-checking for probabilistic real-time systems. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, edited by J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, vol. 510 of *Lecture Notes in Computer Science*, 115–126, Madrid, Spain. Springer.

———. 1993. Model-checking in dense real-time. *Information and Computation* 104, no. 1: 2–34.

Alur, Rajeev and David L. Dill. 1994. A theory of timed automata. *Theoretical Computer Science* 126, no. 2: 183–235.

Alur, Rajeev and Thomas A. Henzinger. 1992. Logics and models of real time: A survey. In *Proceedings of the REX Workshop on Real-Time: Theory in Practice*, edited by J. W. de Bakker et al., vol. 600 of *Lecture Notes in Computer Science*, 74–106. Berlin: Springer.

———. 1999. Reactive modules. *Formal Methods in System Design* 15, no. 1: 7–48.

Anderson, T. W. and Milton Friedman. 1960. A limitation of the optimum property of the sequential probability ratio test. In *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, edited by Ingram Olkin et al., 57–69. Stanford, California: Stanford University Press.

Asmussen, Søren, Olle Nerman, and Marita Olsson. 1996. Fitting phase-type distributions via the EM algorithm. *Scandinavian Journal of Statistics* 23, no. 4: 419–441.

Atkins, Ella M., Edmund H. Durfee, and Kang G. Shin. 1996. Plan development using local probabilistic models. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, edited by Eric Horvitz and Finn V. Jensen, 49–56, Portland, Oregon. Morgan Kaufmann Publishers.

Aziz, Adnan, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. 1996. Verifying continuous time Markov chains. In *Proceedings of the 8th International Conference on Computer Aided Verification*, edited by Rajeev Alur and Thomas A. Henzinger, vol. 1102 of *Lecture Notes in Computer Science*, 269–276, New Brunswick, New Jersey. Springer.

———. 2000. Model-checking continuous-time Markov chains. *ACM Transactions on Computational Logic* 1, no. 1: 162–170.

Bahar, R. Iris, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. 1993. Algebraic decision diagrams and their applications. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, 188–191, Santa Clara, California. IEEE Computer Society Press.

Baier, Christel, Edmund M. Clarke, Vasiliki Hartonas-Garmhausen, Marta Kwiatkowska, and Mark Ryan. 1997. Symbolic model checking for probabilistic processes. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, edited by Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, vol. 1256 of *Lecture Notes in Computer Science*, 430–440, Bologna, Italy. Springer.

Baier, Christel, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. 2000. Model checking continuous-time Markov chains by transient analysis. In *Proceedings of the 12th International Conference on Computer Aided Verification*, edited by E. Allen Emerson and A. Prasad Sistla, vol. 1855 of *Lecture Notes in Computer Science*, 358–372, Chicago, Illinois. Springer.

———. 2003. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering* 29, no. 6: 524–541.

Baier, Christel, Joost-Pieter Katoen, and Holger Hermanns. 1999. Approximate symbolic model checking of continuous-time Markov chains. In *Proceedings of the 10th International Conference on Concurrency Theory*, edited by Jos C. M. Baeten and Sjouke Mauw, vol. 1664 of *Lecture Notes in Computer Science*, 146–161, Eindhoven, the Netherlands. Springer.

Balemi, S., G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin. 1993. Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control* 38, no. 7: 1040–1059.

Bartlett, M. S. 1966. *An Introduction to Stochastic Processes with Special Reference to Methods and Applications*. 2nd ed. London: Cambridge University Press.

Bellman, Richard. 1957. *Dynamic Programming*. Princeton, New Jersey: Princeton University Press.

Bellman, Richard, Robert Kalaba, and Bella Kotkin. 1963. Polynomial approximation—a new computational technique in dynamic programming: Allocation processes. *Mathematics of Computation* 17, no. 82: 155–161.

Ben-Ari, Mordechai, Zohar Manna, and Amir Pnueli. 1981. The temporal logic of branching time. In *Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 164–176, Williamsburg, Virginia. Association for Computing Machinery.

Ben-Ari, Mordechai, Amir Pnueli, and Zohar Manna. 1983. The temporal logic of braching time. *Acta Informatica* 20, no. 3: 207–226.

Bernstein, Arthur and Paul K. Harter, Jr. 1981. Proving real-time properties of programs with temporal logic. In *Proceedings of the Eighth ACM Symposium on Operating Systems Principles*, 1–11, Pacific Grove, California. ACM SIGOPS.

Bianco, Andrea and Luca de Alfaro. 1995. Model checking of probabilistic and nondeterministic systems. In *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, edited by P. S. Thiagarajan, vol. 1026 of *Lecture Notes in Computer Science*, 499–513, Bangalore, India. Springer.

Blythe, Jim. 1994. Planning with external events. In *Proceedings of the Tenth Conference on Uncer-*

tainty in Artificial Intelligence, edited by Ramon Lopez de Mantaras and David Poole, 94–101, Seattle, Washington. Morgan Kaufmann Publishers.

Bobbio, Andrea and Aldo Cumani. 1992. ML estimation of the parameters of a PH distribution in triangular canonical form. In *Computer Performance Evaluation: Modelling Techniques and Tools*, edited by Gianfranco Balbo and Giuseppe Serazzi, 33–46. Amsterdam: Elsevier.

Bobbio, Andrea, A. Horváth, M. Scrapa, and Miklós Telek. 2003. Acyclic discrete phase type distributions: Properties and a parameter estimation algorithm. *Performance Evaluation* 54, no. 1: 1–32.

Bobbio, Andrea, A. Horváth, and Miklós Telek. 2004. The scale factor: A new degree of freedom in phase-type approximation. *Performance Evaluation* 56, no. 1–4: 121–144.

Bonet, Blai, Gábor Loerincs, and Héctor Geffner. 1997. A robust and fast action selection mechanism for planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 714–719, Providence, Rhode Island. AAAI Press.

Boutilier, Craig, Thomas Dean, and Steve Hanks. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11: 1–94.

Boutilier, Craig and Richard Dearden. 1994. Using abstractions for decision-theoretic planning with time constraints. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1016–1022, Seattle, Washington. AAAI Press.

Boutilier, Craig, Richard Dearden, and Moisés Goldszmidt. 1995. Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, edited by Chris S. Mellish, 1104–1111, Montreal, Canada. Morgan Kaufmann Publishers.

Boyan, Justin A. and Michael L. Littman. 2001. Exact solutions to time-dependent MDPs. In *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*, edited by Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, 1026–1032. Cambridge, Massachusetts: The MIT Press.

Bratley, Paul, Bennett L. Fox, and Linus E. Schrage. 1987. *A Guide to Simulation*. 2nd ed. Berlin: Springer.

Brown, Lawrence D., T. Tony Cai, and Anirban DasGupta. 2001. Interval estimation for a binomial proportion. *Statistical Science* 16, no. 2: 101–133.

Bryant, Randal E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* C-35, no. 8: 677–691.

Buchholz, Peter. 1998. A new approach combining simulation and randomization for the analysis of large continuous time Markov chains. *ACM Transactions on Modeling and Computer Simulation* 8, no. 2: 194–222.

Buchholz, Peter, Joost-Pieter Katoen, Peter Kemper, and Carsten Tepper. 2003. Model-checking large structured Markov chains. *The Journal of Logic and Algebraic Programming* 56, no. 1–2: 69–97.

Cantaluppi, Laurent. 1984. Optimality of piecewise-constant policies in semi-Markov decision chains. *SIAM Journal on Control and Optimization* 22, no. 5: 723–739.

Chitgopekar, S. S. 1969. Continuous time Markovian sequential control processes. *SIAM Journal on Control* 7, no. 3: 367–389.

Chow, Y. S. and Herbert Robbins. 1965. On the asymptotic theory of fixed-width sequential confidence intervals for the mean. *Annals of Mathematical Statistics* 36, no. 3: 457–462.

Cimatti, Alessandro, Marco Roveri, and Paolo Traverso. 1998. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 875–881, Madison, Wisconsin. AAAI Press.

Çinlar, Erhan. 1975. *Introduction to Stochastic Processes*. Englewood Cliffs, New Jersey: Prentice-Hall.

Clarke, Edmund M. and E. Allen Emerson. 1982. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the 1981 Workshop on Logics of Programs*, edited by Dexter Kozen, vol. 131 of *Lecture Notes in Computer Science*, 52–71. Berlin: Springer.

Clarke, Edmund M., E. Allen Emerson, and A. Prasad Sistla. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8, no. 2: 244–263.

Clarke, Edmund M., K. L. McMillan, X. Zhao, and M. Fujita. 1993. Spectral transforms for large Boolean functions with applications to technology mapping. In *Proceedings of the 30th International Conference on Design Automation*, 54–60, Dallas, Texas. ACM Press.

Courcoubetis, Costas and Mihalis Yannakakis. 1995. The complexity of probabilistic verification. *Journal of the Association for Computing Machinery* 42, no. 4: 857–907.

Cox, D. R. 1955. A use of complex probabilities in the theory of stochastic processes. *Proceedings of the Cambridge Philosophical Society* 51, no. 2: 313–319.

De Moivre, A. 1738. A method of approximating the sum of the terms of the binomial $(a + b)^n$ expanded into a series, from whence are deducted some practical rules to estimate the degree of assent which is to be given to experiments. In *The Doctrine of Chances: or, A Method of Calculating the Probabilities of Events in Play*, 235–243. 2nd ed. London: H. Woodfall.

Dean, Thomas and Keiji Kanazawa. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5, no. 3: 142–150.

Dearden, Richard and Craig Boutilier. 1997. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence* 89, no. 1–2: 219–283.

Dechter, Rina, Itay Meiri, and Judea Pearl. 1991. Temporal constraint networks. *Artificial Intelligence* 49, no. 1–3: 61–95.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39, no. 1: 1–38.

Devroye, Luc. 1986. *Non-Uniform Random Variate Generation*. New York: Springer.

Dirac, P. A. M. 1927. The physical interpretation of the quantum dynamics. *Proceedings of the Royal Society of London. Series A, Containing Papers of Mathematrical or Physical Charachter* 113, no. 765: 621–641.

Dodge, H. F. and H. G. Romig. 1929. A method of sampling inspection. *The Bell System Technical Journal* 8: 613–631.

Doob, J. L. 1942. What is a stochastic process? *The American Mathematical Monthly* 49, no. 10: 648–653.

———. 1953. *Stochastic Processes*. New York: John Wiley & Sons.

Draper, Denise, Steve Hanks, and Daniel S. Weld. 1994. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, edited by Kristian Hammond, 31–36, Chicago, Illinois. AAAI Press.

Drummond, Mark and John Bresina. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 138–144, Boston. AAAI Press.

Duncan, Acheson J. 1974. *Quality Control and Industrial Statistics*. 4th ed. Homewood, Illinois: Richard D. Irwin.

Eckhardt, Roger. 1987. Stan Ulam, John von Neumann, and the Monte Carlo method. *Los Alamos Science*, no. 15: 131–137.

Emerson, E. Allen. 1990. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, edited by Jan van Leeuwen, vol. B, 995–1072. Amsterdam: Elsevier.

Emerson, E. Allen, A. K. Mok, A. Prasad Sistla, and Jai Srinivasan. 1990. Quantitative temporal reasoning. In *Proceedings of the 2nd International Conference on Computer Aided Verification*, edited by Edmund M. Clarke and R. P. Kurshan, vol. 531 of *Lecture Notes in Computer Science*, 136–145, New Brunswick, New Jersey. Springer.

———. 1992. Quantitative temporal reasoning. *Real-Time Systems* 4, no. 4: 331–352.

Erlang, A. K. 1917. Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges. *The Post Office Electrical Engineers' Journal* 10: 189–197.

Farley, Arthur M. 1983. A probabilistic model for uncertain problem solving. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13, no. 4: 568–579.

Feller, William. 1957. *An Introduction to Probability Theory and Its Applications*, vol. I. 2nd ed. New York: John Wiley & Sons.

Feng, Zhengzhu, Richard Dearden, Nicolas Meuleau, and Richard Washington. 2004. Dynamic programming for structured continuous Markov decision processes. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, edited by Max Chickering and Joseph Halpern, 154–161, Banff, Canada. AUAI Press.

Ferrenberg, Alan M., D. P. Landau, and Y. Joanna Wong. 1992. Monte Carlo simulations: Hidden errors from "good" random number generators. *Physical Review Letters* 69, no. 23: 3382–3384.

Fox, Bennett L. and Peter W. Glynn. 1988. Computing Poisson probabilities. *Communications of the ACM* 31, no. 4: 440–445.

Fox, Maria and Derek Long. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20: 61–124.

Freeman, David and Lionel Weiss. 1964. Sampling plans which approximately minimize the maximum expected sample size. *Journal of the American Statistical Association* 59, no. 305: 67–88.

Fujino, Yoritake. 1980. Approximate binomial confidence limits. *Biometrika* 67, no. 3: 677–681.

Fujita, M., P. C. McGeer, and J. C.-Y. Yang. 1997. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design* 10, no. 2/3: 149–169.

Ghallab, Malik, Adele E. Howe, Craig A. Knoblock, Drew McDermott, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David Wilkins. 1998. PDDL—the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, Connecticut.

Ginsberg, Matthew L. 1989. Universal planning: An (almost) universally bad idea. *AI Magazine* 10, no. 4: 40–44.

Glynn, Peter W. 1989. A GSMP formalism for discrete event systems. *Proceedings of the IEEE* 77, no. 1: 14–23.

Goldman, Robert P. and Mark S. Boddy. 1994a. Conditional linear planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, edited by Kristian Hammond, 80–85, Chicago, Illinois. AAAI Press.

———. 1994b. Epsilon-safe planning. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, edited by Ramon Lopez de Mantaras and David Poole, 253–261, Seattle, Washington. Morgan Kaufmann Publishers.

Goldman, Robert P., Michael J. S. Pelican, and David J. Musliner. 2004. Guiding planner backjumping using verifier traces. In *Proceedings of the Fourteenth International Conference on Automated Planning* *and Scheduling*, edited by Shlomo Zilberstein, Jana Koehler, and Sven Koenig, 279–286, Whistler, Canada. AAAI Press.

Gonnet, Gaston H. 1976. Heaps applied to event driven mechanisms. *Communications of the ACM* 19, no. 7: 417–418.

Gordon, Geoffrey J. 1995. Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning*, edited by Armand Prieditis and Stuart Russell, 261–268, Tahoe City, California. Morgan Kaufmann Publishers.

Grassmann, W. K. 1977. Transient solutions in Markovian queueing systems. *Computers & Operations Research* 4, no. 1: 47–53.

Gross, Donald and Douglas R. Miller. 1984. The randomization technique as a modeling tool and solution procedure for transient markov processes. *Operations Research* 32, no. 2: 343–361.

Grosu, R. and Scott A. Smolka. 2004. Quantitative model checking. In *Proceedings of the 1st International Symposium on Leveraging Applications of Formal Methods*, Paphos, Cyprus.

Grubbs, Frank E. 1949. On designing single sampling inspection plans. *Annals of Mathematical Statistics* 20, no. 2: 242–256.

Guestrin, Carlos, Daphne Koller, and Ronald Parr. 2002. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems 14: Proceedings of the 2001 Conference*, edited by Thomas G. Ditterich, Suzanna Becker, and Zoubin Ghahramani, 1523–1530. Cambridge, Massachusetts: The MIT Press.

Guestrin, Carlos, Daphne Koller, Ronald Parr, and Shobha Venkataraman. 2003. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research* 19: 399–468.

Hall, Peter. 1982. Improving the normal approximation when constructing one-sided confidence intervals for binomial or Poisson parameters. *Biometrika* 69, no. 3: 647–652.

Halmos, Paul R. 1950. *Measure Theory*. New York: Van Nostrand Reinhold Company.

Hansson, Hans and Bengt Jonsson. 1989. A framework for reasoning about time and reliability. In *Proceedings of the Real-Time Systems Symposium*, 102–111, Santa Monica, California. IEEE Computer Society Press.

———. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6, no. 5: 512–535.

Hart, Sergiu and Micha Sharir. 1984. Probabilistic temporal logics for finite and bounded models. In *Proceedings of the Sixteenth ACM Symposium on Theory of Computing*, 1–13, Washington, D.C. ACM SIGACT.

Hart, Sergiu, Micha Sharir, and Amir Pnueli. 1983. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems* 5, no. 3: 356–380.

Hastings, Jr., Cecil. 1955. *Approximations for Digital Computers*. Princeton, New Jersey: Princeton University Press.

Heidelberger, Philip. 1995. Fast simulation of rare events in queueing and reliability models. *ACM Transactions on Modeling and Computer Simulation* 5, no. 1: 43–85.

Hermanns, Holger, Joost-Pieter Katoen, Joachim Meyer-Kayser, and Markus Siegle. 2000. A Markov chain model checker. In *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, edited by Susanne Graf and Michael Schwartzbach, vol. 1785 of *Lecture Notes in Computer Science*, 347–362, Berlin. Springer.

Hermanns, Holger, Joachim Meyer-Kayser, and Markus Siegle. 1999. Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains. In *Proceedings of the 3rd International Workshop on the Numerical Solution of Markov Chains*, edited by B Plateau, William J. Stewart, and M. Silva, 188–207, Zaragoza, Spain. Prensas Universitarias de Zaragoza.

Hoel, Paul G., Sidney C. Port, and Charles J. Stone. 1972. *Introduction to Stochastic Processes*. Boston: Houghton Mifflin Company.

Hoey, Jesse, Robert St-Aubin, Alan Hu, and Craig Boutilier. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, edited by Kathryn B. Laskey and Henri Prade, 279–288, Stockholm, Sweden. Morgan Kaufmann Publishers.

Hogg, Robert V. and Allen T. Craig. 1978. *Introduction to Mathematical Statistics*. 4th ed. New York: Macmillan Publishing Co.

Howard, Ronald A. 1960. *Dynamic Programming and Markov Processes*. New York: John Wiley & Sons.

———. 1963. Semi-Markov decision processes. *Bulletin de l'Institut International de Statistique* 40, no. 2: 625–652.

———. 1971a. *Dynamic Probabilistic Systems*, vol. I: Markov Models. New York: John Wiley & Sons.

———. 1971b. *Dynamic Probabilistic Systems*, vol. II: Semi-Markov and Decision Processes. New York: John Wiley & Sons.

Ibe, Oliver C. and Kishor S. Trivedi. 1990. Stochastic Petri net models of polling systems. *IEEE Journal on Selected Areas in Communications* 8, no. 9: 1649–1657.

Infante López, Gabriel G., Holger Hermanns, and Joost-Pieter Katoen. 2001. Beyond memoryless distributions: Model checking semi-Markov chains. In *Proceedings of the 1st Joint International PAPM-PROBMIV Workshop*, edited by Luca de Alfaro and Stephen Gilmore, vol. 2165 of *Lecture Notes in Computer Science*, 57–70, Aachen, Germany. Springer.

Jensen, Arne. 1953. Markoff chains as an aid in the study of Markoff processes. *Skandinavisk Aktuarietidskrift* 36: 87–91.

Jensen, Rune M. and Manuela M. Veloso. 2000. OBDD-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research* 13: 189–226.

Jensen, Rune M., Manuela M. Veloso, and Randal E. Bryant. 2004. Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, edited by Shlomo Zilberstein, Jana Koehler, and Sven Koenig, 335–344, Whistler, Canada. AAAI Press.

Johnson, Mary A. and Michael R. Taaffe. 1989. Matching moments to phase distributions: Mixtures of Erlang distributions of common order. *Communications in Statistics—Stochastic Models* 5, no. 4: 711–743.

———. 1990. Matching moments to phase distributions: Nonlinear programming approaches. *Communications in Statistics—Stochastic Models* 6, no. 2: 259–281.

Kabanza, Froduald, M. Barbeau, and R. St-Denis. 1997. Planning control rules for reactive agents. *Artificial Intelligence* 95, no. 1: 67–113.

Katoen, Joost-Pieter, Marta Kwiatkowska, Gethin Norman, and David Parker. 2001. Faster and symbolic CTMC model checking. In *Proceedings of the 1st Joint International PAPM-PROBMIV Workshop*, edited by Luca de Alfaro and Stephen Gilmore, vol. 2165 of *Lecture Notes in Computer Science*, 23–38, Aachen, Germany. Springer.

Kiefer, J. and Lionel Weiss. 1957. Some properties of generalized sequential probability ratio tests. *Annals of Mathematical Statistics* 28, no. 1: 57–74.

Koenig, Sven, Richard Goodwin, and Reid G. Simmons. 1995. Robot navigation with Markov models: A framework for path planning and learning with limited computational resources. In *Proceedings of the International Workshop on Reasoning with Uncertainty in Robotics*, edited by Leo Dorst, Michiel van Lambalgen, and Frans Voorbraak, vol. 1093 of *Lecture Notes in Computer Science*, 322–337, Amsterdam. Springer.

Kolmogoroff, A. 1931. Über die analytischen Methoden in der Wahrscheinlichkeitsrechnung. *Mathematische Annalen* 104: 415–458.

Kullback, S. and R. A. Leibler. 1951. On information and sufficiency. *Annals of Mathematical Statistics* 22, no. 1: 79–86.

Kushmerick, Nicholas, Steve Hanks, and Daniel S. Weld. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76, no. 1–2: 239–286.

Kwiatkowska, Marta, Gethin Norman, and António Pacheco. 2002a. Model checking CSL until formulae with random time bounds. In *Proceedings of the 2nd Joint International PAPM-PROBMIV Workshop*, edited by Holger Hermanns and Roberto Segala, vol. 2399 of *Lecture Notes in Computer Science*, 152–168, Copenhagen, Denmark. Springer.

Kwiatkowska, Marta, Gethin Norman, and David Parker. 2002b. Probabilistic symbolic model checking with PRISM: A hybrid approach. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, edited by Joost-Pieter Katoen and Perdita Stevens, vol. 2280 of *Lecture Notes in Computer Science*, 52–66, Grenoble, France. Springer.

———. 2004. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer* 6, no. 2: 128–142.

Kwiatkowska, Marta, Gethin Norman, Roberto Segala, and Jeremy Sproston. 2000. Verifying quantitative properties of continuous probabilistic timed automata. In *Proceedings of the 11th International Conference on Concurrency Theory*, edited by Catuscia Palamidessi, vol. 1877 of *Lecture Notes in Computer Science*, 123–137, State College, Pennsylvania. Springer.

Lai, Tze Leung. 1988. Nearly optimal squential tests of composite hypotheses. *The Annals of Statistics* 16, no. 2: 856–886.

———. 2001. Sequential analysis: Some classical problems and new challenges. *Statistica Sinica* 11, no. 2: 303–408.

Lamport, Leslie. 1980. "Somtime" is sometimes "not never": On the temporal logic of programs. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 174–185, Las Vegas, Nevada. Association for Computing Machinery.

Larson, Harry R. 1966. A nomograph of the cumulative binomial distribution. *Industrial Quality Control* 23: 270–278.

Lassaigne, Richard and Sylvian Peyronnet. 2002. Approximate verification of probabilistic systems. In *Proceedings of the 2nd Joint International PAPM-PROBMIV Workshop*, edited by Holger Hermanns and Roberto Segala, vol. 2399 of *Lecture Notes in Computer Science*, 213–214, Copenhagen, Denmark. Springer.

Lehmann, Daniel and Saharon Shelah. 1982. Reasoning with time and chance. *Information and Control* 53, no. 3: 165–198.

Leland, Will E. and Teunis J. Ott. 1986. Load-balancing heiristics and process behavior. *ACM SIGMETRICS Performance Evaluation Review* 14, no. 1: 54–69.

Lesh, Neal, Nathaniel Martin, and James Allen. 1998. Improving big plans. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 860–867, Madison, Wisconsin. AAAI Press.

Li, Haksun, Ella M. Atkins, Edmund H. Durfee, and Kang G. Shin. 2003. Resource allocation for a limited real-time agent. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, edited by Jeffrey S Rosenschein et al., 1050–1051, Melbourne, Australia. ACM Press.

Lippman, Steven A. 1975. Applying a new device in the optimization of exponential queuing systems. *Operations Research* 23, no. 4: 687–710.

Littman, Michael L., Anthony R. Cassandra, and Leslie Pack Kaelbling. 1995. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, edited by Armand Prieditis and Stuart Russell, 362–370, Tahoe City, California. Morgan Kaufmann Publishers.

Malhotra, Manish, Jogesh K. Muppala, and Kishor S. Trivedi. 1994. Stiffness-tolerant methods for transient analysis of stiff Markov chains. *Microelectronics and Reliability* 34, no. 11: 1825–1841.

Marie, Raymond. 1980. Calculating equilibrium probabilities for $\lambda(n)/C_k/1/N$ queues. In *Proceedings of the 7th IFIP W.G.7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation*, 117–125, Toronto, Canada. ACM SIGMETRICS.

Matsumoto, Makoto and Takuji Nishimura. 1998. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation* 8, no. 1: 3–30.

———. 2000. Dynamic creation of pseudorandom number generators. In *Monte-Carlo and Quasi-Monte Carlo Methods 1998*, edited by Harald Niederreiter and Jerome Spanier, 56–69. Berlin: Springer.

Matthes, Klaus. 1962. Zur Theorie der Bedienungsprozesse. In *Transactions of the Third Prague Conference on Information Theory, Statistical Decision Functions, Random Processes*, edited by Jaroslav Kožešník, 513–528, Liblice, Czechoslovakia. Publishing House of the Czechoslovak Academy of Sciences.

Mausam and Daniel S. Weld. 2004. Solving concurrent Markov decision processes. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 716–722, San Jose, California. AAAI Press.

McCormack, William M. and Robert G. Sargent. 1981. Analysis of future event set algorithms for discrete event simulation. *Communications of the ACM* 24, no. 12: 801–812.

McDermott, Drew. 2000. The 1998 AI planning systems competition. *AI Magazine* 21, no. 2: 35–55.

Metropolis, Nicholas. 1987. The beginning of the Monte Carlo method. *Los Alamos Science*, no. 15: 125–130.

Metropolis, Nicholas and S. M. Ulam. 1949. The Monte Carlo method. *Journal of the American Statistical Association* 44, no. 247: 335–341.

Michie, Donald. 1968. “memo” functions and machine learning. *Nature* 218, no. 5136: 19–22.

Musliner, David J., Edmund H. Durfee, and Kang G. Shin. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74, no. 1: 83–127.

Nádas, Arthur. 1969. An extension of a theorem of Chow and Robbins on sequential confidence intervals for the mean. *Annals of Mathematical Statistics* 40, no. 2: 667–671.

Nelson, Wayne. 1985. Weibull analysis of reliability data with few or no failures. *Journal of Quality Technology* 17, no. 3: 140–146.

Neuts, Marcel F. 1975. Probability distributions of phase type. In *Liber Amicorum Professor emeritus dr. H. Florin*, edited by R. Holvoet, 173–206. Leuven, Belgium: Katholieke Universiteit Leuven.

———. 1981. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. Baltimore, Maryland: Johns Hopkins University Press.

Newcombe, Robert G. 1998. Two-sided confidence intervals for the single proportion: Comparison of seven methods. *Statistics in Medicine* 17, no. 8: 857–872.

Nikovski, Daniel and Matthew Brand. 2003. Decision-theoretic group elevator scheduling. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*, edited by Enrico Giunchiglia, Nicola Muscettola, and Dana S. Nau, 133–142, Trento, Italy. AAAI Press.

Osogami, Takayuki and Mor Harchol-Balter. 2003. A closed-form solution for mapping general distributions to minimal PH distributions. In *Proceedings of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, edited by Peter Kemper and William H. Sanders, vol. 2794 of *Lecture Notes in Computer Science*, 200–217, Urbana, Illinois. Springer.

Parker, David. 2002. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, School of Computer Science, University of Birmingham, Birmingham, United Kingdom.

Peach, Paul and S. B. Littauer. 1946. A note on sampling inspection. *Annals of Mathematical Statistics* 17, no. 1: 81–84.

Pearson, Karl. 1924. Historical note on the origin of the normal curve of errors. *Biometrika* 16, no. 3/4: 402–404.

Peot, Mark A. and David E. Smith. 1992. Conditional nonlinear planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, edited by James Hendler, 189–197, College Park, Maryland. Morgan Kaufmann Publishers.

Pistore, Marco and Paolo Traverso. 2001. Planning as model checking for extended goals in non-deterministic domains. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, edited by Bernhard Nebel, 479–484, Seattle, Washington. Morgan Kaufmann Publishers.

Pnueli, Amir. 1977. The temporal logic of programs. In *Proceedings of the 18th Annual Sumposium on Foundations of Computer Science*, 46–57, Providence, Rhode Island. IEEE Computer Society.

Puterman, Martin L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: John Wiley & Sons.

Puterman, Martin L. and Moon Chirl Shin. 1978. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science* 24, no. 11: 1127–1137.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning* 1, no. 1: 81–106.

Raatikainen, Kimmo E. E. 1995. Simulation-based estimation of proportions. *Management Science* 41, no. 7: 1202–1223.

Reibman, Andrew and Kishor S. Trivedi. 1988. Numerical transient analysis of Markov models. *Computers & Operations Research* 15, no. 1: 19–36.

Reif, John H. 1980. Logics for probabilistic programming. In *Proceedings of the Twelfth ACM Symposium on Theory of Computing*, 8–13, Los Angeles, California. ACM SIGACT.

Rescher, Nicholas and Alasdair Urquhart. 1971. *Temporal Logic*. New York: Springer.

Riley, Patrick and Manuela M. Veloso. 2004. Advice generation from observed execution: Abstract Markov decision process learning. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 631–636, San Jose, California. AAAI Press.

Rintanen, Jussi. 2003. Expressive equivalence of formalism for planning with sensing. In *Proceedings* *of the Thirteenth International Conference on Automated Planning and Scheduling*, edited by Enrico Giunchiglia, Nicola Muscettola, and Dana S. Nau, 185–194, Trento, Italy. AAAI Press.

Rohanimanesh, Khashayar and Sridhar Mahadevan. 2001. Decision-theoretic planning with concurrent temporally extended actions. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, edited by Jack Breese and Daphne Koller, 472–479, Seattle, Washington. Morgan Kaufmann Publishers.

Sauer, C. H. and K. M. Chandy. 1975. Approximate analysis of central server models. *IBM Journal of Research and Development* 19, no. 3: 301–313.

Schoppers, M. J. 1987. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, edited by John McDermott, 1039–1046, Milan, Italy. Morgan Kaufmann Publishers.

Schwarz, Gideon. 1962. Asymptotic shapes of Bayes sequential testing regions. *Annals of Mathematical Statistics* 33, no. 1: 224–236.

Sen, Koushik, Mahesh Viswanathan, and Gul Agha. 2004. Statistical model checking of black-box probabilistic systems. In *Proceedings of the 16th International Conference on Computer Aided Verification*, edited by Rajeev Alur and Doron A. Peled, vol. 3114 of *Lecture Notes in Computer Science*, 202–215, Boston. Springer.

Serfozo, Richard F. 1979. An equivalence between continuous and discrete time Markov decision processes. *Operations Research* 27, no. 3: 616–620.

Shedler, Gerald S. 1993. *Regenerative Stochastic Simulation*. Boston: Academic Press.

Simmons, Reid G. 1988. A theory of debugging plans and interpretations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 94–99, Saint Paul, Minnesota. AAAI Press.

Smith, David E. and Daniel S. Weld. 1998. Conformant graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 889–896, Madison, Wisconsin. AAAI Press.

Stone, Lawrence D. 1973. Necessary and sufficient conditions for optimal control of semi-Markov jump processes. *SIAM Journal on Control* 11, no. 2: 187–201.

Teichroew, Daniel and John Francis Lubin. 1966. Computer simulation—discussion of the techniques and comparison of languages. *Communications of the ACM* 9, no. 10: 723–741.

Telek, Miklós and A. Heindl. 2002. Matching moments for acyclic discrete and continuous phase-type distributions of second order. *International Journal of Simulation Systems, Science & Technology* 3, no. 3–4: 47–57.

Ulam, S. M. and John von Neumann. 1947. On combination of stochastic and deterministic processes. *Bulletin of the American Mathematical Society* 53: 1120. Abstract 403.

Utgoff, Paul E., Neil C. Berkman, and Jeffery A. Clouse. 1997. Decision tree induction based on efficient tree restructuring. *Machine Learning* 29, no. 1: 5–44.

Vardi, Moshe Y. 1985. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th Annual Sumposium on Foundations of Computer Science*, 327–338, Portland, Oregon. IEEE Computer Society.

von Neumann, John. 1951. Various techniques used in connnection with random digits. *National Bureau of Standards Applied Mathematics Series* 12: 36–38.

Wadsworth, George P. and Joseph G. Bryan. 1960. *Introduction to Probability and Random Variables*. New York: McGraw-Hill Book Company.

Wald, Abraham. 1945. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics* 16, no. 2: 117–186.

———. 1947. *Sequential Analysis*. New York: John Wiley & Sons.

Wald, Abraham and J. Wolfowitz. 1948. Optimum character of the sequential probability ratio test. *Annals of Mathematical Statistics* 19, no. 3: 326–339.

Weibull, Waloddi. 1951. A statistical distribution function of wide applicability. *Journal of Applied Mechanics* 18: 293–297.

Weiss, Lionel. 1953. Testing one simple hypothesis against another. *Annals of Mathematical Statistics* 24, no. 2: 273–281.

———. 1962. On sequential tests which minimize the maximum expected sample size. *Journal of the American Statistical Association* 57, no. 299: 551–566.

Weld, Daniel S. 1994. An introduction to least commitment planning. *AI Magazine* 15, no. 4: 27–61.

Whitt, Ward. 1982. Approximating a point process by a renewal process I: Two basic methods. *Operations Research* 30, no. 1: 125–147.

Younes, Håkan L. S. 2003. Extending PDDL to model stochastic decision processes. In *Proceedings of the ICAPS-03 Workshop on PDDL*, 95–103, Trento, Italy.

———. 2004. "Black-box" probabilistic verification. Technical Report CMU-CS-04-162, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Younes, Håkan L. S., Marta Kwiatkowska, Gethin Norman, and David Parker. 2004. Numerical vs. statistical probabilistic model checking: An empirical study. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, edited by Kurt Jensen and Andreas Podelski, vol. 2988 of *Lecture Notes in Computer Science*, 46–60, Barcelona, Spain. Springer.

Younes, Håkan L. S. and Michael L. Littman. 2004. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Younes, Håkan L. S. and David J. Musliner. 2002. Probabilistic plan verification through acceptance sampling. In *Proceedings of the AIPS-02 Workshop on Planning via Model Checking*, edited by Froduald Kabanza and Sylvie Thiébaux, 81–88, Toulouse, France.

Younes, Håkan L. S., David J. Musliner, and Reid G. Simmons. 2003. A framework for planning in continuous-time stochastic domains. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*, edited by Enrico Giunchiglia, Nicola Muscettola, and Dana S. Nau, 195–204, Trento, Italy. AAAI Press.

Younes, Håkan L. S. and Reid G. Simmons. 2002a. On the role of ground actions in refinement planning. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling Systems*, edited by Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, 54–61, Toulouse, France. AAAI Press.

———. 2002b. Probabilistic verification of discrete event systems using acceptance sampling. In *Proceedings of the 14th International Conference on Computer Aided Verification*, edited by Ed Brinksma and Kim Guldstrand Larsen, vol. 2404 of *Lecture Notes in Computer Science*, 223–235, Copenhagen, Denmark. Springer.

———. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research* 20: 405–430.

———. 2004a. Policy generation for continuous-time stochastic domains with concurrency. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, edited by Shlomo Zilberstein, Jana Koehler, and Sven Koenig, 325–333, Whistler, Canada. AAAI Press.

———. 2004b. A formalism for stochastic decision processes with asynchronous events. In *Papers from the AAAI Workshop on Learning and Planning in Markov Processes—Advances and Challenges*, 107–110, San Jose, California. AAAI Press. Technical Report WS-04-08.

———. 2004c. Solving generalized semi-Markov decision processes using continuous phase-type distributions. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 742–747, San Jose, California. AAAI Press.

Zaki, Mohammed J., Neal Lesh, and Mitsunori Ogihara. 2000. PlanMine: Predicting plan failures using sequence mining. *Artificial Intelligence Review* 14, no. 6: 421–446.

Index