Importance sampling for Probabilistic Timed Automata

Marco Pasveer
Computer Science, Formal Methods and Tools for Verification,
University of Twente, Enschede, The Netherlands.
marcopasveer@hotmail.com

Supervisors:
P.T. de Boer
D.P. Reijsbergen
M.I.A. Stoelinga
B.R.H.M. Haverkort

February 8, 2013
Abstract

In the last few years computers have become a much larger part of our daily lives. Because of this it has also gotten more important for computer programs to be correct. Nobody would want the medical software in hospitals to be incorrect. If such software has to be patched or recalled because it is incorrectly working, it can cost a large amount of money to fix the flaws in the software. Using computer models and so called model checkers, it has become possible to verify if a computer model is working as intended or not. However due to the complexity or size of certain computer models it becomes hard to completely verify if a computer program does what it is supposed to do. Simulation can help in this case. In simulation we run a model several thousands of times and see whether or not certain failures occur. Statistical tests on the results can be done to see how often a failure occurs. A problem arises when the probability of a certain failure occurring is very low (a rare event). When rare events are present it could take a very large number of simulation runs before the rare event is even perceived.

This is where importance sampling comes in. With importance sampling we increase the probability that a certain failure is triggered so we can reduce the number of simulation runs needed to make an accurate estimate. Afterwards the result is multiplied with a factor to account for the error we create when increasing the probability that a certain failure is triggered. Importance sampling can be applied on a wide number of models. Known techniques such as failure biasing are already applied on models like Discrete/Continuous Time Markov Chains.

We have worked on techniques to apply importance sampling on a class of models called probabilistic timed automata (PTA). We used a number of different papers as our basis, however the semantics and definitions of PTA in these papers were not always clear to us. There were also a number of discrepancies between the papers. Due to these definition problems, we had to perform proper research into the true workings of these PTA during simulation in order to apply importance sampling on them. We found out that semantics and underlying probability distributions normally used for simulating PTA were not behaving as we would expect and that there were several different versions of PTA defined by different people. We supply updated definitions on the real semantics of PTA later on in this thesis. We also supply a number of alternative distributions for simulation of PTA. These alternative distributions behave more logically in our eyes than the distributions used for simulating PTA mentioned in the papers we researched.

Another problem was the flexible behavior of timed automata which resulted in the fact that timed automata can cause a range of different kinds of rare events trough for examples very low probabilities on edges, small waiting times (delays), the flexible behavior of the model-checker Uppaal, via networks of PTA influencing other PTA and more. Each kind of rare event possibly requires a different approach in applying importance sampling. We limited ourselves during this master project to only the most important kinds of rare events; rare events caused by the distributions attached to locations and rare events caused by branching edges.

We altered a specific importance sampling technique called failure biasing, in which the probabilities of certain edges towards (rare) locations are increased in Discrete/Continuous Time Markov Chain models, in order to apply importance sampling on PTA by performing a number of case studies.

We also developed a technique called delay biasing in which we calculate the probability that a certain edge is taken by letting the model wait a specific amount of time, and increase said probability if necessary, also in order to apply importance sampling on
a number of case studies. On the case studies we performed for both delay biasing and failure biasing we have seen some good results that show that importance sampling could be a viable alternative and an improvement to regular simulation. Unfortunately we have also seen results where the estimates were incorrect when wrongfully applying importance sampling. This showed that our developed failure biasing technique does not necessarily have to be correct.

Both delay biasing and failure biasing were implemented in a Java program capable of simulating PTA, which we named MISS: Marco’s Importance Sampling Simulator for PTA.

By using the MISS tool we can simulate PTA via regular Monte Carlo simulation and via importance sampling. Some restrictions do have to be made before the MISS tool can simulate a PTA. As input a specially prepared PTA is taken, created in Uppaal. The estimate of the probability to reach a failure location is given as output. We used the MISS tool to simulate a number of case studies.
Chapter 0

List of Symbols

$L$ The location set.

$l$ A single location. $l'$ denotes the target location.

$l_0$ The initial location.

$\Sigma$ The action alphabet.

$C$ The clock set.

$x_1$ Clock $i$. (Multiple clocks possible)

$x$ Clock. Equals $x_1$. (x used in pictures)

$B(C)$ The set of all possible clock constraints.

$U(C)$ The set of all possible clock valuations.

$\nu$ Valuation symbol.

$\nu_0$ The clock valuation at time 0.

$\nu'$ Target valuation. (After transition)

$E$ Edge set.

$E_l$ Union of guards attached to location $l$.

$e$ A single edge.

$I$ Invariant function, assigns clock constraints to locations.

$g$ A single guard.

$\bar{g}$ A merged guard.

$\sigma$ Execution run notation.

$d$ A single delay.

$d_i$ Delay at step $i$.

$a$ A single action.

$a_i$ Action at step $i$.

$o$ A single output. $o \in \Sigma$.

$I(l)$ The invariant at location $l$.

$\mathcal{L}$ Labeling function assigning labels to locations.

$r$ A set denoting the clocks to be reset during a transition.

$R$ Exponential rate set.

$R_s$ Rate at location $l$.

$\models$ Denotes the satisfaction relation.

$s = (l, \nu)$ A state. Combination of both a location and a valuation.

$s' = (l', \nu')$ The target state. (After transition)

$I(s)$ Set of all possible delays satisfying a guard at state $s$.

$p(r, l') \in P(2^C \times L)$ Probability of a single branching edge.

$p(\cdot\cdot\cdot)$ Probability density function. (Original)

$q(\cdot\cdot\cdot)$ Probability density function. (Change of measure)

$q^*(\cdot\cdot\cdot)$ Probability density function. (Optimal change of measure)

? Multiple variables used. e.g. $(x, \sigma)$. 
\(\hat{\gamma}\) Estimated value.

\(\gamma\) True value.

\(N\) The number of executions/samples.

\(L(\sigma_i)\) The likelihood ratio for sample \(\sigma_i\).

\(G(e)\) Represents the guard of edge \(e\).

\(G_s\) Set of all guards attached to location \(l\) via some edge.

\(\bar{G}_s\) Set of all merged guards attached to location \(l\) via some edge.

\(Sup(g) = \omega(g, \nu)\) Guard \(g\)'s supremum delay.

\(Inf(g) = \kappa(g, \nu)\) Guard \(g\)'s infimum delay.

\(\kappa(l, \nu)\) Infimum delay for location \(l\) and valuation \(\nu\).

\(\omega(l, \nu)\) Supremum delay for location \(l\) and valuation \(\nu\).

\(G^s(l, \nu)\) Guard gap set for location \(l\) and valuation \(\nu\).

\(G^s(l, \nu)\) Guard gap section for location \(l\) and valuation \(\nu\).

\(H_s\) The set of all enabled edges in state \(s\).

\(M_s(K)\) Probability distribution function over delays in state \(s\) for interval \(K\).

\(K = [a, b]\) with \(a, b \in \mathbb{R}^+\) A closed interval ranging from \(a\) to \(b\).

\(\eta_s\) Probability distribution function over output in state \(s\).
# Contents

0 List of Symbols  
1 Thesis Introduction  
   1.1 Motivation  
   1.2 Problem definition  
      1.2.1 Research Topics  
   1.3 Thesis structure  
   1.4 State of the art  
      1.4.1 Importance sampling on other models  
      1.4.2 Different approaches  
      1.4.3 Tools  
   1.5 Restrictions & Correctness  
2 Roadmap  
   2.1 Background  
      2.1.1 Model checking  
      2.1.2 Timed Automata and Probabilistic Timed Automata  
      2.1.3 Uppaal  
      2.1.4 Rare Event Simulation and Importance Sampling  
   2.2 New Research  
      2.2.1 The semantics of Uppaal  
      2.2.2 Alternative semantics  
      2.2.3 Rare events & properties  
      2.2.4 Failure biasing  
      2.2.5 Delay biasing  
      2.2.6 The MISS tool  
      2.2.7 Conclusion  
   2.3 Appendix  
   2.3.1 Model alterations/improvements  
   2.3.2 Discrete event simulation & Statistical model checking  
I Background  
3 Model checking  
   3.1 Model checking  
   3.2 Simulation  
      3.2.1 Importance sampling
8.2.4 Output density function $\eta_s$ .............................................. 57
8.3 New semantics 1: Guard gaps .................................................... 58
  8.3.1 Uniform distribution ......................................................... 58
  8.3.2 Exponential distribution ..................................................... 58
  8.3.3 Implementation ............................................................... 59
8.4 New Semantics 2: Guard gaps .................................................... 60
8.5 New Semantics 3: Guard gaps .................................................... 60
  8.5.1 The uniform distribution .................................................... 61
  8.5.2 The exponential distribution ............................................... 62
8.6 New Semantics 4: Overlapping guards ........................................ 63
  8.6.1 The uniform distribution .................................................... 63
  8.6.2 The exponential distribution ............................................... 64
8.7 Timelock semantics ............................................................... 65
  8.7.1 Current timelock semantics ................................................. 65
  8.7.2 New timelock semantics ..................................................... 65
9 Rare events & properties .......................................................... 66
  9.1 Rare events ........................................................................ 66
    9.1.1 Is this list of rare events exhaustive? ................................ 69
    9.1.2 Interesting rare events ..................................................... 70
  9.2 Model Properties .................................................................. 72
    9.2.1 Statistical Properties ......................................................... 72
    9.2.2 State Properties $\psi$ and $\phi$ ............................................. 73
    9.2.3 Interesting properties ........................................................ 73
  9.3 Conclusion: Interesting rare events & properties ....................... 74
10 Failure biasing ........................................................................ 75
  10.1 Introduction ......................................................................... 75
  10.2 Known failure biasing techniques ........................................... 76
    10.2.1 Simple failure biasing ....................................................... 76
    10.2.2 Balanced Failure Biasing .................................................... 77
  10.3 PTA failure biasing requirements ............................................ 77
    10.3.1 Repair/failure transitions ............................................... 77
    10.3.2 Loop transitions .............................................................. 78
    10.3.3 Model exploration ........................................................... 78
    10.3.4 How smart is the user? ...................................................... 78
    10.3.5 A good change of measure ................................................. 79
  10.4 PTA failure biasing .............................................................. 79
    10.4.1 The branch likelihood ratio ............................................... 80
    10.4.2 Case studies ................................................................. 80
    10.4.3 PTA output failure biasing ............................................... 83
  10.5 Conclusion ......................................................................... 84
11 Delay Biasing .......................................................................... 86
  11.1 Introduction ......................................................................... 86
  11.2 PTA delay biasing requirements ............................................. 87
    11.2.1 Guard assumptions .......................................................... 87
    11.2.2 Delay & Edge selection process ....................................... 88
    11.2.3 Failure biasing approach ................................................. 89
  11.3 Delay biasing for the uniform distribution .................................. 89
    11.3.1 Calculating the probabilities ............................................. 89
    11.3.2 Delay biasing ................................................................. 90
11.4 Delay biasing on the exponential distribution ............................................. 90
  11.4.1 Calculating the probabilities .......................................................... 91
  11.4.2 Delay Biasing ................................................................................. 91
11.5 The jump likelihood ratio ........................................................................ 92
11.6 Case studies ............................................................................................ 92
11.7 Conclusion ............................................................................................... 97
11.8 Delay biasing & Failure biasing ............................................................... 97
11.9 Markovian correctness ........................................................................... 97
11.10 Delay/Failure biasing correctness ........................................................ 98

12 The MISS Tool ............................................................................................ 99
  12.1 Input ....................................................................................................... 100
    12.1.1 Rule 1: Declarations ................................................................. 100
    12.1.2 Rule 2: Updates/assignments .................................................. 101
    12.1.3 Rule 3: Locations .................................................................. 101
  12.2 File chooser .......................................................................................... 102
  12.3 Parser .................................................................................................. 102
  12.4 Simulator ............................................................................................... 103
  12.5 Output ................................................................................................... 104

13 Conclusion .................................................................................................. 106
  13.1 The Uppaal semantics .......................................................................... 106
  13.2 Alternative semantics ......................................................................... 106
  13.3 Rare events ........................................................................................... 107
  13.4 Failure biasing ..................................................................................... 107
  13.5 Delay biasing ....................................................................................... 108
  13.6 Modeling limitations ........................................................................... 108
  13.7 The MISS tool ..................................................................................... 109

A Model alterations/improvements .................................................................. 110
  A.0.1 Networks of PTA ............................................................................. 110
  A.0.2 Time locks ....................................................................................... 110
  A.0.3 Dead locks ....................................................................................... 111
  A.0.4 Urgent locations ............................................................................. 111
  A.0.5 Committed locations ..................................................................... 112
  A.0.6 Branching edges ............................................................................ 112
  A.0.7 Merging locations .......................................................................... 113
  A.0.8 Other distributions ...................................................................... 114
  A.0.9 The uniform/exponential distribution .......................................... 114
  A.0.10 Guard splitting ............................................................................. 114

B Discrete-Event Simulation .......................................................................... 118

C Statistical Model Checking ......................................................................... 119
  C.1 Hypothesis testing .............................................................................. 119
  C.2 Single Sampling Plan ......................................................................... 120
  C.3 Sequential Tests ................................................................................. 122
    C.3.1 SPRT ........................................................................................... 122
    C.3.2 Supergaussian Sequential Tests ............................................... 123
Chapter 1

Thesis Introduction

In the last few years computers have become a much larger part of our daily lives. Because of this it has also gotten more important for computer programs to be correct. Nobody would want the medical software in hospitals to be incorrect. If such software has to be patched or recalled because it is incorrectly working, it can cost a large amount of money to fix the flaws in the software. Using computer models and so called model checkers, it has become possible to verify if a computer model is working as intended or not [11]. Model checkers are tools (examples are Uppaal or Prism [8, 20]) that given a model of a system, exhaustively and automatically check whether this model meets a given specification. However due to the complexity or size of some computer models it becomes hard to completely verify if a computer program does what it is supposed to do.

In simulation we run a model several thousands of times and see whether or not certain failures occur. We can perform statistical tests on the results of the simulation and see how often a failure occurs. A problem with simulation arises when the probability of a certain failure occurring is very low (a rare event). In those cases it could take a lot of simulation runs before we can even detect the failure, let alone determine the probability of triggering the failure. Another problem with simulation is the accuracy of the results. Since simulation is based on random tests, there is a possibility that the results are not accurate. This is where importance sampling comes in. With importance sampling we increase the probability that a certain failure is triggered so we can reduce the number of simulation runs needed to make an accurate guess. Afterwards the result is multiplied with a factor to account for the error we create when increasing the probability that a certain failure is triggered.

In this thesis we apply importance sampling on a specific class of models called Probabilistic Timed Automata (PTA) based on the Timed Automata from Alur [4]. Importance sampling techniques for this model have not been invented yet, but have been for other models. In this thesis we will alter specific importance sampling techniques to work with PTA.

1.1 Motivation

While Timed Automata [4] have been around for almost two decades, Probabilistic Timed Automata are relatively new [13, 12, 25, 21]. Simulation techniques for PTA have only been developed in the last 3 years. Simulation for PTA has been implemented into model-checkers like Uppaal and Prism.

While importance sampling techniques have been around for a while, importance sampling for PTA does not exist yet. With this thesis we supply techniques that apply importance sampling to PTA. While normal simulation for PTA already works in the tool Uppaal [3], with these importance sampling techniques PTA will be able to be model-
checked faster and more accurately. In turn larger and more complex PTA can also be model-checked if normal simulation was already sufficient.

By using the importance sampling techniques we can achieve a variance reduction over regular simulation. Estimates become more accurate and fewer simulation runs have to be performed for an accurate result. Also by applying importance sampling on PTA, something which has not been before, we aim to fill a void of rare event simulation for PTA.

1.2 Problem definition

Simulation is the imitation of the operation of a real-world process or system over time. [7] The act of simulating something first requires that a model is developed; this model represents the key characteristics or behaviors of the selected physical or abstract system or process. The model represents the system itself, whereas the simulation represents the operation of the system over time.

During simulation the model is executed a fixed number of times. Each time the model starts in the specified initial location and based on the probabilities in the model the model randomly selects a direction to head towards. The simulation stops when it reaches a certain place or point in time such that it can or should not continue or when a failure is triggered. From the number of failures triggered and the number of total simulation runs started in total it is possible to obtain estimates for the probability that a failure is triggered. The higher the number of simulations the better the estimate is in accuracy.

Problems with simulation arise when the probability of a certain failure occurring is very low, also called a rare event. The rarer the event, the more simulation runs need to be done to get an accurate result. If for example a rare event has a probability of $10^{-20}$ of being triggered it would take on average $10^{20}$ simulations before the rare event is triggered. This can be problematic as current day computers might take too long to get an accurate result. A technique called importance sampling (IS) can reduce the number of simulation runs needed if applied correctly. However finding a way to apply IS correctly is no easy feat when we want to apply it on a model, called Probabilistic Timed Automata (PTA), that can contain a wide area of different rare events. Each rare event could potentially require a different way of applying IS.

Another problem is that the models we will look into in this thesis have only seen research the last couple of years. There has been no research on applying importance sampling on PTA and only a small amount of research on simulation for PTA. In order to get a good understanding of PTA we have to research how certain aspects of PTA work.

From research done during the research topics project we formulated a number of research questions.

These research questions are:

Question 1: How is the relation between the practical use of the uniform and exponential distributions and Uppaals implementation of them?

Question 2: What are the different kinds of rare events and properties in PTA and which of them are interesting for this project?

Question 3: Can we use failure biasing techniques for particular rare events in PTA to create a good change of measure?

Question 4: Should our change of measure use different probability densities for the transition times? And if so, what would they look like?

Question 5: Do we have to restrict our models and properties in some way to be able to
construct a well-performing change of measure and apply IS in Uppaal?

From research during this master thesis we were able to formulate another major research question:

Question 1b: What are the current and alternative semantics for models where no edges are enabled at some point in time?

Note: This question is a follow up question to question one. At this point in the thesis the reader might not know the reasoning behind this question. This will be discussed in chapter 7.

1.2.1 Research Topics

During the course/project research topics we made some initial studies into various subjects discussed in this thesis. The results of research topics was a report. Some background information and related work can be found in this report. More information on this can be found later in this thesis. We also added a few chapters of our research topics report to the appendix.

1.3 Thesis structure

In this thesis in 'part 1: Background’, after this introduction, we will first introduce a number of subjects to make the reader understand the theory behind the subjects we will be talking about in the rest of the paper. In chapter 3 we will give an introduction in modelchecking and simulation. Chapter 4 will contain some of the models we intend to modelcheck. Chapter 5 will introduce a tool used to modelcheck some models from chapter 4. Chapter 6 will give an introduction to importance sampling.

In 'part 2: New research’ of this thesis we will try to answer the research questions we formulated under problem definition. Each question, including 1b, excluding 5, has a chapter in which we try to answer the research question. We end with a conclusion chapter in which we look back at all research questions. Question 5 will be answered during the failure biasing and delay biasing chapters. We also included a chapter on a tool we developed ourselves as a replacement for IS in Uppaal.

More information on the content of this thesis can be found in the roadmap chapter, which comes up next.

1.4 State of the art

We have split this section into three parts: IS on other models, different IS approaches and tools.

1.4.1 Importance sampling on other models

Importance sampling for Probabilistic Timed Automata is new. However there have been published a wide number of articles about importance sampling applied to other models.

In [39, 31, 40, 15] importance sampling techniques have been discussed by Shahabud-din, Nakayama, Glynn et al., which were applied on Discrete/Continuous Time Markov Chains. One such technique named failure biasing increases/decreases the fixed probability that a certain edge is taken by a given algorithm and parameters set by the user. We
based the failure biasing techniques in this master thesis on the techniques developed by Shahabudin et. al..

In [33] importance sampling techniques for stochastic activity networks (SANs) were discussed by Obal et. al. SANs have been demonstrated as an effective framework for discrete event system modeling. SANs are quite similar to TA. They contain places/locations, inputs, outputs, delays (timed activities) and instant edge traversals (instantaneous activities). IS for SANs is achieved by an altered failure biasing approach and is based on work from Shahabuddin and Juneja.

In [15] importance sampling for generalized semi-Markov processes (GSMP) was discussed. A GSMP is basically a mathematical formalization of a discrete-event simulation; the process is driven by an event-scheduling algorithm, in exactly the same way as is a discrete-event system.

In [14] Glasserman goes into importance sampling for tandem Jackson queues. The estimator they study is based on interchanging the arrival rate and the smallest service rate and is a generalization of the asymptotically optimal estimator for an M/M/1 queue.

\subsection*{1.4.2 Different approaches}

In [36] Ridder investigates Markovian reliability models and applies an algorithm known as the cross entropy (CE) method. In a nutshell the CE method consists of two phases: First generate a random data sample (trajectories, vectors, etc.) according to a specified mechanism. Second update the parameters of the random mechanism based on the data to produce a "better" sample in the next iteration. A number of Markov chain examples using the CE method were shown that outperform failure biasing techniques.

Simulated annealing is a method where one moves from a tractable distribution to a distribution of interest via a sequence of intermediate distributions using Markov chains. Neal in [32] discusses annealed importance sampling where it is shown how one can use the Markov chain transitions for such an annealing sequence to define an importance sampler.

Kwiatkowska et. al. in [25] present symbolic model-checking algorithms for probabilistic timed automata to verify both qualitative and quantitative temporal logic properties. The algorithms operate on zones, which represent sets of valuations of the probabilistic timed automaton clocks.

When importance sampling can not be used, one could stick to regular statistical model checking (SMC). The crux of the SMC approach is that since sample executions of a stochastic system are drawn according to the distribution defined by the system, they can be used to get estimates of the probability measure on executions. In [28] an overview is given by Legay et. al..

Statistical model checking techniques have also been discussed by David in [12] for (priced) timed automata. He discusses a number of tests that can be used to estimate the probability that a given property is true. Among these tests is a hypothesis test, based on Walds sequential hypothesis test [42]. More info on hypothesis testing can be found in the appendix.
1.4.3 Tools

PRISM is a probabilistic model checker [20, 2], a tool for formal modeling and analysis of systems that exhibit random or probabilistic behavior. It supports a wide range of probabilistic models, including probabilistic timed automata. PRISM can model check a wide range of quantitative properties, expressed in a language that subsumes the temporal logics PCTL, CSL, LTL and PCTL*, using both symbolic (BDD-based) and explicit-state model-checking engines. PRISM also contains a discrete-event simulator, with support for statistical model checking methods, including confidence-level approximation and acceptance sampling.

Uppaal [8, 3] is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata or probabilistic timed automata, extended with data types (bounded integers, arrays, etc.). The tool is developed in collaboration between the Department of Information Technology at Uppsala University, Sweden and the Department of Computer Science at Aalborg University in Denmark.

1.5 Restrictions & Correctness

During the course of this thesis we will be making a number of assumptions, definitions and algorithms about PTA and importance sampling for PTA. There are a range of different PTA definitions, there are many different importance sampling approaches and for this reason we had to restrict ourselves to the elements of importance sampling and PTA that were most interesting to us. Most of the details for the restrictions are discussed in the corresponding chapters. Some of the restrictions are:

- We go into models with only one clock.
- We assume that guard gaps are filled up with selfloops as shown in section 8.2.2 and in the appendix A.0.9. This disallows the selection of multiple delays (waiting times) in a row. After a delay has been drawn an edge, with a corresponding action, to a new location must be taken.
- We focus ourselves on the PTA that can be created with the Uppaal tool [3]. More info on these PTA and its semantics can be found in the Uppaal help or in the papers [12, 13].
- We only take into account reachability properties. Other kinds of properties are only briefly discussed, but not used during model-checking.

These restrictions apply on our semantics chapters, our failure biasing techniques and our developed MISS tool.

Correctness

The failure biasing approaches we used for applying importance sampling on PTA were created for Markov chains, where models have the memoryless property. While failure biasing has been proven correct for Markov chains, we did not prove its correctness for PTA. Some failure biasing results we show later in this thesis that we can obtain inaccurate estimates if failure biasing is applied wrong.
Chapter 2

Roadmap

In this chapter we will introduce the reader to the contents of this thesis. We will do this in the same order as the thesis is written. We start with the background information which is needed for the reader to understand the new research we have done for the master project that this thesis is a part of.

2.1 Background

The background part of this thesis serves as an introduction to the subjects we have researched.

2.1.1 Model checking

In the background part of this thesis we will start with an introduction into model checking and simulation in chapter 3. Model checking is done via a model checking tool which takes a model and a specification and checks whether or not the model satisfies the specification. In simulation or statistical model checking the model is simply run started a number of times. If the run of the model at some point no longer satisfies the specification an error is generated and a new run is started. From the number of runs and the number of errors an estimate can be obtained denoting how likely an error is triggered each run.

2.1.2 Timed Automata and Probabilistic Timed Automata

In chapter 4 we discuss the definitions of several models that can be used in model checking tools like Uppaal and Prism. The first model is a Timed Automaton which are finite state transition system extended with real-valued clocks that denote time. The second model is a Probabilistic Timed Automaton, which is a Timed Automaton extended with probabilistic transitions. In the Simulation Models chapter we define each aspect of the TA and the PTA as they are used in the remainder of the thesis. We conclude the chapter with another model, namely the Discrete Time Markov Chain (DMTC), which will be referenced in the remainder of the thesis a couple of times.

2.1.3 Uppaal

In order to model check a model a model checking tool is required. In chapter 5 we discuss the working of a tool we heavily used during our master project and which is referenced a high number of times during this thesis. This tool, named Uppaal, can symbolically and statistically model check a PTA and provide estimates for how likely it is that the PTA model satisfies its specification. We give Uppaals algorithm for statistically model
check/simulate a given model based on a number of definitions and assumptions. We finish with a section on strange behavior in Uppaal which the user should be made aware of.

2.1.4 Rare Event Simulation and Importance Sampling

In chapter 6 we discuss the details behind simulating a standard model by techniques such as Monte Carlo simulation and Importance Sampling, which is a general technique for estimating properties of a particular distribution, while drawing samples from a different distribution rather than the distribution of interest. We give formula’s that can be used to estimate the fraction of simulated runs that do not satisfy its specification. We also alter the formula’s to work with Importance Sampling and supply a few measures for the accuracy of the estimates. We end chapter 6 with information on how Importance Sampling works for PTA models. This information acts as a basis for the rest of the thesis.

2.2 New Research

The new research part of this thesis is the core of this thesis. It features a number of chapters on research we have achieved during the course of the master project.

2.2.1 The semantics of Uppaal

In order to simulate PTA in Uppaal it is required that either an invariant or a rate of exponential is specified on a location. The Uppaal simulator then uses the invariant/rate of exponential to draw delays, which are waiting times for locations/places in the PTA, according to either the uniform distribution or the exponential distribution. In chapter 7 we discuss the semantics and formulas used for drawing the delays. These formulas can take a wide range of shapes and can be influenced by the invariant/rate of exponential and guards which are attached to the outgoing edges attached to the location the PTA is currently in.

2.2.2 Alternative semantics

The Uppaal semantics from the previous section can behave weirdly when guard gaps are present. In chapter 8 we discuss a number of semantics as alternatives to the current Uppaal semantics for simulating PTA. The new semantics behave more like the reader might expect of a tool like Uppaal. Together with these new semantics several formulas for determining the delay, examples and modeling alternatives are presented.

2.2.3 Rare events & properties

PTA models that do not satisfy the specifications can trigger errors during simulation. When the probability of the error being triggered is very low the error being triggered can be considered a rare event. In chapter 9 an exhaustive list of all rare events is presented which can help us and the reader understand when and when not importance sampling can be used. A number of properties, which can be used to create specifications, are also given. We conclude with a discussion on which rare events and properties are interesting for this master project.

2.2.4 Failure biasing

Importance sampling has been achieved on a number of different models, however on PTA it has not been applied yet. One importance sampling technique named failure biasing has
shown interesting results for Discrete/Continuous Time Markov Chains. Failure biasing is a technique used to reduce the variance of estimates of performance measures and to increase the likelihood that rare events are triggered. In chapter 10 we discuss existing failure biasing techniques and apply them on the branching edges that are present in PTA. Using failure biasing on PTA we can increase the likelihood an error is triggered by increasing the probability that certain branching edges are chosen.

We also give a number of case studies that apply our failure biasing techniques on PTA.

2.2.5 Delay biasing

In PTA it is possible that the probability that a rare event is triggered does not depend on branching edges, where we can apply failure biasing, but on drawing the right delay. In chapter 11 we discuss an importance sampling technique called delay biasing, which is an altered form of failure biasing where we increase the probability that a certain delay is drawn. Drawing the right delay can help increase the likelihood that a rare event is triggered.

We also give a number of case studies that apply our delay biasing techniques on PTA.

2.2.6 The MISS tool

We created our own model-checking tool which we named MISS: Marco’s Importance Sampling Simulator for PTA. This tool is capable of simulating PTA using both Monte Carlo techniques and the failure/delay biasing techniques we discuss in this thesis. The tool takes as input a specially prepared PTA created in Uppaal and outputs the estimate to trigger a rare event and the variance of the estimate. We discuss the tool in chapter 12.

2.2.7 Conclusion

We follow-up on the new research we have performed with a conclusion chapter.

2.3 Appendix

Some additional work was done, but ultimately it did not fit in among the new research part of this thesis, so we discuss the additional work in the appendix.

2.3.1 Model alterations/improvements

Chapter A contains a number of model alterations/improvements that the reader could use to improve his/her models such that better importance sampling estimates can be obtained. Various PTA models can be altered in such a way that failure biasing or delay biasing can be used while the original models might not allow importance sampling techniques. Models can also be improved by for example merging certain vertices and/or edges together to obtain more accurate results.

2.3.2 Discrete event simulation & Statistical model checking

Also in the appendix are two chapters on discrete event simulation and statistical model checking. These two chapters were part of my research topics project.
Part I

Background
Chapter 3

Model checking

3.1 Model checking

Model Checking [34, 45, 12] refers to the following: Given a model of a system, check to see whether the model meets a given specification or not. A model checker or tool takes a model and a specification of the system and checks whether or not the model satisfies the specification, which is a property of the system usually defined in a formal language. A property example is “the system never deadlocks”, more information on properties can be found in chapter 9. The tool afterwards outputs yes or true if the given model satisfies given specifications and no/false otherwise. Some model checkers also generate counterexamples if the specification is not satisfied. The counterexample details why the model doesn’t satisfy the specification. By studying the counterexample, one can pinpoint the source of the error in the model, correct the model, and try again. The idea is that by ensuring that the model satisfies enough system properties, one can increase his/her confidence in the correctness of the model.

Models are usually represented as state-transition graphs. These graphs describe the behavior of the system. This behavior is analyzed and represented in series of events, also called executions or sometimes paths, by the model checker. For each of these executions the model checker checks whether or not it satisfies a certain specified property. Figure 3.1 shows a very simple state transition diagram of an elevator.

![Figure 3.1: A state transition diagram of an elevator.](image)

The model in figure 3.1 starts in the open location where the only possible action is closing the door. When the door is closed the elevator can start moving up or down or open the door again. When the elevator is moving it must stop at some point in time again to allow the door to be opened again if needed.

Properties are usually represented in temporal logic languages. These languages allow us to describe any system of rules and symbolism for representing, and reason about, properties qualified in terms of time. We will not be going into these temporal logics in detail.

There exist a wide range of different kinds of models one could use to model a system.
Systems that have some kind of randomness attached to it can be called stochastic systems and they can have one or more components that are stochastic in nature. For example, they can have random-time delays, probabilistic transitions or stochastic inputs. Properties of stochastic systems are usually specified to allow one to see the number of executions satisfying those properties. The model checking problem for stochastic systems is usually solved by a numerical approach that iteratively approximates the exact measure of paths satisfying relevant sub formulas. One example on how this is done is by exploring all the possible executions and then checking how much of those executions satisfy a given property.

Despite the usefulness of numerical model checking algorithms, there are also many problems with it. Larger systems require a lot of time and space when using numerical algorithms. The systems can in practice become unverifiable because of those time/space issues. We will not be going into those numerical model checking algorithms.

3.2 Simulation

Simulation is the imitation of the operation of a real-world process or system over time. The act of simulating something first requires that a model is developed; this model represents the key characteristics or behaviors of the selected physical or abstract system or process. During simulation, a simulation run describes the randomly chosen behavior of the system over time.

During simulation the model is executed a fixed number of times. Each time the model starts in the specified initial location and based on the probabilities in the model the model randomly selects a direction to head towards. The simulation stops when it reaches a certain place or point in time such that it can or should not continue or when a failure is triggered. From the number of failures triggered and the number of total simulation runs started in total it is possible to obtain estimates for the probability that a failure is triggered. The higher the number of simulations the better the estimate is in accuracy.

Simulation can also be used to verify quantitative properties of stochastic systems, see Legay [28] for more info. One can simulate the system a finite number of runs, and use hypothesis testing to check whether the samples provide a statistical evidence for the satisfaction or violation of any specified properties. This is known as statistical model checking (SMC). Sample executions are generated according to the underlying system model which describes the possible states and transitions between states and its probabilities. We included a chapter on statistical model checking in the appendix.

This approach of simulating the model a finite number of times has advantages. First, these algorithms only require that the system is executable. Thus, it can be applied to larger class of systems than numerical model checking algorithms. The algorithm is also easily parallelizable, which can help scale to larger systems.

Simulation can help reduce the time and space needed to obtain an estimate for a property. One of the primary flaws of simulation is that it can take an unacceptable long time before an accurate result can be produced. When the probability to reach a certain state is very low we call the event to reach said state a rare event. If we would use simulation to acquire an estimate of the probability that the rare event is triggered it would be highly inefficient as we are spending most of the simulation time on other events uninteresting for our estimates. Intuitively, we want to make the interesting, or important, events occur more often, so that we obtain a higher rate of relevant samples. This can be achieved by using importance sampling.
Note: Most of the work for this was done for my research topics report. The appendix contains two chapters on discrete event simulation and statistical model checking that contain more information.

3.2.1 Importance sampling

For very accurate results or for rare event simulation importance sampling could be used to get more accurate results or to get results faster. In importance sampling certain probabilities to transition to a new location are increased in such a way that the probability to enter a rare state is increased. The result of each simulation run gets multiplied a value called the likelihood ratio to correct the error introduced when increasing certain probabilities.

We will discuss importance sampling more in detail in chapter 6.
Chapter 4

Simulation models

There are several different ways of modeling systems. We will be introduce a few of them, namely those few that are also used in the rest of this thesis. In section 4.1 we will introduce timed automata. In 4.2 we discuss an extension to timed automata where probabilistic edges were added, namely probabilistic timed automata (PTA). There are some aspects of PTA not well-defined. In 4.3 we discuss literature on PTA. This is done to potentially flesh out a number of discrepancies that occur when trying to simulate a PTA. In 4.4 we end with 2 other models that are referenced in this thesis.

Shown pictures were created with the Uppaal model-checker [3].

4.1 Timed Automata

Timed Automata (TA) are finite state transition systems extended with clocks [4, 8]. Clocks are variables that denote time. As soon as the initial state is entered for the first time all clocks in the TA start. Locations and edges can have invariants and guards, respectively, attached to them that limit what the model can do. TA’s can be model-checked using the Uppaal tool.

Definition 4.1.1 (Clock valuation):
Let $C$ be a set of clocks, then a function $\nu : C \rightarrow \mathbb{R}^+$ is called a clock valuation. The set of clock valuations on $C$ is written $U(C)$ with $U(C) : C \rightarrow \mathbb{R}^+$.
For delay $d \in \mathbb{R}^+$ let $\nu + d$ denote the clock assignment that maps all $x \in C$ to $\nu(x) + d$.

Let $\nu_0(x) = 0$ for all $x \in C$.

Definition 4.1.2 (Clock reset):
A clock reset resets a given set of Clocks back to zero.
Let $r \subseteq C$ denote the set of clocks to be reset.
Let $[r \mapsto 0] \nu$ denote the clock valuation which maps each clock in $r$ to 0 and agrees with $\nu$ over $C \setminus r$. [8]

Definition 4.1.3 (Clock constraint):
Let $C$ be a set of clocks. A clock constraint on $C$ is a finite conjunction of expressions of the form $x_1 \sim k$ or $x_1 - x_2 \sim k'$ where $x_1, x_2 \in C, \sim \in \{<, \leq, =, \geq, >\}$, and $k \in \mathbb{N}^+ \cup \{0\}, k' \in \mathbb{Z}$.
Denote by $B(C)$ the set of all clock constraints.
A clock valuation $\nu$ satisfies the clock constraint $g \in B(C)$, written $\nu \models g$, if and only if $g$ resolves to true after substituting each clock $x \in C$ with the corresponding clock value from $\nu$. [13]
A clock constraint assigned to an edge is named a guard.
A clock constraint assigned to a location is named an invariant.

Invariants are usually in the form \( x_1 < k \) or \( x_1 \leq k \). Generally it is assumed that timed automata are well-formed, meaning: when taking an edge to a location the location’s invariant must always be true (mathematical definition given in chapter 5). This assumption makes invariants of the form \( x_1 > k \) and \( x_1 \geq k \) obsolete since the location of the invariant will always be entered with a clock-value larger than \( k \) and thus the invariant will always be true no matter how much time is spend in the locations.

Invariants of the form \( x_1 = k \) can be rewritten in to invariants \( x_1 \leq k \) where all incoming edges have guards \( x_1 = k \).

**Definition 4.1.4** (Timed Automata):
A TA is a tuple \((L, l_0, \Sigma, C, E, I, \mathcal{L})\) given by:
- \( L \) is the set of locations or nodes.
- \( l_0 \in L \) is the initial location.
- \( \Sigma \) is the action alphabet.
- \( C \) are the clocks denoted with a value \( \in \mathbb{R}_{\geq 0} \). All clocks evolve at the same rate. Clocks can be reset when an edge is taken. We denote \( x_i \) to be clock \( i \). We omit \( i \) when no other clocks are present (\( x \) is the only clock).
- \( E \subseteq L \times B(C) \times \Sigma \times 2^C \times L \) is the edge set. The power set of all possible combinations of set of clocks possible is denoted \( 2^C \) and denotes which clocks are reset.
- \( I : L \rightarrow B(C) \) assigns invariants to locations.
- \( \mathcal{L} : L \rightarrow 2^{\text{AP}} \) is a labeling function assigning atomic propositions to locations. \( \text{AP} \) is the set of all atomic propositions.

**Definition 4.1.5** (TA semantics):
The semantics of a TA \((L, l_0, \Sigma, C, E, I)\) are given by a labeled transition system \( \langle S, s_0, \rightarrow \rangle \) where:
- \( S \subseteq L \times B(C) \) is the set of states.
- \( s_0 = (l_0, u_0) \) is the initial state.
- \( \rightarrow \subseteq S \times S \) is the smallest transition relation such that:
  - \( (l, \nu) \xrightarrow{d} (l, \nu + d) \) if \( \forall d' : 0 \leq d' \leq d \implies \nu + d' \models I(l) \) with delay \( d \), location \( l \) and \( I(l) \) the invariant at location \( l \);
  - \( (l, \nu) \xrightarrow{e} (l', \nu') \) if there exists \( e = (l, a, g, r, l') \in E \) s.t. \( \nu \models g, \nu' = [r \mapsto 0]/\nu \models I(l) \) with edge \( e \), guard \( g \), action \( a \). \( r \) is the set of clocks that are reset to 0.

In other words: A delay can only be added to a clock if the new clock value does not break the invariant. The system can only take a transition if there exists an edge \( e = (l, a, g, r, l') \in E \) if the target location’s invariant and the guard \( g \) both hold. The selection of delay \( d \) is done non-deterministically.

![Figure 4.1: A Timed Automaton.](image)

Figure 4.1 shows a timed automaton. The model starts in the start location and can only continue to the idle location as long as the clock constraint \( x \leq 0 \) holds. There are two actions: Arrival and Departure. On arrival the clock is reset and goes towards the Busy location. In the Busy location the model must continue to the Idle locations via a departure before the clock \( x \) has surpassed 10.
Often timed automata are composed into a network of timed automata (NTA). NTA have a common set of clocks and actions and consist of a fixed number of timed automata working together.

More information about timed automata and networks of timed automata can be found in [4, 8].

**Definition 4.1.6 (Enabled edges):**

We denote $G(e)$ to be the guard of edge $e$.

We say that an edge $e$ is enabled in state $s = (l, \nu)$ if it is possible to traverse from $s$ to $s' = (l', \nu')$ using $e$ and $G(e)$ is satisfied, i.e. $s \xrightarrow{e} s'$ and $\nu \models G(e)$. $l$ is a location and $\nu$ a valuation.

We say that a guard $G(e)$ is enabled iff $e$ is enabled.

We denote $H_s$ the set of all enabled edges in state $s$, i.e. $H_s = \{ e \in E | e \text{ enabled in } s \}$

We denote with $|H_s|$ the number of elements in $H_s$.

We denote $G_s$ to be the set of guards attached to location $l$ in state $s = (l, \nu)$, i.e. $G_s = \{ g | \exists e = (l, g, -, -, -, -) \in E \}$.

## 4.2 Probabilistic Timed Automata

Probabilistic Timed Automata (PTA) are Timed Automata extended with probabilities. In PTAs edges can be split into branching edges where each branching edge has a probability attached to it.

Due to the various different definitions of a PTA by several people we will present a PTA containing all the elements of the various PTA definitions combined. We also give a table with the contents of each version of PTA. [12, 13, 25]

**Definition 4.2.1 (Probabilistic Timed Automata):**

A PTA is a tuple $(L, l_0, \Sigma, C, E, R, I, L)$ given by:

- $L$ is the set of locations/nodes.
- $l_0 \in L$ is the initial location.
- $C$ is a finite set of clocks.
- $\Sigma = \Sigma_i \cup \Sigma_o \cup \tau$ is a finite set of actions partitioned into inputs ($\Sigma_i$) and outputs ($\Sigma_o$) and the silent $\tau$ action.
- $E \subseteq L \times B(C) \times \Sigma \times P(Q_C \times L)$ is a finite set of edges from locations $l \in L$ with an action $\in \Sigma$, a clock constraint $\in B(C)$ and a probability distribution $P$ over the set of clocks to be reset and the target location $\in L$. This edge is unique.
- $I : L \to B(C)$ assigns invariants to locations.
- $R : L \to \mathbb{R}^+$ assigns a rate of exponential to locations.
- $L : L \to 2^{AP}$ is a labeling function assigning atomic propositions to locations. $AP$ is the set of all atomic propositions.

The model-checker Uppaal does not require that both an invariant and a rate of exponential is assigned to locations. This means that the functions $I$, $R$ and $L$ are partial functions.

PTA are defined in different ways by different people. Below we give a table what each definition does and does not have. Y = Yes. N = No. Delay density function $\mu_s$ and output probability distribution $\eta_s$ are explained in section 4.3.
The definition and the semantics of PTA were obtained from the sources given in the table.

We added TA to the list as reference. Baiers does not specifically call his automata PTA (just TA), but since his TA do have probabilistic semantics we call them PTA. For the definition of Priced Timed Automata see David’s paper [11].

**Definition 4.2.2** (PTA semantics):
The semantics of a PTA are given by a labeled Markov system \(<S, s_0, \rightarrow>\) where:
- \(S \subseteq L \times B(C)\) is the set of states.
- \(s_0 = (l_0, \nu_0)\) is the initial state.
- \(\rightarrow \subseteq S \times S\) is a transition relation such that:
  - \((l, \nu) \xrightarrow{d} (l, \nu + d)\) if \(\forall d' : 0 \leq d' \leq d \implies \nu + d' \models I(l)\);
  - \((l, \nu) \xrightarrow{a} X(\nu, p)\) if there exists \(e = (l, a, g, r, p, l') \in E\) s.t. \(\nu \models [r \mapsto \rightarrow 0] \nu, \nu' \models I(l)\)

The (target) random variable of \(\nu\) and \(p \in P(2^C \times L)\), named \(X(\nu, p)\) is defined as:

\[
P[X(\nu, p) = (l', \nu')] = \sum_{r | \nu' = [r \mapsto \rightarrow 0] \nu} p(r, l', \nu' \in U(C))
\]

In other words: A delay can only be added to a clock if the new clock value does not break the invariant. The system can only take a transition if there exists an edge \(e = (l, a, g, r, p, l') \in E\) if all possible target location invariants and the guard \(g\) hold and a random process chooses a target location according to a pre-specified probability distribution \(P\). In standard PTA the selection of delay \(d\) is done non-deterministically. In the next section we will discuss deterministic delay selection.

The difference between TA and PTA is that in PTA an edge transition can still branch out according to probability distribution \(P\) we mentioned earlier. An example can be seen in figure 4.2 below where an edge branches into 2 edges, each with a probability attached to it.

Note: In certain definitions of PTA the action set is excluded. When this is the case the semantics of PTA define an edge to be \(e = (l, g, r, p, l') \in E\).

An example of a PTA:

![Figure 4.2: A Probabilistic Timed Automaton.](image)

Figure 4.2 shows a probabilistic timed automaton. In this automaton a delay \(\leq 1\) is first drawn after which the PTA transitions to the unsafe location with a probability of 0.1. From the orange location there is a probability of 0.8 to transition to the safe location after drawing a delay \(\leq 1\).
Often PTA, just like regular TA, are composed into a network of timed automata (NPTA). NPTA also have a common set of clocks and actions and consists of \( n \) PTA. We will not go into this NPTA in this thesis. More information on NPTA can be found in [13].

4.3 Discussion: PTA probabilistic semantics

In standard TA and PTA delay selection happens non-deterministically (where given the same input multiple outputs can occur and where we do not know the probabilities of an output being generated) and because of this it is impossible to calculate the probability that a certain delay gets selected. When trying to simulate a PTA we need to attach a probability density function to locations to be able to calculate the probability that a certain delay is selected and to traverse an edge. We will call this probability density function the delay density function from now on. This delay density function is denoted with symbol \( \mu_s \) and depends on the state the system is in.

Note: A problem arising with the delay density function is that it gives a probability to delays that do not result in any enabled edges. This can lead to some odd behavior we will discuss later on in this thesis. After doing a literature study and a number of case studies in Uppaal we were able to understand what truly happens when a delay is selected that does not result in an enabled edge.

4.3.1 David, Uppaal

David, one of the creators of the Uppaal tool, in [12] defined priced timed automata (PrTA) in 2011. PrTA are regular timed automata where clocks can travel at various rates independent of each other (which we will not go into). David tries to simulate Priced Timed Automata by adding a uniform or exponential distribution to locations in order to draw a delay. This paper unfortunately does not show what happens in the case that there exists an area where no edge is enabled (guard gap, definition below).

David also defines a probabilistic measure over the set of all possible runs for networks of PrTA which we will not go into. The probabilistic measure for a single PrTA is similar to the one Baier gives in his section further below.

David defines:

Definition 4.3.1 (Delay density function):

Denote by \( E_l \) the disjunction of guards \( g \) such that \( (l, g, o, -, -) \in E \) for some output \( o \in \Sigma_o \).

Denote by \( \kappa(l, \nu) \) the infimum delay before enabling an output, i.e. \( \kappa(l, \nu) = \inf\{d \in \mathbb{R^+}|\nu + d \models E_l\} \), and denote by \( \omega(l, \nu) \) the supremum delay, i.e. \( \omega(l, \nu) = \sup\{d \in \mathbb{R^+}|\nu + d \models I(l)\} \).

Let \( \mu_s \) be a delay density function over delays in \( \mathbb{R}_{\geq 0} \).

If \( \omega(l, \nu) < \infty \) then the delay density function \( \mu_s \) is a uniform distribution over points in \( [\kappa(l, \nu), \omega(l, \nu)] \).

If \( \omega(l, \nu) = \infty \) the delay density function \( \mu_s \) is an exponential distribution with rate \( R(l) \).

Denote by \( \eta_s \) a probability function over output \( \Sigma_o \). For every state \( s = (l, \nu) \), \( \eta_s \) is the uniform distribution over the set \( \{o : (l, g, o, -, -) \in E \land \nu \models g\} \) whenever this set is non-empty.

In order to simulate PTA it is required to specify how much time the model should wait before leaving a location. This is achieved by assigning either an invariant or a rate of
exponential to a location and following what we listed under the definition of the delay density function. The model-checker Uppaal will generate an error and abort the simulation if it enters a location without both an invariant and a rate of exponential and wants to draw a delay. David unfortunately does not go into this special case.

We will assume the presence of either a rate of exponential or an invariant with an upper-bound for the remainder of this thesis.

Following up on David’s definition we give a few definitions and assumptions around guards and guard gaps, which we will use during the remainder of this thesis:

**Definition 4.3.2 (Guard range):**

To see which clock values can be selected and which not it is required to know the guard ranges. We define the range of a guard $g$ under valuation $\nu$ to be:

$$[\kappa(g, \nu), \omega(g, \nu)]$$

with $\kappa(g, \nu) = \inf\{d \in \mathbb{R}^+ | \nu + d \models g\}$ and $\kappa(g, \nu) = \sup\{d \in \mathbb{R}^+ | \nu + d \models g\}$.

We will denote guards with a single point as range, i.e. $\kappa(g, \nu) = \omega(g, \nu)$, as exact guards.

**Definition 4.3.3 (Guard gap):**

We define a guard gap $G(l, \nu) = \{d \in \mathbb{R}^+ | \nu + d \not\models E_l, \nu + d \models I(l)\}$ the set of points within the interval $[\kappa(l, \nu), \omega(l, \nu)]$ that can not be used to transition to a new location.

We denote delays that fall within the guard gap as guard gap delays, i.e. a delay is a guard gap delay iff $x + d \in G(l, \nu)$ for clock $x$.

**Assumption 4.3.1 ($\tau$ transitions):**

Davids $\mu_s$ assigns probabilities $> 0$ to values from the guard gap. Neither David nor Uppaal defines what happens when a delay is drawn that equals one of the points of the guard gap. From testing in Uppaal we have discovered that the model stays in the same location while still adding the drawn delays to all clocks. This is achieved without notifying the user. When no transitions can be done to a new location we assume that a silent transition is done while remaining in same location:

For all $d \in G(l, \nu)$ we add an edge $e = (l, \tau, g, \emptyset, 1, l)$ to $E$ such that guard $g$ equals the clock constraint $x = \nu + d$.

Note: By adding silent transitions $G(l, \nu)$ would become empty in theory. We will slightly abuse this by assuming that adding silent transitions does not alter $G(l, \nu)$.

Note 2: Because we "filled" up any guard gap this way (with empty action transitions) there is always an action transition possible after drawing a delay.

4.3.2 Baier, Bertrand

Baier and Bertrand were among the first to define a probabilistic measure over all possible runs around the year 2008. They achieved this for regular timed automata. However they allow the delay density function $\mu_s$ to take any shape and specify a probability measure over paths, thus allowing a model to transition to a new location with the same probabilities defined in a PTA if necessary. Baiers TA can thus be seen as a PTA if needed.

Baier in [6] defines the notion of non-blocking timed automata. The timed automaton $A$ is said non-blocking whenever for every state $s \in S$, $\Xi(s) \neq \emptyset$ with $\Xi(s) = \cup_e I(s, e)$ and with $I(s, e) = \{d \in \mathbb{R}_+ | s \xrightarrow{de} st\}$. $s$ and $st$ are states $\in S$. $S$ contains all possible combinations of locations and values from $\mathbb{R}_{\geq 0}$. $e$ is an edge from edge set $E$. In essence Non-blocking means that time-locks are not allowed. Time must always be able to be spend, or a transition must always be allowed to be taken.
Baier and Bertrand also define a probability measure over finite paths:

\[ P(\pi(s, e_1, \ldots, e_n)) = \int_{t \in I(s, e_1)} \eta_{s+t}(e_1) \mu_s(t) \, d\mu_s(t) \]

with \( \pi(s, \ldots) \) an unconstrained symbolic path and \( \eta_{s+t}(e_1) \) the probability to end up in state \( s+t \) via edge \( e_1 \). \( \mu_s \) is the probability measure over \( \mathbb{R}_+ \) with total probability \( \mu_s(\Xi(s)) = \mu_s(\mathbb{R}_+) = 1 \). \( \mu_s \) is not limited to the uniform and exponential, it can take any shape the user sees fit.

### 4.3.3 Kwiatkowska, Lanotte

Kwiatkowska et al., between 1999 and 2007, in [25, 21, 23] and Lanotte in [26] do not define a \( \mu_s, \eta_s \) or any probabilistic semantics. Kwiatkowska does present a method for symbolically model checking a PTA. She does this by looking at the minimum and maximum probabilities of entering a certain location. Delays are still chosen non-deterministically and essentially ignored in the model checking algorithm. The algorithm only looks at probabilistic branching edges, specified in \( P \), to calculate these minimum and maximum probabilities. Kwiatkowska et al. implemented their techniques in the model-checker PRISM.

Kwiatkowska also defines the following:

In state \((l, \nu)\), there is a nondeterministic choice of either (1) making a discrete transition or (2) letting time pass. In case (1), a discrete transition can be made according to any probabilistic edge \((l, g, \sigma, p) \in \text{prob} \) with source location \( l \) which is enabled; that is, the zone of guard \( g \) is satisfied by the current clock valuation \( \nu \). Then the probability of moving to the location \( l' \) and resetting all of the clocks in \( r \) to 0 is given by \( p(r, l') \). In case (2), the option of letting time pass is available only if the invariant from location \( l \) remains satisfied while time elapses and there does not exist an enabled probabilistic edge with an urgent event.

Under Kwiatkowska’s definition an execution run would become: \( \sigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n \) with \( \alpha_1, \alpha_2, \ldots, \alpha_n \in \Sigma \cup \mathbb{R}_{\geq 0} \). In the case that guard gaps are present this would mean that a new delay is chosen and added to all clock values when a delay is selected that falls within a guard gap.

Kaczmarczyks in [19] had a similar definition where she tells us that it must always be possible to do either a delay transition or an discrete transition at any point in time, non-deterministically chosen in a state \((l, \nu)\). David does reference this paper in his paper, but does not mention anything about Kaczmarczys or Kwiatkowska definition on discrete/delay transitions, he only mentions it in an example he used from Kaczmarczyk.

### 4.3.4 Conclusion

The three sources show that there are several different definitions for PTA. David allows guard gaps, but does not mention what should happen in the case of guard gaps. Uppaal, which is based on David’s paper, does an internal transition in the case a delay is drawn that falls within a guard gap. Kwiatkowska does allow several delay transitions to occur in a row, which could solve David’s problem with guard gaps.

Baier and Bertrand do not allow guard gaps at all, they only allow delays to be drawn that can be used to transition to a new location. Their version of the delay density function \( \mu_s \) basically excludes guard gaps.

For the remainder of this thesis we will base our work on the semantics of David.
4.4 Discrete Time Markov Chain

An other model that will be talked about in this thesis is the Discrete Time Markov Chain. A Discrete Time Markov Chain (DTMC) is a stochastic process (which is a collection of random variables) with a discrete state space that has the Markov property. \[1, 44\]

**Definition 4.4.1** (Discrete Time Markov Chains):
A discrete time stochastic process \( \{X_n, n = 0, 1, 2, \ldots\} \) with discrete state space is a Markov chain if it satisfies the Markov property:
\[
P(X_n = i_n | X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}) = P(X_n = i_n | X_{n-1} = i_{n-1}),\]
where \( i_k \) for all \( k = 0, 1, \ldots, n \) are realized states of the stochastic process.

**Definition 4.4.2** (One-step transition probabilities):
The one-step transition probability is the probability that the process, when in state \( i \) at time \( n \), will next transition to state \( j \) at time \( n + 1 \). We write \( p_{ij}^{n,n+1} = P(X_{n+1} = j | X_n = i) \).
1. \( 0 \leq p_{ij}^{n,n+1} \leq 1 \) since the transition probabilities are (conditional) probabilities.
2. \( \sum_{j=0}^{\infty} p_{ij}^{n,n+1} = 1 \) since the chain must transition somewhere and summing over all \( j \) is an application of the addition law for a set of disjoint and exhaustive events.

**Definition 4.4.3** (Time homogeneity):
When the one-step transition probabilities do not depend on time, so that \( p_{ij}^{n,n+1} = p_{ij} \) for all \( n \), then the one-step transition probabilities are said to be stationary and the Markov chain is also said to be stationary or time homogeneous.

**Definition 4.4.4** (One-step transition matrix):
The one-step transition matrix, \( P \), is formed by arranging the one-step transition probabilities into a matrix:
\[
P = \begin{pmatrix}
p_{00} & p_{01} & p_{02} & \cdots \\
p_{10} & p_{11} & p_{12} & \cdots \\
p_{20} & p_{21} & p_{22} & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{pmatrix}
\]

1. \( P \) is a square matrix, possibly of infinite dimension if the state space is countably infinite.
2. The rows sum to 1, by properties of one-step transition probabilities given above.

![Figure 4.3: A Discrete Time Markov Chain](image)
Figure 4.3 shows a Discrete Time Markov Chain. The one-step transition probabilities are attached to the edges between the states $S_0, S_1, S_2, S_3$. For more information on DTMCs see [1, 44].

Continuous Time Markov Chains (CTMCs) also exist: A CTMC is a stochastic process $\{X(t) : t \geq 0\}$ that satisfies the Markov property and is the continuous-time version of the DTMC, its time variable can take on any value $\in \mathbb{R}^+$. We wont go into CTMCs in detail.
Chapter 5

Uppaal

Uppaal is a tool used to verify real-time systems and is jointly developed by employees of the universities of Uppsala and Aalborg. The tool was designed to be able to verify (networks of) timed automata. Uppaal also allows integer variables, structured data types, and channel synchronization to be specified in each (network of) timed automata.

Uppaal was initially released in 1995 and has seen many improvements over the last few years. The most noteworthy improvements for us are priced and probabilistic timed automata, a statistical model checker and a plot composer.

The model checker Uppaal is based on the theory of timed automata. Uppaal uses a query language to specify the properties that need to be checked. This query language is a subset of CTL (computation tree logic). In this section we will explain how Uppaal works. Assumptions and possible strange behavior from Uppaal is also given.

A part of the contents in this chapter comes from the Uppaal help function and [8].

5.1 The Uppaal simulation algorithm

Assumption 5.1.1 (Model assumptions):
The following assumptions about the PTA have to be made:

- Input enableness (non-blocking inputs) [11]: Sending cannot be blocked, i.e. the channel is either broadcast or there is always one process with an enabled receiving edge-transition:
  For all states \((l,ν)\) and input actions \(i \in Σ\), there is an edge \((l,g,i,r,l')\) such that \(ν \models g\)

- Input determinism [11]: There is exactly one enabled receiving edge-transition at a time (edges with an input action attached). For binary synchronizations there is at most one receiving process at a time:
  For all states \((l,ν)\) and actions \(a \in Σ\), whenever \((l,ν) \xrightarrow{a} (l',ν')\) and \((l,ν) \xrightarrow{a} (l'',ν'')\) then \(l' = l''\) and \(ν' = ν''\) where \(l,l',l''\) are locations and \(ν,ν',ν''\) are valuations.

- Non-Zenos: Time always diverges. It can not get “stuck” under a constant time value. In essence this also excludes time-locks. A time-lock is when a clock has reached the upperbound of an invariant and can not transition to a new location.

- Well-formedness [25]: When taking an edge to a location the location invariant must always be true. In essence the invariants range must always overlap the combined range of the guards from all the incoming edges, or the clock must be reset to a value that satisfies the invariant:
  For each edge \((l,ν) \xrightarrow{r} (l',ν')\) and action \(a\) such that \(ν \models G(e)\) it holds that \(ν[r :=
If a PTA is not well-formed then it can be transformed into a well-formed PTA. For more information on this transformation, see \cite{22}.

**Definition 5.1.1 (Executions/runs):**

During each transition either an edge \( e \in E \), with an action, is taken or a delay is drawn (from \( \mathbb{R}_+ \)) and added to all clocks.

We define a run \( i \) to be: \( \sigma_i = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n \) with \( a_1, a_2, \ldots, a_n \in E \cup \mathbb{R}_+ \).

In the case that guard gaps are present there is a possibility that multiple delays are selected in a row without an edge being traversed.

If there are urgent/committed locations in the model there is a possibility that multiple edges are traversed in a row without delays being selected.

**Definition 5.1.2 (Urgent/committed locations):**

There are two kinds of locations that are used by Uppaal that can alter the delay selection process.

Urgent locations freeze time; i.e. time is not allowed to pass when a process is in an urgent location. Semantically, urgent locations are equivalent to:
- Adding an extra clock, \( x \), that is reset on every incoming edge, and
- Adding an invariant \( x \leq 0 \) to the location.

![Figure 5.1: The orange locations are urgent locations](image)

Like urgent locations, committed locations freeze time. Furthermore, if any process is in a committed location, the next transition must involve an edge from one of the committed locations. Committed locations are useful for creating atomic sequences and for encoding synchronization between more than two components. Notice that if several processes are in a committed location at the same time, then they can interleave.

### 5.1.1 Properties

During the simulation of a specified model, Uppaal monitors the clock values and the current location. Using these values Uppaal checks whether or not the, by the user, specified properties are eventually satisfied or not.

Properties like \( P[x \leq c]\langle \psi \rangle \) are checked by Uppaal. The meaning of the property is whether or not the model has satisfied expression \( \psi \) before clock \( x \) has reached \( c \). Expression \( \psi \) can be a location but also expressions like \( x < 5 \) that evaluate to true or false depending on the value of \( x \).

For more information on properties see chapter 9.

### 5.1.2 The algorithm

Uppaal tries to simulate a model by executing the model a fixed number of times. How often Uppaal executes a model depends on certain settings the user has to supply. When
executing a model Uppaal starts in the initial location and starts traversing the model according to the semantics of PTA until it can stop. The algorithm can also be found in the Uppaal help function [3]. We did take the courtesy to add a few details to make the reader understand the algorithm better. What Uppaal does in each location is given in the following algorithm:

1. Uppaal runs a number of checks to determine if a property holds or not. Stop if a property holds or when we can no longer continue traversing trough the PTA (when the deadlock property is set to true by Uppaal).

2. The PTA chooses a delay based on its current location:
   - If the current location is committed/urgent location then the concrete delay is zero.
   - If the current location has an invariant with an upperbound, then the concrete delay is taken according to the uniform distribution from the current clock value up to the upperbound of the invariant. We will call this the range of the invariant.
   - If no invariant is present, or if the invariant does not have an upperbound then the delay is chosen by using the exponential distribution with rate \( \lambda \) specified on the current location. The probability density function of delay \( d \in [0, \infty) \) is \( F(d) = \lambda e^{-\lambda d} \). The concrete delay is generated by \(-\frac{\ln(u)}{\lambda}\) where \( u \) is a uniform random number from the interval \((0,1]\).
   - If no invariant and no rate of exponential is present Uppaal will generate an error and stop the simulation without generating an estimate.

3. The delay is executed and all clocks are updated.

4. The PTA will attempt to take a transition:
   - Check which edges are enabled, i.e. check which edge guards evaluate to true.
   - Pick an enabled edge randomly using the uniform distribution and:
   - If the edge has probabilistic branches, then the probability of taking a branch \( i \) is determined by the ratio \( \frac{w_i}{W} \), where \( w_i \) is weight of the branch \( i \) and \( W \) is the sum of all branch weights: \( W = \sum_{j=1}^{n} w_j \) for \( n \) branches.

5. If no guards are enabled, after a delay is chosen, then several things, which we observed from testing in Uppaal, can happen depending on the model:
   - If an invariant with an upperbound is present and a guard can still be made true the system goes back to step 1 and chooses a new delay.
   - If an invariant with an upperbound is present and there are no guards left that can be made true then Uppaal shows a time lock error with no end result for the specified simulation property.
   - If a rate of exponential is present (and no invariant or no invariant upperbound) and guards can still be made true the system goes back to step 1.
   - If a rate of exponential is present (and no invariant or no invariant upperbound) and no guards are left that can be made true then Uppaal makes the model deadlock.

Note: Weights \( w_i \) in Uppaal are equal to the \( p(r, l') \) in standard PTA, which are given by the probability distribution \( P(2^C \times L) \). For more info on \( P(2^C \times L) \) see the semantics of PTA in chapter 3.
5.2 Strange behavior

A lot of the details about simulation for PTA are not fully specified. This can give problems when trying to perform importance sampling.

5.2.1 Simulation

Uppaal simulates the model by repeatedly picking delays and transitions. In standard timed automata and standard PTA this is done nondeterministically. In simulation this is done either by using the uniform distribution or by using the exponential distribution. What distribution is going to be used has to be specified by the user of the Uppaal tool. When choosing a delay nondeterministically it is possible to choose an exact value, for example \( x = 5 \). When using simulation, specifically the uniform or exponential distribution, this is not possible. This can be a problem since some system behavior can no longer be performed.

![Figure 5.2: Adding a distribution prohibits the system from entering Finish](image)

In theory it is possible to use delay density functions other than the uniform/exponential distribution. However we will focus on the uniform and exponential distribution used in Uppaal.

5.2.2 Guard gap delays

During simulation a delay is chosen based on a specified distribution, usually either the uniform distribution or the exponential distribution. When a delay is chosen and no guards are enabled, then a new delay is re-chosen. When this happens the total delay is not truly drawn according to the initial uniform distribution. This does not sound logical as it would mean that some values are more likely to be chosen than others, something which conflicts with the nature of the uniform distribution.

When there is always a guard enabled then the delay is truly drawn according to the uniform distribution specified on the location. So no re-drawing has to be done.

5.2.3 Probability Estimation (Quantitative Model Checking)

With the probability estimation test, Uppaal tries to estimate the probability of a path property being true. Uppaal does this by computing the number \( N \) of runs needed in order to produce an approximation interval \([p - \epsilon, p + \epsilon]\) for \( p = Pr(\psi) \) with a confidence of \( 1 - \alpha \) which indicates the reliability of the estimate. The values for \( \epsilon \) and \( \alpha \) can be set, by the user, in the statistical parameters window via the options tab. [12], which points us to paper [18], tells us that the algorithm computes \( N \), with parameters \( \epsilon, \alpha \) and confidence \( 1 - \alpha \).

Paper [12] tells us that the formula to compute \( N \) is:

\[
N \geq \frac{4\ln\frac{1}{\alpha}}{\epsilon^2}
\]  

(5.1)

Paper [18] tells us that the formula is:

\[
N \geq \frac{4\ln\frac{2}{\alpha}}{\epsilon^2}
\]  

(5.2)

31
However after running a number of simulations in Uppaal we found out that the actual formula used in Uppaal was:

\[ N \geq \frac{\frac{1}{2} \ln \frac{2}{\alpha^2}}{e^2} \]  

(5.3)

We knew what the formula was for computing \( N \). We found this discrepancy by calculating the expected \( N \) using the first formula and comparing it to the results from Uppaal, which were too low. We found the exact formula Uppaal uses by playing around with the values for \( \alpha \) and \( \epsilon \), while keeping the model the same.

When we altered \( \epsilon \) from 0.01 to 0.1 we found out that \( N \) decreased 100 times, which satisfies the above formulas.

When we altered \( \alpha \) from 0.01 to 0.1 we found out that \( N \) did not decrease by a factor \( \frac{\ln(100)}{\ln(10)} \), so we knew the log-part of the first formula was wrong.

We found out that the second formula above did decrease \( N \) by the right factor, namely \( \frac{\ln(200)}{\ln(20)} \).

From these results we were also able to deduce that \( N \) was off by a factor 8 in general compared to the second formula.

We have not been able to find out why these discrepancies exist.
Chapter 6

Rare event simulation

Importance sampling is a technique used to estimate the probability of entering a certain location by sampling from a different distribution and then correcting for the error introduced. This chapter will give an introduction into the basics of rare event simulation and especially importance sampling and apply it to PTA.

6.1 Introduction

In this section we will discuss the basics of importance sampling. We will do this by first informing the reader about Monte Carlo simulation and rare event simulation followed by the basics of importance sampling.

6.1.1 Monte Carlo Simulation

Monte Carlo simulation \[29, 43\] is a simulation technique where a model or situation gets simulated by executing the model/situation a large number of times using the same initial state. Each time the course and in turn the result of the simulation depends on a number of random variables defined in the model/situation itself. Running a simulation once might not guarantee an accurate result, but running it a large number of times can greatly increase the accuracy of results.

Consider a random variable \(Y\) having a probability density function \(p(y)\). We can calculate the expected value of a function \(f(y)\) with respect to \(p(y)\) analytically using the function:

\[
E_p[f(Y)] = \int_\mathbb{R} f(y)p(y) \, dy
\]

(6.1)

It can be quite hard to solve mathematical problems, like the integral shown above, when \(p(y)\) behaves weird in nature. For that reason it is sometimes easier to just take \(n\) samples \((y_1, \ldots, y_n)\) from the distribution \(p(y)\). We denote by \(\tilde{\gamma}\) the Monte Carlo estimate and \(\gamma = E_p[f(Y)]\) the value we are trying to estimate. We can estimate the mean of a function \(f(Y)\) with:

\[
E_p[f(Y)] \approx \frac{1}{n} \sum_{i=1}^{n} f(y_i) \triangleq \tilde{\gamma}
\]

(6.2)

Event simulation

In (rare) event simulation \[17\] one tries to estimate the probability that an event is triggered by executing some model. This can be achieved in a similar manner as we did for Monte Carlo simulation in the previous section. Earlier in this thesis we defined \(\sigma_i\) to
be run \(i\). Sample \(\sigma_i\) is generated from probability distribution \(p\). We denote by \(\text{fail}\) a (possibly rare) failure event. We denote:

\[
1_{\text{fail}}(\sigma_i) = \begin{cases} 
1 & \text{if run } \sigma_i \text{ ends in fail} \\
0 & \text{otherwise}
\end{cases}
\]

When trying to estimate the percentage of executions \(\sigma_i\) that end in a (rare) \(\text{fail}\) location one can use the formula:

\[
\hat{\gamma} = \frac{1}{N} \sum_{i=1}^{N} 1_{\text{fail}}(\sigma_i) 
\]

(6.3)

Where \(\sigma_i\) denotes the \(i\)th sample, also called execution or run later in this thesis. The exact solution is:

\[
\gamma = \int_{D_\sigma} 1_{\text{fail}}(\sigma) \, dp(\sigma) = E_p[1_{\text{fail}}(\sigma)]
\]

(6.4)

\(p(\sigma)\) denotes the probability distribution of some path \(\sigma\). Normally one would draw \(N\) samples \(\sigma_1, \ldots, \sigma_N\) to estimate \(\hat{\gamma}\). When \(p(\sigma)\) is such that \(\gamma\) is small then it would require a lot of samples to be run to get an accurate estimate. \(D_\sigma\) is the domain of \(\sigma\).

6.1.2 Importance Sampling

Importance sampling (IS) [10, 5, 39, 37, 17] is a technique used for approximating \(E_p[1_{\text{fail}}(\sigma)]\) by not drawing samples using \(p(\sigma)\), but from a different distribution \(q(\sigma)\), usually with \(E_q[1_{\text{fail}}(\sigma)] > E_p[1_{\text{fail}}(\sigma)]\). Sampling from a different distribution is called a ”change of measure”. The error made from sampling from another distribution is corrected afterwards by using a weighting function. To draw samples from \(q(\sigma)\) we have to change formula (6.4) slightly by multiplying and dividing by \(q(\sigma)\) within the integral part:

\[
\gamma = \int_{D_\sigma} 1_{\text{fail}}(\sigma) \frac{p(\sigma)}{q(\sigma)} \, dq(\sigma) = E_q[1_{\text{fail}}(\sigma) \frac{p(\sigma)}{q(\sigma)}] = E_q[1_{\text{fail}}(\sigma) \frac{L(\sigma)}{q(\sigma)}] 
\]

(6.5)

\(L(\sigma) = \frac{p(\sigma)}{q(\sigma)}\) is the weighting function which we usually call the likelihood ratio. \(\hat{\gamma}\) becomes (sampling from \(q(\sigma)\)):

\[
\hat{\gamma} = \frac{1}{N} \sum_{i=1}^{N} 1_{\text{fail}}(\sigma_i) L(\sigma_i)
\]

(6.6)

For individual samples \(i\) the likelihood ratio becomes \(L(\sigma_i) = \frac{p(\sigma_i)}{q(\sigma_i)}\), where \(p(\sigma_i)\) and \(q(\sigma_i)\) denote the probability distributions of an execution with path \(\sigma_i\) happening under their respective distributions \(p(\sigma)\) and \(q(\sigma)\).

6.1.3 Importance Sampling Example

Take for example a model using coin flips with \(Pr[\text{tails}] = Pr[\text{heads}] = 0.5\). If we assume that ten times tails is a failure event then the probability of having a failure event is \(0.5^{10} = \frac{1}{1024}\). If we would want to have an accurate estimation of this value using Monte Carlo simulation we would have to run thousands of simulations.

If we would use importance sampling and increase the probability that a coin flip results in tails we can in theory reduce the number of simulations needed to accurately estimate the probability of a failure event. If we take \(Pr[\text{tails}] = 1 - Pr[\text{heads}] = 0.9\) as the change of measure \(q(\sigma_i)\) for coin flip \(i\) we can apply IS using the following formula:
\[ \tilde{\gamma} = \frac{1}{N} \sum_{i=1}^{N} 1_{\{\text{All heads}\}} L(\sigma_i) \]  

(6.7)

The value under the sum is \(0 \times L(\sigma)\) when a simulation results in a good event and \(1 \times L(\sigma)\) when the simulation ends with a failure event. Because of this the likelihood ratio only matters when the simulation results in a failure event. The likelihood ratio for an execution \(\sigma_i\) that results in all heads is (execution denoted \(\sigma_{\text{fail}}\)):

\[ L(\sigma_{\text{fail}}) = \frac{p(\sigma_{\text{fail}})}{q(\sigma_{\text{fail}})} = \frac{0.5^{10}}{0.9^{10}} = \left(\frac{0.5}{0.9}\right)^{10} \]

Using this value we can reduce the problem to:

\[ \tilde{\gamma} = \frac{1}{N} \sum_{i=1}^{N} 1_{\{\text{All heads}\}} \left(\frac{0.5}{0.9}\right)^{10} = \left(\frac{0.5}{0.9}\right)^{10} \frac{1}{N} \sum_{i=1}^{N} 1_{\{\text{All heads}\}} \]  

(6.8)

In (6.8) we simulate \(N\) times ten coin flips using the change of measure \(q(\sigma)\).

**Coin flip example results**

We wrote a very simple program that estimated the probability that 10 coin flips resulted in 10 tails (formula (6.8)) using the earlier mentioned probabilities. This program gave the following results after running one million sample executions:

- Analytical value: \(0.5^{10} \approx 9.766 \times 10^{-4}\)
- IS Estimate: \(9.778 \times 10^{-4}\)
- IS Confidence: \([9.751 \times 10^{-4}, 9.804 \times 10^{-4}]\)
- IS Sample Variance: \(0.01782 \times 10^{-4}\)
- MC Estimate: \(9.740 \times 10^{-4}\)
- MC Confidence: \([9.129 \times 10^{-4}, 10.351 \times 10^{-4}]\)
- MC Sample Variance: \(9.731 \times 10^{-4}\)

We can see that the analytical value fell within both confidence intervals and that importance sampling was a large improvement over Monte Carlo simulation. Confidence and variance measures are explained in the next section.

**6.1.4 A variance reduction**

When using simulation we want the variance to be as low as possible, because then we require fewer samples to be drawn in total. In normal Monte Carlo simulation (formula (6.3)) the variance of the estimator of \(\tilde{\gamma}\), which is \(N\) times smaller than the variance for a single sample, is given by

\[ \gamma(1 - \gamma)/N \]

This value unfortunately depends on the unknown value, so we have to use the sample variance:

\[ S^2 = \frac{\sum_{i=1}^{N} (1_{\text{fail}}(\sigma_i) - \tilde{\gamma})^2}{(N-1)}, \]

for \(N\) samples \(\sigma_i\).

In Importance Sampling the variance is based on both \(p(\sigma)\) and \(q(\sigma)\). The \(p(\sigma)\) distribution cannot be altered since it is the distribution used to generate an estimate. It is however possible to use any distribution \(q(\sigma)\) we want under certain conditions.
use a $q(\sigma)$ in such a way that it minimizes the variance of the estimate. This distribution is called the optimal distribution or optimal change of measure and we will denote this by $q^*(\sigma)$. Selecting $q^*(\sigma) = \frac{p(\sigma)}{\gamma}$, where $1_{\{\sigma \in \text{fail}\}} = 1$ every run, gives a variance of zero, meaning that drawing even one sample would give mean $\tilde{\gamma} = \gamma$. When the variance is zero every sample drawn would give $\gamma$ exactly, so we would only need to generate one sample to get the right answer. But all this requires knowing the value we are trying to estimate and the correct path. If we would know this information then we wouldn’t need to simulate at all. So in essence the optimal distribution is not feasible and we have to estimate and the correct path. If we would know this information then we wouldn’t need sample to get the right answer. But all this requires knowing the value we are trying to zero every sample drawn would give $\gamma$ meaning that drawing even one sample would give mean $\tilde{\gamma} = \gamma$. The transition during the run the likelihood ratio gets updated: $p$ to either $\sigma_i$ to compute the likelihood ratio. When using simulation to run through a PTA the likelihood ratio must be updated for each transition. Earlier we represented a run as $\sigma_i = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_m} s_n$ with $\alpha_1, \alpha_2, \ldots, \alpha_m \in E \cup \mathbb{R}_+$. States $s_i$ for step $i$ are generated according to either $p(\sigma)$ or $q(\sigma)$. $m$ is a random variable denoting the length of the path. For each transition during the run the likelihood ratio gets updated:

$$L(\sigma_i) = \prod_{j=1}^{m} \frac{P[s_j, s_{j+1}]}{Q[s_j, s_{j+1}]}.$$
where $P$ respectively $Q$ contain the probabilities that a transition is taken from one state to another under the original distribution $p(\sigma)$ respectively the change of measure $q(\sigma)$.

### 6.2.1 The likelihood ratio

As we have seen earlier the likelihood ratio is calculated upon the probabilities of taking a certain transition is made. By looking at the semantics of PTA we are able to differentiate between two kinds of transitions: delay transitions and action transitions. An action transition can still branch out to multiple locations. During the simulation of PTA a delay gets drawn according to either the uniform or the exponential distribution and all clocks "transition" to their new values. After a delay has been drawn an enabled edge gets selected for transitioning to a new location according to the discrete uniform distribution. The selected edge can branch out according to distribution $P$ as discussed in section 4.2.

In order to make it easier for us to apply importance sampling we will split the likelihood ratio into two parts: Branch probability $P_j(B)(e)$ or $Q_j(B)(e)$ is the probability that edge $e$ at step $j$ branched out to location $l$. Jump probability at step $j$ is the probability $P_j(J)(e,s)$ or $Q_j(J)(e,s)$ that edge $e$ is taken from location $l$ after a delay has been drawn and edge $e$ has been uniformly selected. We can use these probabilities to alter the likelihood ratio formula:

$$L(\sigma_i) = \frac{\prod_{j=1}^{m} P[s_j, s_{j+1}]}{Q[s_j, s_{j+1}]} = \prod_{i=1}^{m} \left[ \frac{P_j(B)(e)}{Q_j(B)(e)} \times \frac{P_j(J)(e,s)}{Q_j(J)(e,s)} \right].$$

The symbols $P$ and $Q$ contain the probabilities that a transition is taken from one state to another under $p(\sigma)$ or $q(\sigma)$. These probabilities are state-dependent. When the probabilities under the new change of measure are unaltered the result of each factor remains 1. In the next few chapters we will be discussing how we can create the change of measure using various techniques. We will call $\frac{P_j(B)(e)}{Q_j(B)(e)}$ the branch likelihood ratio and $\frac{P_j(J)(e,s)}{Q_j(J)(e,s)}$ the jump likelihood ratio. We excluded the valuation information in the above formula as the probability that a certain delay is drawn, which alters the clock valuation, is separated from the probability that a transition is taken to and from a certain location. This will be discussed in the delay biasing chapter later on in this thesis.

The **branch change of measure**

Forming the change of measure for branch probabilities ($Q_j(B)(e)$) works in a similar manner as in importance sampling for DTMCs. We increase the probability that edge $e$ branches to location $l$ and lower the probabilities of the other outgoing branches to other locations to keep the total probability at 1. The manner the other branches’ probabilities are lowered does not matter as long as the probability remains larger than zero. The probability can be fixed/predetermined or a ratio (multiplying other outgoing branch probabilities with $\frac{1-p}{1-\epsilon}$, with $p$ the new probability and $\epsilon$ the old probability).

The **jump change of measure**

The change of measure for jump probabilities ($Q_j(J)(e,s)$) depends on the distribution attached to the location with the rare transition. If an invariant is present, one normally uses the uniform distribution on the interval of the current clock value up to the invariant-bound. When an invariant is not present on the location then Uppaal requires the user to supply an exponential exit rate for that location. A transition can be rare when the
invariant-bound is large/exit rate is low compared to the guard-bound or when the guard falls in the tail of the exponential distribution when an invariant is absent. For a change of measure to be good it would be required to increase the probability that said transition is taken. A technique called delay biasing where we force a specific edge to be taken more often is discussed later in this thesis which can make it easier to enter certain locations.

Other distributions than the uniform or exponential distribution are allowed to be used, like a beta distribution for example, since this does not alter the model. Combining invariants with distributions other than the uniform distribution is also possible. In this thesis we will not discuss those other distributions due to time-constraints.
Part II

New research
Chapter 7

The Uppaal semantics

Research question: How is the relation between the practical use of the uniform and exponential distributions and UPPAAL’s implementation of them?

Currently Uppaal uses either the uniform (invariant present) or the exponential distribution (invariant absent, rate of exponential present) to draw delays. Which distribution is used depends on what the user of Uppaal specifies on a location. These distributions that Uppaal uses differ slightly from the uniform/exponential distributions one might normally use when guard gaps are present. How much they relate to each other is discussed in this chapter. We will discuss this by first going into both the uniform and exponential distributions separately and then by looking at a number of case studies we obtained from a literature study. The situation in which no guards are enabled after drawing a delay is also discussed.

The semantics discussed in this chapter will form the basis for the importance sampling techniques discussed in the remainder of this thesis.

7.1 Preliminary

The semantics in the current and following chapters will follow the assumptions and declarations made in sections 1.5, 4.2, 4.3, 5.1.

We will first focus ourselves on guards with the form \( x \sim k \) with \( \sim \in \{ \leq, <, =, >, \geq \} \). Later in this chapter in section 7.4 we will discuss conjunctions of clock constraints as well. We will also focus ourselves on PTA with only one clock.

For the remainder of this thesis we will only consider guards that can be made true, i.e. guards \( g \) such that \( \{ d \in \mathbb{R}^+ | \nu + d \models g \} \neq \emptyset \). Guards that can not be enabled by drawing a delay are unimportant as they do not affect the delay density function.

During the course of this chapter we will be using a large number of ”min” and ”max” functions which denote either the minimum of two values or the maximum of two values. The number of ”min”/”max” functions we use within a single function is high due to the fact that we need to limit the shown distribution function to a given interval (usually \([\kappa(l, \nu), \omega(l, \nu)]\)). Values outside this interval do not have probability so in order to prevent those values from being counted we used the ”min”/”max” functions to limit ourselves to only the values that have a probability.
7.2 The uniform distribution

In this section we will discuss both the Uppaal uniform distribution and the regular uniform distribution and their delay distribution/density functions.

David in [12] denotes $\mu_s$ as a probability density function that follows either the uniform or the exponential distribution on $[\kappa(l, \nu), \omega(l, \nu)]$ for state $s = (l, \nu) \in S$, location $l$, valuation $\nu$. We will denote $M_s(K)$ the delay distribution function, which is the delay density function $\mu_s$ we talked about earlier integrated on interval $K = [a, b]$ where $a, b \in \mathbb{R}$. Interval $K$ corresponds with a variable area $[a, b]$ within the delay distribution function and is used to determine the probability that a delay is drawn that falls within the interval $K$. We will denote the delay density function $\lim_{\delta \to 0} \frac{M_s([t, t+\delta])}{\delta}$ in the current and following chapters.

7.2.1 The regular uniform distribution

The uniform distribution is a continuous probability distribution. The support in the context of PTA is defined by the two parameters, $\kappa(l, \nu)$ and $\omega(l, \nu)$, which are the distribution’s minimum and maximum values (explained in the next section and in section 4.2). All intervals of equal length within the support of the distribution are equally probable. The delay distribution function $M_s$ for interval $K = [a, b]$ is:

$$M_s(K) = \frac{\min(b, \omega(l, \nu)) - \max(a, \kappa(l, \nu))}{\omega(l, \nu) - \kappa(l, \nu)}$$

With the delay density function:

$$\lim_{\delta \to 0} \frac{M_s([t, t+\delta])}{\delta} = \begin{cases} \frac{1}{\omega(l, \nu) - \kappa(l, \nu)} & \text{for } \kappa(l, \nu) \leq t \leq \omega(l, \nu); \\ 0 & \text{Otherwise.} \end{cases}$$

Applications

If we look at real-life applications of the uniform distribution, we can find out that the uniform distribution is often used in pseudo random number generators and when trying to sample from an arbitrary distribution using the inversion method.

In order to simulate a model, a distribution must be attached, often the uniform distribution is used in PTA when it is not known what the real distribution is.

7.2.2 The Uppaal uniform distribution

The PTA in Uppaal are based on the ideas discussed by David in [12]. We have discussed this already in section 4.3 and chapter 5 and we will not go into the semantics again.

The distribution

From running a number of tests in Uppaal we found out that when an invariant is present on a location Uppaal chooses a delay randomly using the uniform distribution on the range of the invariant. However if Uppaal draws a delay such that no guard is enabled under the new clock valuation then Uppaal will try again. These delays still get added to the clock values (possibly multiple times). Because Uppaal may have to try picking a delay several times the probability that the delay satisfies a guard is not truly uniformly distributed. An example of Uppaal’s behavior is shown in figure 7.1.
We also define a merged guard to be the union of two or more guards and denote it \( \vec{g} \).

In the above example the probability that a guard \( g \) is evaluated to true after picking a proper delay depends on where \( g \) is located. The probability that a delay is drawn that falls within a guard gap gets added evenly to the remainder of the delay density function. If a guard comes after the guard gap it has an increased probability of being selected.

Earlier we assumed that guards have the form \( x \sim k \) with \( k \in \{ \leq, <, =, \geq \} \). All guards of the form \( x \leq c \) or \( x < c \) overlap due to not having a lower bound. The same holds for the upper bound of guards of the form \( x \geq c \) or \( x > c \). We can merge any overlapping guards into two merged guards, one of the form \( x \leq c \) or \( x < c \) and one of the form \( x \geq d \) or \( x > d \). If \( c < d \) then a guard gap is in between the two merged guards. If no guard gap is present the formulas from the regular uniform distribution can be used. For the remainder of this section we will assume the presence of a guard gap.

**Definition 7.2.1 (Overlapping guards):**

Multiple guards can overlap. We say that a guard \( g_1 \) overlaps with another guard \( g_2 \) if one of the following 4 situations applies:

1. \( \kappa(g_1, \nu) \leq \kappa(g_2, \nu) \leq \omega(g_1, \nu) \leq \omega(g_2, \nu) \)
2. \( \kappa(g_1, \nu) \leq \kappa(g_2, \nu) \leq \omega(g_2, \nu) \leq \omega(g_1, \nu) \)
3. \( \kappa(g_2, \nu) \leq \kappa(g_1, \nu) \leq \omega(g_2, \nu) \leq \omega(g_1, \nu) \)
4. \( \kappa(g_2, \nu) \leq \kappa(g_1, \nu) \leq \omega(g_1, \nu) \leq \omega(g_2, \nu) \)

**Definition 7.2.2 (Merged guards):**

We define a merged guard to be the union of two or more guards and denote it \( \vec{g} \). In addition we do not require all guards in \( \vec{g} \) to overlap with all other guards in \( \vec{g} \). We only require each guard in \( \vec{g} \) to overlap with one other guard in \( \vec{g} \).

We also define \( \kappa(\vec{g}, \nu) = \inf \{ d \in \mathbb{R}^+ | \nu \leq d \} \) as the lower bound of the merged guard \( \vec{g} \) and \( \omega(\vec{g}, \nu) = \sup \{ d \in \mathbb{R}^+ | \nu + d = \vec{g} \} \) as the upper bound of the merged guard \( \vec{g} \).

An example of two overlapping guards is: \( g_1 = x \leq 2 \) and \( g_2 = x \leq 4 \) with current state is \( s = (l_0, \nu(x) = 0) \). This results in: \( \kappa(g_1, \nu) = \kappa(g_2, \nu) = 0 \) and \( \omega(g_1, \nu) = 2, \omega(g_2, \nu) = 4 \).

**Definition 7.2.3 (Merged guards \( \vec{g}_1, \vec{g}_2 \)):**

Consider two guards \( g_1, g_2 \) of the form \( x \leq c \) or \( x < c \) for some constant \( c > \kappa(l, \nu) \) and some clock \( x \in C \). The infimum of said guards is equal (\( \inf(g_1) = \inf(g_2) = \kappa(l, \nu) \)). For this reason we can merge both guards into a single merged guard \( \vec{g}_1 \). If more guards of such form exist we also merge them into \( \vec{g}_1 \).

The same reasoning holds for guards of the form \( x \geq c \) or \( x > c \) and the supremum being equal (with each other and \( \omega(l, \nu) \)), which we can merge into a single merged guard \( \vec{g}_2 \).

If no guards of the form \( x \leq c \) or \( x < c \) exist we say that \( \vec{g}_1 \) does not exist.
If only one guard \( g \) of the form \( x \leq c \) or \( x < c \) exists we say \( g = ĝ_1 \).
If only one guard \( g \) of the form \( x \geq c \) or \( x > c \) exists we say \( g = ĝ_2 \).
As we assumed that time-locks are forbidden \( ĝ_2 \) must exist.

We are able to construct a delay distribution function for both merged guards and then merge them together into a single distribution function \( M_s(K) = M_s^{(1)}(K) + M_s^{(2)}(K) \) to make it easier for the reader to understand what is happening. \( M_s^{(1)}(K) \) corresponds with the delay distribution function of merged guard \( ĝ_1 \). \( M_s^{(2)}(K) \) corresponds with \( ĝ_2 \).

**Delay distribution** \( M_s^{(1)}(K) \)

The formula for the uniform delay distribution distribution for the interval \( K = [a, b] \) when \( ĝ_1 \) exists becomes:

\[
M_s^{(1)}(K) = \frac{\max(\kappa(ĝ_1, \nu), \min(b, \omega(ĝ_1, \nu))] - \min(\omega(ĝ_1, \nu), \max(a, \kappa(ĝ_1, \nu)))}{\omega(l, \nu) - \kappa(l, \nu)}
\]

With the delay density function:

\[
\lim_{\delta \to 0} \frac{M_s^{(1)}([t, t + \delta])]}{\delta} = \begin{cases} \frac{1}{\omega(l, \nu) - \kappa(l, \nu)} & \text{when } v(x) = t \models ĝ_1; \\ 0 & \text{Otherwise.} \end{cases}
\]

These formulas are similar to the uniform distribution without guard gaps. Which is obvious since no guard gaps are present before \( \omega(l, \nu) \).

Note: When \( ĝ_1 \) does not exist we say that both \( M_s^{(1)}(K) \) and its derivative are zero. This will also hold for following merged guards in the remainder of this paper

**Delay distribution** \( M_s^{(2)}(K) \)

The total probability to select a delay is one. Delays that fall within the guard gap can not be selected. For this reason the remainder of the probability, which we denote \( H = 1 - M_s^{(1)}(\kappa(ĝ_1, \nu), \omega(ĝ_1, \nu)) \), should be the probability to select a delay that satisfies \( ĝ_2 \). The formula for \( ĝ_2 \) can be created by taking the \( M_s \) for the uniform distribution without guard gaps, limit it to the range of merged guard \( ĝ_2 \) and multiplying it with \( H \) to set the probability of selecting a delay satisfying \( ĝ_2 \) to \( H \).

The formula for \( ĝ_2 \) becomes:

\[
M_s^{(2)}(K) = \frac{\max(\kappa(ĝ_2, \nu), \min(b, \omega(ĝ_2, \nu))] - \min(\omega(ĝ_2, \nu), \max(a, \kappa(ĝ_2, \nu)))}{\omega(ĝ_2, \nu) - \kappa(ĝ_2, \nu)} \times H
\]

With:

\[
\lim_{\delta \to 0} \frac{M_s^{(2)}([t, t + \delta])]}{\delta} = \begin{cases} \frac{H}{\omega(ĝ_2, \nu) - \kappa(ĝ_2, \nu)} & \text{when } v(x) = t \models ĝ_2; \\ 0 & \text{Otherwise.} \end{cases}
\]

\( M_s(K) \) can be constructed with \( M_s(K) = M_s^{(1)}(K) + M_s^{(2)}(K) \).

Note: The probability of drawing an exact value \( c \) under the uniform distribution is not zero when \( c = \omega(l, \nu) \) and \( \omega(ĝ_2, \nu) = \kappa(ĝ_2, \nu) \). \( ĝ_2 \) must contain a single point in this case as we assume no time-locks are possible. Since we set the probability to draw a delay from \( ĝ_2 \) to \( H \) it follows that the probability to select the single point in \( ĝ_2 \) is \( H (M_s([c, c]) = H) \).
Its derivative becomes infinite. This applies to the exponential distribution as well.
It holds that: \( \lim \) either satisfies \( \vec{g} \) and since \( P \) we also know that delays that fall within the guard gap cannot be chosen. So:

We can conclude that for each \( d \) we denote with \( d \) that \( \inf_1 \) is uniformly selected from \( [\kappa(l, \nu) - \kappa(l, \nu)] \).

Also if \( z > 1 \) we have drawn more than one delay, which means the first delay fell within the guard gap. Thus:

This means that \( P(\inf_1(\vec{g}_2) \leq X \leq \omega(l, \nu)|z > 1) \) must be the leftover probability

And since \( P(\inf_1(\vec{g}_2) \leq X \leq \omega(l, \nu)) = P(\inf_1(\vec{g}_2) \leq X \leq \omega(l, \nu)|z = 1) + P(\inf_1(\vec{g}_2) \leq X \leq \omega(l, \nu)|z > 1) \) the probability of selecting a delay that falls in the second merged guard is \( P(\inf_1(\vec{g}_2) \leq X \leq \omega(l, \nu)) = 1 - \frac{\sup_1(\vec{g}_1) - \kappa(l, \nu)}{\omega(l, \nu) - \kappa(l, \nu)} \), which equals \( H \).

What remains is to prove that all values \( d \) such that \( d \in \mathbb{R}^+ | \nu + d \geq \vec{g}_2 \) are all evenly likely to be selected.

We denote with \( d_1, d_2, \ldots \) the delays drawn before an edge is taken.

Before a first delay is taken we know that for all values \( \in [\kappa(l, \nu), \omega(l, \nu)] \) it holds that

\[
\lim_{\delta \to 0} \frac{M_\delta([l, l+\delta])}{\delta} = \frac{1}{\omega(l, \nu) - \kappa(l, \nu)}.
\]

If \( d_1 \in \mathcal{G}(l, \nu) \) a new delay \( d_2 \) is uniformly selected from \( [\kappa(l, \nu + d_1), \omega(l, \nu)] \) that either satisfies \( \vec{g}_2 \) or falls within \( \mathcal{G}(l, \nu) \). At this point it holds that \( \lim_{\delta \to 0} \frac{M_\delta([l, l+\delta])}{\delta} = \frac{1}{\omega(l, \nu) - \kappa(l, \nu + d_1)} \).

Also if \( d_n \in \mathcal{G}(l, \nu) \) a new delay \( d_{n+1} \) is uniformly selected from \( [\kappa(l, v*), \omega(l, \nu)] \) that either satisfies \( \vec{g}_2 \) or falls within \( \mathcal{G}(l, \nu) \). At this point it holds that \( \lim_{\delta \to 0} \frac{M_\delta([l, l+\delta])}{\delta} = \frac{1}{\omega(l, \nu) - \kappa(l, v*)} \) with \( v* = v + d_1 + d_2 + \ldots + d_n \).

We know that with probability \( \frac{\omega(l, \nu) - \kappa(l, \nu + v)}{\omega(l, \nu) - \kappa(l, \nu)} \) it holds that \( v + d* \geq \vec{g}_2 \);

with probability \( \frac{\kappa(l, \nu + v) - \kappa(l, \nu + v)}{\omega(l, \nu) - \kappa(l, \nu)} \) it holds that \( d_2 \in \mathcal{G}(l, \nu) \).

It holds that: \( \lim_{\kappa(l, v*) \to \kappa(l, \nu)} \frac{\kappa(l, \nu + v) - \kappa(l, \nu)}{\omega(l, \nu) - \kappa(l, \nu)} \to 0 \), so eventually a delay will be selected that satisfies \( \vec{g}_2 \).

We can conclude that for each \( d_i \in \mathcal{G}(l, \nu) \) all values \( \in [\kappa(\vec{g}_2, \nu), \omega(\vec{g}_2, \nu)] \) are all evenly
likely to be selected when drawing $d_{i+1}$.

**An algorithm for sampling a random delay**

The delay can be determined using the following algorithm, which we have created ourselves:

1. Merge guards that overlap together into a single guard. This results in two merged guards $\vec{g}_1, \vec{g}_2$ and a guard gap $G(l, \nu)$.

2. Calculate the probability $p_1$ that a delay $d$ for a given valuation $\nu$ is drawn such that $\nu + d \models \vec{g}_1$ using $M_s^{(1)}(K)$. Calculate the probability $p_2$ such that $\nu + d \models \vec{g}_2$ using $M_s^{(2)}(K)$.

3. Draw a random value $p$ from $[0,1]$ according to the uniform distribution.

4. If $p \leq p_1$ draw a delay $d$ using the continuous uniform distribution from the interval $[\kappa(\vec{g}_1, \nu), \omega(\vec{g}_1, \nu)]$ or use $p$ to calculate the delay via $d = [\omega(\vec{g}_1, \nu) - \kappa(\vec{g}_1, \nu)] \times p + \kappa(\vec{g}_1, \nu)$.

5. If $p > p_1$ draw a delay $d$ using the continuous uniform distribution from the interval $[\kappa(\vec{g}_2, \nu), \omega(\vec{g}_2, \nu)]$ or use $p$ to calculate the delay via $d = [\omega(\vec{g}_2, \nu) - \kappa(\vec{g}_2, \nu)] \times p + \kappa(\vec{g}_2, \nu)$.

**Not natural**

When an invariant is present Uppaal uses the uniform distribution to draw delays. When guard gaps are present it becomes possible, as we have seen, that the delay selection is no longer uniform. This does not sound logical as it would mean that some clock values are more likely to be chosen than others, something that contradicts the nature of the uniform distribution.

It is possible to alter a model to exclude the situation where no guards are enabled without altering rare event probabilities. This is up to the user of Uppaal. (We go into this later on in this thesis)

**Applications for guard gaps**

It is possible to use guard gaps as time spenders or deadlines. I.E. wait 2 seconds and then jump to the next location. For any other application the user of Uppaal is better of altering the model in such a way that guard gaps are not possible. This is shown in this thesis later on in appendix A. The current version of Uppaal calculates the delay times internally and hidden from the user. When altering the model it becomes easier for the user to understand what truly is happening as it is possible to see what happens by the user.

![Figure 7.2: A timer. The model must wait for two seconds to continue.](image)
7.3 The exponential distribution

In this section we will discuss both the Uppaal exponential distribution and the standard exponential distribution and their formulas.

7.3.1 The regular exponential distribution

The exponential distribution is a continuous probability distribution. It describes the time between events in a Poisson process, a process in which events occur continuously and independently at a constant average rate. An important property of the exponential distribution is that it is memoryless. This means that if a random variable $T$ is exponentially distributed, its conditional probability obeys $Pr(T > s + t | T > s) = Pr(T > t)$ for all $s, t \geq 0$. If a delay has been drawn in the past it will not affect delays drawn in the future in the same location.

The formulas for the exponential distribution with interval $K = [a, b]$ are:

$$M_s(K) = e^{-\lambda \times \max(a - \kappa(l, \nu), 0)} - e^{-\lambda \times \max(b - \kappa(l, \nu), 0)}$$

With:

$$\lim_{\delta \to 0} \frac{M_s([t, t + \delta])}{\delta} = \begin{cases} \lambda e^{-\lambda(t - \kappa(l, \nu))} & \text{for } t \geq \kappa(l, \nu); \\ 0 & \text{for } t < \kappa(l, \nu). \end{cases}$$

Note: $\lambda$ equals the rate of exponential at location $l$.

Applications

There are several different applications for the exponential distribution. In queuing theory, the arrival and the service times of agents in a system (e.g. how long it takes for a bank teller etc. to serve a customer) are often modeled as exponentially distributed variables. In reliability theory failure rates are usually exponentially distributed. In the real world a number of natural phenomena approximately follow the exponential distribution, like radioactive particle decay, customer arrival and service, daily rainfall. Most of the applications listed here can be used in the field of model-checking in some form.

7.3.2 The Uppaal exponential distribution

When locations do not have an invariant present the user of Uppaal is required to specify a rate of exponential $R$, where $R : L \to \mathbb{R}^+$, which we denote $\lambda$. With a rate of exponential attached to a location the chosen delay is distributed according to the exponential distribution. However in Uppaal the same problems as for the uniform distribution exist for the exponential distributions: The probability that a delay gets chosen such that there is no guard true should be zero for the theoretical exponential distribution. Take for example figure 7.3:
Figure 7.3: There is no guard enabled on \((1 \leq x_1 \leq 2)\) and \((3 \leq x_1 \leq 4)\)

There are no guards enabled on the time intervals \((1, 2)\) and \((3, 4)\). What this does is that any probability mass on intervals where no guard is enabled is added to the remainder of the exponential distribution, in the same manner as we have seen earlier for the uniform distribution. Effectively what happens is that the probability to select a delay when no guards are enabled drops to zero and as soon as a guard can be enabled again the exponential distribution continues as normal. This behavior can be seen in figure 7.4:

Figure 7.4: There is no guard enabled on \((1 \leq x_1 \leq 2)\) and \((3 \leq x_1 \leq 4)\)

The reason that the model continues at the same probability value as it ended before a guard gap is because the probability that the delay falls in one of the guard gaps gets added to the remainder of the function distributed according to the exponential distribution. This results in exactly such an increase that the probability height after the guard gap is the same as before the gap.
We can construct the formula for the delay distribution with \( K = [a, b] \) in two parts again in a similar manner as we did for Uppaals uniform distribution. We do this for PTA without clock constraint conjunctions, which means that there are only 2 merged guards \( \vec{g}_1, \vec{g}_2 \)

**Delay distribution** \( M_s^{(1)}(K) \)

For \( \vec{g}_1 \) the formula becomes:

\[
M_s^{(1)}(K) = e^{-\lambda \times \max(\min[a, \omega(\vec{g}_1, \nu)] - \kappa(\vec{g}_1, \nu), 0)} - e^{-\lambda \times \min(\max[b, \kappa(\vec{g}_1, \nu)] - \omega(\vec{g}_1, \nu), 0)}
\]

With the delay density function:

\[
\lim_{\delta \to 0} \frac{M_s^{(1)}([t, t + \delta])}{\delta} = \begin{cases} 
\lambda e^{-\lambda (t - \kappa(\vec{g}_1, \nu))} & \text{when } v(x) = t \models \vec{g}_1; \\
0 & \text{otherwise.}
\end{cases}
\]

These formulas are similar to the exponential distribution without guard gaps. Which is obvious since no guard gaps are present before \( \omega(l, \nu) = \kappa(\vec{g}_1, \nu) \).

**Delay distribution** \( M_s^{(2)}(K) \)

The probability, which we denote \( H \), that a delay gets selected such that \( v + d \models \vec{g}_2 \) is

\[
H = 1 - M_s^{(1)}([\kappa(\vec{g}_1, \nu), \omega(\vec{g}_1, \nu)])
\]

We can multiply this \( H \) using the same reasoning as for the uniform distribution with the formula for the uniform distribution, starting at \( \kappa(\vec{g}_2, \nu) \).

This is achieved by taking the total probability of a delay being drawn from interval \( K \), dividing it by the total probability that a delay is drawn satisfying \( \vec{g}_2 \) and multiplying it with \( H \). For \( \vec{g}_2 \) the formula becomes:

\[
M_s^{(2)}(K) = \frac{(e^{-\lambda \times \max(a, \kappa(\vec{g}_2, \nu))} - e^{-\lambda \times \max(b, \kappa(\vec{g}_2, \nu))})}{e^{-\lambda \times \kappa(\vec{g}_2, \nu)}} \times H
\]

With the delay density function:

\[
\lim_{\delta \to 0} \frac{M_s^{(2)}([t, t + \delta])}{\delta} = \begin{cases} 
\lambda e^{-\lambda (t - \kappa(\vec{g}_2, \nu)) - [\sup\{G(l, \nu)\} - \inf\{G(l, \nu)\}]} & \text{when } v(x) = t \models \vec{g}_2; \\
0 & \text{otherwise.}
\end{cases}
\]

Basically the \([\sup\{G(l, \nu)\} - \inf\{G(l, \nu)\}]\) in the function 'postpones' the delay density function during the guard gap.

Note: When \( \vec{g}_2 \) does not exist we say that both \( M_s^{(2)}(K) \) and its derivative are zero. This is possible as time-locks can only happen when an invariant with an upperbound is present.

**Applications for guard gaps**

The situation where no guards are enabled can be seen as a natural break or a deadline. For example take a barber that goes for a lunch break while clients might still be in the queue. Service is postponed till the lunch break is over.

For any other application the user of Uppaal is better off altering the model in such a way that guard gaps are not possible. This is shown in this thesis later on in section A.
7.4 Clock constraint conjunctions

In Uppaal and standard PTA it is possible to specify conjunctions of clock constraints. Figure 7.5 shows a guard with a clock constraint conjunction \( x > a \land x < b \).

![Figure 7.5: The conjunction of two clock constraints (green area)](image)

Guard conjunctions can cause a wide set of problems for the delay distributions we defined earlier. Since guards with clock constraint conjunctions can now have both an upper and a lower bound it becomes possible that there is more than one guard gap in the delay density distribution. In this section we will only discuss guards with a clock constraint conjunction \( x > a \land x < b \) where \( a < b \) because if \( a > b \) then the guard can never be true and guards containing a clock constraint of the form \( x = a \) can either never be true or can be merged with the other clock constraint to create the clock constraint \( x \geq b \) or \( x \leq b \).

Consider the example shown in figure 7.6.

![Figure 7.6: Guard gaps get added to the remainder of the area](image)

This figure shows the distribution of a PTA which is present in a location with 3 attached guards (shown in yellow) and one invariant \( (x \leq 20) \). Normally the invariant \( (x \leq 20) \) chooses a delay from the interval \([0,20]\), however a delay that falls in a blue gap can not be used to transition over an edge with. There are 2 gaps in our example, on the intervals \([3,4]\) and \([8,12]\). When a delay is chosen that falls in one of those gaps another delay gets chosen starting from that point. The result is that the probability mass that a delay gets chosen that falls within one of the gaps gets added to the remainder of the enabled area. The \([3,4]\) gap gets added to the \((4,20]\) area. The \([8,12]\) gap gets added to the \((12,20]\) area. Do note that the \([8,12]\) gap already received some added probability from the \([3,4]\) gap, so the \( x > 12 \) guard receives probability twice from the \([3,4]\) gap.

![Figure 7.7: The PTA of figure 7.6](image)
7.4.1 An alternative

In the appendix, section A.0.10 we proposed an alternative to clock constraint conjunctions by splitting them up. We have not proven that this works correctly.

7.4.2 Multiple guard gaps in the uniform distribution

In figure 7.6 we presented a PTA that contained several guard gaps. The probability that the guard $g$ is evaluated to true after picking a proper delay depends on both the guards in front of $g$ and on the probability that the delay is initially chosen in such a way that no guard is is evaluated to true.

It is possible to iteratively calculate the probability that a guard is selected. To make things easier we merge all overlapping guards together and obtain an ordered set of merged guards: $\vec{g}_1, \vec{g}_2, \ldots, \vec{g}_n$. We omit the delay distribution $M_s(K)$ due to the fact that the formulas for those functions are rather difficult and simply give the formula that gives the probability that a merged guard is enabled after drawing a delay:

$$\Pr(\vec{g}_j) = \begin{cases} \frac{1 - \sum_{i=1}^{j-1} [Pr(\vec{g}_i)]}{\omega(\nu_2, \nu_1) - \kappa(\vec{g}_j, \nu)} & \text{when } \exists g_j \text{ s.t. } (\omega(g_j, \nu) - \kappa(g_j, \nu)) > 0; \\ 1 - \sum_{i=1}^{j-1} [Pr(\vec{g}_i)] & \text{when } j = n \land \omega(\vec{g}_n, \nu) - \kappa(\vec{g}_n, \nu) = 0; \\ 0 & \text{Otherwise} \end{cases}$$

We will explain this formula using the example from figure 7.6. First we calculate the probability that a guard is selected. To do this, we calculate the probability that a delay is drawn that satisfies the first guard $\vec{g}_1$:

$$\Pr(\vec{g}_1) = 1 - \sum_{i=1}^{0} [Pr(\vec{g}_i)] \times \[3 - 0\] = \frac{3}{20}$$

This value is what we would expect, it is equal to the probability of the guard being selected when no guard gaps are present. The leftover probability to select a delay satisfying one of the other guards is $1 - \frac{3}{20} = \frac{17}{20}$. This probability is spread out over the remainder of the area, namely $[4, 20]$, excluding the first guard gap after the first guard. This is achieved in a similar manner as was done probability $H$ in the previous two sections. This results in:

$$\Pr(\vec{g}_2) = 1 - \sum_{i=1}^{1} [Pr(\vec{g}_i)] \times \[8 - 4\] = \frac{17}{16} \times 4 = \frac{17}{4}$$

The total probability of the first two guards being satisfied after drawing a delay is $\frac{3}{20} + \frac{17}{4}$. The leftover probability is $1 - \left[\frac{3}{20} \times \frac{17}{4}\right]$ and has to be spread out over the remainder of the area $[12, 20]$, not including guard gaps. This results in:

$$\Pr(\vec{g}_3) = 1 - \sum_{i=1}^{2} [Pr(\vec{g}_i)] \times \[20 - 12\] = 1 - \left[\frac{3}{20} \times \frac{17}{4}\right]$$

If the last guard can only be satisfied by drawing an exact value as delay, for example $x = 20$ or $x \geq 20$ when the invariants upper bound is 20 then we would calculate:

$$\Pr(\vec{g}_3) = 1 - \sum_{i=1}^{3} [Pr(\vec{g}_i)] = 1 - \left[\frac{3}{20} \times \frac{17}{4}\right]$$

The delay density function becomes:

$$\lim_{\delta \to 0} \frac{M_s([t, t + \delta])}{\delta} = \begin{cases} \frac{1 - \sum_{i=1}^{j-1} [Pr(\vec{g}_i)]}{\omega(\nu_2, \nu_1) - \kappa(\vec{g}_j, \nu)} & \text{when } \exists g_j \text{ s.t. } (\omega(g_j, \nu) - \kappa(g_j, \nu)) \leq t \leq \omega(g_j, \nu) \\
\land (\omega(g_n, \nu) - \kappa(g_n, \nu)) > 0; \\ \infty & \text{when } \omega(g_n, \nu) - \kappa(g_n, \nu) = 0; \\ 0 & \text{Otherwise} \end{cases}$$

50
Note: The delay density function can become infinite if a fixed probability is assigned to a specific point (normally the invariant’s upper bound).

### 7.4.3 Multiple guard gaps in the exponential distribution

The exponential distribution Uppaal uses to draw delays can also contain multiple guard gaps. This can be seen in the figures 7.3 and 7.4. In the same manner as for the uniform distribution we can come to a function describing the probability that a guard is enabled after drawing a delay:

$$Pr(g_j) = 1 - \sum_{i=1}^{j-1} [Pr(g_i)] e^{-\lambda \times \kappa(g_j, \nu)}$$

The total remaining probability $1 - \sum_{i=1}^{j-1} [Pr(g_i)]$ is divided by the total probability of the remainder of the original exponential distribution $e^{-\lambda \times \kappa(g_j, \nu)}$. These values are possibly unequal due to the guard gaps. This fraction is multiplied with the original (exponentially distributed) probability $[e^{-\lambda \times \omega(g_j, \nu)} - e^{-\lambda \times \kappa(g_j, \nu)}]$ that the guard is enabled after drawing a delay to obtain the new probability $Pr(g_j)$ that the guard is enabled after drawing a delay.

The delay density function is:

$$\lim_{\delta \to 0} \frac{M_s([t, t + \delta])}{\delta} = \begin{cases} \lambda e^{-\lambda(t - \kappa(l, \nu) - R(t))} & \text{when } \exists g_j \text{ s.t. } \kappa(g_j, \nu) \leq t \leq \omega(g_j, \nu); \\ 0 & \text{Otherwise}. \end{cases}$$

$R(t)$ in the formula above is the total range between $\kappa(l, \nu)$ and $t$ that no guard is active. Basically what happens is that the function drop to zero when no guard is active no more and resumes as it did normally when a guard can be active again.

### 7.5 Literature study

To answer the research question we need to see what happens in actual systems modeled with PTA. and whether guard gaps are present or special delay distributions are used. We have found a number of case studies that use PTA to model certain systems/phenomenons.

Kwiatkowska [25, 21, 23, 24] gave a number of PTA examples in her papers. The examples were comparable to:

Figure 7.8: A simple communication protocol. From [21]

This example tries to send a packet. It fails with a 1% probability. If it fails it waits 4 to 8 seconds before trying again. These papers unfortunately do not go into PTA simulation. No assumptions are made about the distributions attached to the locations. However since on the model some invariants are present we would be forced to use the uniform distribution in Uppaal. This can give problems when the actual system that the model represents uses a different distribution. Packet back-off times and service times in queuing theory are often exponentially distributed. A real-life example could be the binary exponential
backoff that is often part of network congestion avoidance as can be seen in [25].

In [24] an example is given for the IEEE 802.11 wireless LAN protocol, a model visually made with Uppaal. The model shows a two-way handshake mechanism of the IEEE 802.11 medium access control scheme, operating in a fixed network topology consisting of two sending stations and two destination stations.

Even though the example does not give any information on distribution times we can obtain some information from it. Backoff times are exponentially distributed and calculated using an function specified in Uppaal’s modeling language. Unfortunately these backoff times are converted to integers and represent the number of timeslots to backoff after a collision. This makes it impossible to alter the model to work with the exponential distribution attached to a location as that does not create an integer value but a real value. Transmission times are chosen nondeterministically, they normally depend on the packet size. Most invariants, including the backoff timer invariant, are used as deadlines: time must be spend before a transition can be made or the channel must sense that it is busy. The uniform distribution can be used to choose a delay time when there is a deadline present. A deadline in this situation is a specific amount of time where the model can not do anything. When the deadline has been passed the model can continue. Uppaal already uses the uniform distribution for these deadlines.

The free and busy signals that are used in the model are generated by outside variables. In [21] a ZeroConf dynamic configuration protocol for IPv4 link-local addresses is modeled. The aim of the protocol is to configure an IP address for a device which newly joins the local network. The IP address is then used to facilitate local communication between the devices of the network.

The model contains three locations that function as deadline locations, so we would be able to use the uniform distribution for those like Uppaal does currently. The model also contains 4 function calls that draw a value randomly from a set using the discrete uniform distribution. Two of these function calls, which pick a delay, were simplified from a continuous uniform choice over [0,2] to a discrete uniform choice over {0,1,2}. These two function calls could be reverted back to a continuous uniform delay selection since they originally intended to function as those. The writer of the paper had to abstract these function calls to a discrete uniform choice as simulation for PTA back in 2006 was not made yet. The other two function calls can be represented as a branching edge as well.

PTA are used to model two case studies in [25]. The first is for a CSMA/CD protocol. The CSMA/CD protocol is designed for networks with a single channel and specifies the behaviour of stations with the aim of minimizing simultaneous use of the channel (data collision). The basic structure of the protocol is as follows: when a station has data to send, it listens to the medium, after which, if the medium was free (no other station is transmitting), the station starts to send its data. On the other hand, if the medium was sensed busy, the station waits a random amount of time, based on the number of failed transmissions of the packet, and then repeats this process.

The model itself uses mainly deadlines to model the backoff timers and transmit times. Backoff times work just as in the IEEE 802.11 wireless LAN protocol. A random integer backoff time gets chosen according to the geometric distribution. The model then uses this backoff time as a deadline.

The second case study is about a FireWire protocol. The IEEE1394 FireWire root contention protocol concerns the election of a leader between two contending nodes of a network. The protocol consists of a number of rounds in which each of the contending
nodes flips a coin; given the result of the coin flip, a node may decide to wait for a short amount of time or a long amount of time. After this amount of time has elapsed, a node then checks to see if the other node has already deferred, and declares itself to be the leader if so; otherwise, this node defers.

Since there is underspecification in the paper and nothing in the official documentation regarding delay times (only lower and upper bounds are provided) we cannot assume that a certain distribution is used. Uppaal uses the uniform distribution, however we cannot say if the distribution is a good approximation of reality.

In [11] an example is shown where 5 trains need to cross a bridge. The bridge only has one set of tracks which can result in that trains might have to wait for each other before they can cross the bridge. This example shows a model with both exponential and uniform distributions attached to the locations in the model. The arrival rate of the trains is modeled using an exponential distribution. The exponential distribution is often used to model arrival rates in queuing theory in general so for this very simple example it is not a bad choice. The bridge crossing, train stopping and starting is represented by 3 locations using uniform distributions. These 3 actions should normally happen within a fixed amount of time. The example itself is a very large simplification of a real-life situation and could be seen as fictional.

Uppaal contains a number of examples that make use of the uniform distribution. None of the models unfortunately make use of the exponential distribution. A CSMA/CD example uses deadlines to make sure a fixed amount of time passes before the model continues to a new location. A FireWire example is also implemented that works the same as the one explained earlier. In this example also lower and upper bounds are given, but without proper specification it is impossible to know what distribution the protocol normally follows (the uniform distribution used now).

7.6 Conclusion

We have shown the differences and relations between the standard distribution and how Uppaal implements them in this chapter.

To properly conclude on our research question we have to consider two cases: One where guard gaps exist and one they do not.

When guard gaps are absent the distributions applied in Uppaal can be modeled exactly how the user intends to. The distributions follow the regular uniform/exponential distributions completely as they are equal when guard gaps are absent. From the various applications and case studies for the exponential distribution we can see that the distribution is often used for service, arrival, failure and decay times (which are all transition times). These transition times can be used as delays for transitions between two states, easily modeled in PTA and Uppaal in the same manner as used in practice.

If the exact distribution is unknown or can not be modeled the user can choose to use the uniform distribution as a simple alternative. Very good accurates cannot be obtained this way, but this does allow the model to be model checked and give some indicative results. Other applications that the uniform distribution is often used for are random number generation or from sampling from other arbitrary distributions. Unfortunately these applications do not involve state transition times so modeling might be troublesome or not needed.

When guard gaps are present in the model the distributions depend on the ranges of
the guard gaps in the model. This can lead to unintended behavior when the user does not know about what happens with these guard gaps. The user might expect that a delay is drawn uniformly/exponentially from all possible choices and not know that guard gaps increase the probability that a delay after a guard gap is drawn.

There are a number of uses for guard gaps. Guard gaps without guard areas in front of them can be used as deadlines as we have seen in practice from the applications and the case studies. "Wait/backoff $c$ seconds before an action should be done." This can easily be modeled in Uppaal. Other uses are in theory for suspensions in service/arrival/failure rates. But those uses could also be achieved by altering/splitting the model and removing guard gaps, which makes the model easier to understand. This is achieved in the upcoming chapters.

Guard gaps can cause problems for the user that simulates them. The way guard gaps behave might sound unlogical to the user. It lead us to wonder about what could be a better, more logical alternative to guard gaps. We discuss some of these in the next chapter.
Chapter 8

Alternative PTA semantics

Research question: What are the current and alternative semantics for models where no edges are enabled at some point in time (guard gaps)?

In this chapter we will discuss a number of alternative semantics of guard gaps. We will start with a small introduction into the current semantics that apply to guard gaps and possible alternatives. After this small introduction we will discuss other options for guard gaps. We will also go into other aspects like alternative timelock semantics and alternative guard semantics.

For the alternative semantics we will construct both the delay distribution function $M_s$ and the output density function $\eta_s$. The delay distribution function $M_s$ will be able to be used for both David’s version of PTA and Baier’s version of TA. The output density function $\eta_s$ can not be used by both versions of PTA. For this reason we will construct two different versions. One for Baier and one for David.

In sections 8.2 - 8.5 we discuss the current and alternative semantics for PTA with guard gaps.
In 8.6 we discuss alternative semantics for overlapping guards.
For these sections we discuss two separate cases: When location $l$ does not have a rate of exponential the exponential distribution is used. And when an invariant is present at $l$ the uniform distribution is used.
In 8.7 we discuss alternative semantics for PTA with timelocks.

Note: Most work in this chapter has been done in preparation for a paper. For this reason some material discussed here might seem out of place.

8.1 Preliminary

During the course of this chapter we will assume that the discussed (probabilistic) timed automata have only one clock. We only briefly discuss PTA with multiple clocks, but will not go into these PTA in detail.

We also define a ‘global’ PTA: $(L, l_0, \Sigma, C, E, R, I, \mathcal{L})$, currently present in state $s = (l, \nu)$. The edges that are enabled in $s$ are $e_1, \ldots, e_n$ where $e_i = (\tau, g_i, \ldots)$.
We also write $G_s = \bigwedge_{i=1} g_i$ and $G^s = \bigwedge_{j=1} g_j$.

Earlier we defined a guard gap $G(l, \nu) = \{d \in \mathbb{R}^+|\nu + d \not= E_i, \nu + d \not= I(l)\}$ the set of points within the interval $[\kappa(l, \nu), \omega(l, \nu)]$ that can not be used to transition to a new
location. However when guards can consist of conjunctions of clock constraints it becomes possible that there are several ‘gaps’, not just one. For this reason we split the existing guard gap into multiple continuous sections \(G^i(l, \nu)\):

**Definition 8.1.1 (Guard gap section):**

Consider an ordered set of in total \(j\) merged guards: \(g_1, \ldots, g_j\). We define a guard gap section as the set of points between two merged guards or between a merged guard and the invariants upper/lower bound that cannot be used to transition to a new location. Three examples of guard gap sections are:

\[
G^1(l, \nu) = \{d \in \mathbb{R}^+ | \nu + d \neq E_l, \nu + d = I(l), d \in [\kappa(l, \nu), \kappa(g_1, \nu)]\},
\]

\[
G^2(l, \nu) = \{d \in \mathbb{R}^+ | \nu + d \neq E_l, \nu + d = I(l), d \in [\omega(g_1, \nu), \kappa(g_2, \nu)]\},
\]

\[
G^3(l, \nu) = \{d \in \mathbb{R}^+ | \nu + d \neq E_l, \nu + d = I(l), d \in [\omega(g_j, \nu), \omega(l, \nu)]\}.
\]

These guard gap sections as seen in the examples do not all necessarily need to exist. If this is the case the index numbers are adjusted accordingly.

If there are \(n\) guard gap sections, then \(G(l, \nu) = G^1(l, \nu) \cup \ldots \cup G^n(l, \nu)\)

For each alternative semantic we propose we give the delay distribution function \(M_s(K)\) and its delay density function \(\lim_{\delta \to 0} \frac{M_s([t, t+\delta])}{\delta}\). The symbol \(K = [a, b]\) denotes a closed interval ranging from \(a\) to \(b\) with \(a, b \in \mathbb{R}^+\).

### 8.2 Current Uppaal semantics

Most of the current semantics of Uppaal are already explained in chapter 4 and 7. We will not go into detail about the semantics again, we will only give a brief recap and provide some additional information.

#### 8.2.1 Introduction

When an invariant is present Uppaal uses the uniform distribution to draw a delay. When Uppaal draws a delay that falls within a guard gap it will draw a new delay according to the specified distribution continuing with the new clock values that include the delay that fell within the guard gap. When an invariant is absent, an exponential distribution is used to simulate PTA. An example:

![Figure 8.1: Guard gaps influence up the pdf](image)

There is no edge with a guard enabled between one and nine. Uppaal will pick a new delay if this happens, after adding the old delay to all clocks. The result is that the lower edge is traversed 90% of the time.

#### 8.2.2 An alternative

Earlier in this thesis we assumed that if a delay is selected that falls within a guard gap, a silent \(\tau\)-transition is performed. An alternative to this could be considered by adding actual selfloops to the model. This way, the delay selection is truly uniform/exponential.

When a delay is selected that falls in one of the guard gaps the model will traverse the
selfloop and select a new delay in the next step. This method is effectively the same as the method currently implemented in Uppaal.

![Figure 8.2: Selfloops can fill up guard gaps](image)

In figure 8.2 we filled up the guard gap that exists in figure 8.1 with a selfloop.

In essence we have achieved the same in our earlier assumption: for all $d \in G(l, \nu)$ we add an edge $e = (l, \tau, g, \emptyset, 1, l)$ to $E$ such that guard $g$ equals the clock constraint $x = \nu + d$.

**Multiple clocks**

The above method also works when multiple clocks are present. By replacing the $[x \geq 9]$ guard with for example the guard $[x_2 \geq 9]$ we would require the selfloop to be the complement of the two existing guards combined. For our example this would be $x_1 > 1 \land x_2 < 9$.

**8.2.3 An example**

Timers are example that make use of guard gaps when using the semantics currently implemented in Uppaal. Wait 2 seconds before the model can jump to a new location.

![Figure 8.3: A timer. The model must wait for two seconds to continue.](image)

Another example could be in the haircut completion time of a barber. He might take a break (guard gap) and postpone service until the break is over.

**8.2.4 Output density function $\eta_s$**

Due to the fact that David discusses PTA and Baier regular TA we need to have two separate output density functions $\eta_s$. For this reason we will denote Uppaal’s/David’s output density function $\eta_s^{(D)}$ and the output probability distribution functions, which denote the probability that a certain edge is taken at a given state, from Baier $\eta_s^{(B)}$ for the remainder of this paper.

We know from David’s definition of PTA that his version of the output density function chooses uniformly among all enabled edges. From this we can construct the formula: $\eta_s^{(D)}(e) = \frac{1}{|H_s|}$, with $H_s$ the set of all enabled edges in state $s$.

Using $\eta_s^{(D)}$ we can construct Baier $\eta_s^{(B)}$. The probability that edge $e = (l, a, g, r, p, l')$ is selected is $\frac{1}{|H_s|}$. The probability that edge $e$ branches to location $l'$ and resetting the clocks in set $r$ is given by $p(r, l')$. Combining the two we can construct $\eta_s^{(B)}$:

$\eta_s^{(B)}(e) = \frac{p(r, l')}{|H_s|} = p(r, l') \times \eta_s^{(D)}$ for edge $e = (l, a, g, r, p, l')$. 
8.3 New semantics 1: Guard gaps

Consider the current Uppaal semantics. When a gap is present it becomes possible that a new delay has to be drawn when the initial delay falls within the guard gap. A different semantics could be considered that excludes the guard gaps from the delay density set.

8.3.1 Uniform distribution

Consider the case that an invariant is present. In this case Uppaal uses the uniform distribution. For the uniform distribution we can alter the delay density function to exclude the guard gaps, in essence drawing a delay uniformly from points in \(\{[\kappa(l,\nu), \omega(l,\nu)]\} \setminus \mathcal{G}(l,\nu)\).

An example where a delay is drawn uniformly from all delays that lead to an enabled edge is shown in figure 8.4:

![Figure 8.4: The model is truly uniform](image)

In order to supply \(M_s(K) = \sum_{\vec{g} \in \vec{G}_s} M_s^{(\vec{g})}(K)\) we need to combine the ranges of all merged guards \(\vec{g} \in \vec{G}_s\), where \(\vec{G}_s\) is the set of all merged guards in state \(s\). Guards are merged to reduce the complexity of the listed functions. We only need to know which delays satisfy some guard, we do not need to know how many guards are satisfied by drawing a specific delay as that does not alter the probability to select that delay.

\[
S = \sum_{\vec{g} \in \vec{G}_s} [\omega(\vec{g},\nu) - \kappa(\vec{g},\nu)],
\]

with \(S\) the combined range of of all merged guards. Using this we can construct \(M_s(K) = \sum_{\vec{g} \in \vec{G}_s} M_s^{(\vec{g})}(K)\) for all guards:

\[
M_s^{(\vec{g})}(K) = \frac{\max[\kappa(\vec{g},\nu), \min(b, \omega(\vec{g},\nu))] - \min[\omega(\vec{g},\nu), \max(a, \kappa(\vec{g},\nu))]}{S}
\]

With:

\[
\lim_{\delta \to 0} \frac{M_s([t, t + \delta])}{\delta} = \begin{cases} \frac{1}{S} & \text{when } \exists \vec{g} \text{ s.t. } v(x) = t \models \vec{g}; \\ 0 & \text{Otherwise}. \end{cases}
\]

8.3.2 Exponential distribution

When an invariant is absent or does not have an upper bound the exponential distribution is used to draw delays.

In order to supply \(M_s(K) = \sum_{\vec{g} \in \vec{G}_s} M_s^{(\vec{g})}(K)\) we need to calculate the probability that a guard is evaluated to true.

\[
S = \sum_{\vec{g} \in \vec{G}_s} [e^{-\lambda \times \omega(\vec{g},\nu)} - e^{-\lambda \times \kappa(\vec{g},\nu)}]
\]

\(S\) gives the probability that a delay is selected that enables a guard. Using \(S\) we can increase the probability that a delay is selected that satisfies a guard to one and construct
\[ M_s(\vec{g})(K) : \]

\[ M_s(\vec{g})(K) = \left[ \frac{1}{S} \right] \times (e^{-\lambda \times \max(a, \omega(\vec{g}, \nu) - \kappa(\vec{g}, \nu), 0)} - e^{-\lambda \times \min(b, \kappa(\vec{g}, \nu) - \omega(\vec{g}, \nu), 0)}) \]

With:

\[ \lim_{\delta \to 0} \frac{M_s(\vec{g})[t, t + \delta]}{\delta} = \begin{cases} \left[ \frac{1}{S} \right] \times \lambda e^{-\lambda (t - \kappa(l, \nu))} & \text{when } v(x) = t \models \vec{g}; \\ 0 & \text{otherwise.} \end{cases} \]

These formulas are similar to the exponential distribution without guard gaps. Which is obvious since no guard gaps are present before \( \omega(l, \nu) = \kappa(\vec{g}, \nu) \).

### 8.3.3 Implementation

The semantics for the uniform distribution discussed in this section can be implemented in Uppaal by altering the model slightly. Using probabilistic branches it is possible to increase/decrease the probability of certain delays to be drawn. Figure 8.5 shows an example of this where the probabilities to transition to a certain location are altered into probabilities on branching edges.

![Figure 8.5: The implementation of our example in Uppaal](image)

The semantics for the exponential distribution in those cases where the clock is at zero when entering the location can be implemented with a selfloop and a reset. Unfortunately if the clock is not at zero when entering the location this implementation using branching edges does not work. A working implementation is:

![Figure 8.6: The implementation of our example with clocks > 0](image)

This implementation adds all the guard ranges together into a single location (in our picture the black location) and draws from the combined range a delay. This does take into consideration clocks larger than zero. If the clock is larger than one the model automatically goes to the location annotated with the \([x \leq 9]\) guard. In the \([x \leq 10]\) location the model chooses a delay uniformly from \([9, 10]\) and can continue if needed.

### Multiple clocks

The presented semantics unfortunately does not work when there are multiple clocks being used on the guards of any attached edges to the first location. By replacing the \([x_1 \geq 9]\) guard with for example \([x_2 \geq 9]\) guard new problems arise. When \(x_2 \geq 9\) any delay drawn can satisfy the guard. When \(x_2 == 0\), only a delay > 9 satisfies the guard. For \(0 < x_2 < 9\) it depends on the value of \(x_2\) (Example: when \(y == 4\) initially, only delays from \([5, 10]\)
satisfy).

A solution could be to just try picking a delay again without adding the previous delay to all clocks.

### 8.4 New Semantics 2: Guard gaps

A second alternative semantic could be considered that forbids the selection of multiple delays in the same location and lets a selected delay that falls within a guard gap deadlock the model.

Our earlier model changes as follows:

![Figure 8.7: This model can deadlock](image)

In the above PTA the model deadlocks if a delay is drawn larger than one and smaller than nine.

These new semantics work exactly the same for the exponential distribution and work also when there are multiple guard gaps present in the PTA.

**An Alternative**

An alternative for these semantics could be created by adding a deadlock location and 'filling' the guard gap with edges to the deadlock location: For all $d \in G(l, \nu)$ we add an edge $e = (l, \tau, g, \emptyset, l', 1')$ to $E$ such that guard $g$ equals the clock constraint $x = \nu + d$. We also add a new location $l'$ to $L$ which represents a deadlock location.

We can create this alternative semantic in Uppaal by adding a deadlock state to 'catch' guard gaps:

![Figure 8.8: This model deadlocks in a new location](image)

The above figure shows the same PTA as in figure 8.7.

**Multiple clocks**

The problem has the same problem and possible solutions as listed in the multiple clocks section under alternative semantics 1.

### 8.5 New Semantics 3: Guard gaps

When a delay falls within a guard gap the clock continues running until that point where there is a guard enabled and then immediately takes the guards edge without redrawing
a new delay.

Figure 8.9: The system waits until a guard becomes enabled before transitioning

If a delay \( d \) is drawn such that \( d \in (1, 9) \) then it waits until 9 before transitioning to location train2.

To make things easier for the reader we will split the delay distribution function \( M_s \) into two parts. Part one is the sum over the probabilities of merged guards \( \vec{g} \). Part two is the sum over the probabilities that a delay is selected that falls within a guard gap. We did this for both the uniform and the exponential distribution. The delay distribution function becomes:

\[
M_s(K) = \sum_{\vec{g} \in \vec{G}_s} M_s^{(\vec{g})}(K) + \sum_{i=1}^j i M_s^{(i)}(K).
\]

For \( M_s^{(\vec{g})}(K) \) we can use the regular uniform/exponential distribution limited to the interval \([\kappa(\vec{g}, \nu), \omega(\vec{g}, \nu)]\) to calculate the probability a delay is selected satisfying \( \vec{g} \).

Originally we used the symbol \( G(l, \nu) \) to represent all the guard gaps. We are able to split this into several guard gaps as we know the ranges of the guards attached to a location. We split \( G(l, \nu) \) and order the newly obtained guard gaps as followed:

\[
G_1(l, \nu), G_2(l, \nu), \ldots, G_j(l, \nu).
\]

The symbol \( M_s^{(i)}(K) \) represents the probability that a delay gets selected that falls within the guard gap \( G(l, \nu) \), added to the first point at which a guard becomes active again \( \sup\{G(l, \nu)\} \), of course only when that point falls within the interval of \( K \).

### 8.5.1 The uniform distribution

The first part is to calculate the probability that a delay is selected from the interval \( K \) for merged guard \( \vec{g} \) according to the uniform distribution limited to the range of merged guard \( \vec{g} \):

\[
M_s^{(\vec{g})}(K) = \frac{\max[\kappa(\vec{g}, \nu), \min(b, \omega(\vec{g}, \nu))] - \min[\omega(\vec{g}, \nu), \max(a, \kappa(\vec{g}, \nu))]}{\omega(l, \nu) - \kappa(l, \nu)}
\]

The second part calculates the probability that a delay is selected that falls within the guard gap \( G(l, \nu) \), added to the first point at which a guard becomes active again \( \sup\{G(l, \nu)\} \), but only if this point falls within the interval of \( K \):

\[
M_s^{(i)}(K) = \begin{cases} \frac{\sup\{G(l, \nu)\} - \inf\{G(l, \nu)\}}{\omega(l, \nu) - \kappa(l, \nu)} & \text{when } a \leq \sup\{G(l, \nu)\} \leq b; \\ 0 & \text{otherwise.} \end{cases}
\]

With the delay density function:

\[
\lim_{\delta \to 0} \frac{M_s([t, t + \delta])}{\delta} = \begin{cases} 1 & \text{when } \exists \text{ merged guard } \vec{g} \text{ s.t. } (v(x) = t \models \vec{g}) t \neq \sup\{G(l, \nu)\}; \\ \infty & \text{when } \exists G(l, \nu) \text{ s.t. } t = \sup\{G(l, \nu)\}; \\ 0 & \text{Otherwise.} \end{cases}
\]

Note: The probability that a delay gets selected that falls within a guard gap gets added to the next value that satisfies that satisfies a guard. As this is a single point the derivative is infinite.

61
8.5.2 The exponential distribution

Using the same reasoning we used for the uniform distribution and applying it to the exponential distribution we can construct the delay distribution function in three parts again:

\[ M_s(\vec{g})(K) = e^{-\lambda \times \min(a, \omega(\vec{g}, \nu))} \]

And:

\[ M_s^{(i)}(K) = \begin{cases} 
(e^{-\lambda \times \inf \{G^i(l, \nu)\}} - e^{-\lambda \times \sup \{G^i(l, \nu)\}}) & \text{when } a \leq \sup \{G^i(l, \nu)\} \leq b; \\
0 & \text{otherwise.}
\end{cases} \]

With the delay density function:

\[
\lim_{\delta \to 0} \frac{M_s([t, t+\delta])}{\delta} = \begin{cases} 
\lambda e^{-\lambda \times (t - \kappa(l, \nu))} & \text{when } \exists \text{ merged guard } \vec{g} \text{ s.t. } v(x) = t \models \vec{g} \land t \neq \sup \{G^i(l, \nu)\}; \\
\infty & \text{when } \exists G^i(l, \nu) \text{ s.t. } t = \sup \{G^i(l, \nu)\}; \\
0 & \text{Otherwise.}
\end{cases}
\]

Note: The probability that a delay gets selected that falls within a guard gap gets added to the next value that satisfies a guard. As this is a single point the derivative is infinite.

A figure which shows what happens for the exponential distribution:

![Figure 8.10: Guard gap delay probabilities are added to the blue arrows](image)

**An example**

Consider the figure below. A train traveler arrives at the station according to the uniform distribution at \([0, 10]\). In the first minute and the last minute he can enter an available train that leaves at the end of the minute. In between there is no train available and the traveler will have to wait until he can board a train. The probability to board the train peaks at time nine as as that point the new train has arrived and is boarded by the traveler.

![Figure 8.11: The system waits until a guard becomes enabled before transitioning](image)
Note: If the guard is \([x > 9]\) then this can give problems as we cannot select the first real number after 9. A possible solution could be to add the probability to the value closest to the guard, in this case 9. Or to exclude guards of the form \(x > c\) for some constant \(c\).

**Implementation**

The semantics in this section can be implemented in Uppaal. This is done by adding an extra location that ‘catches’ the guard gap and delays it to a specified constant. The alternative implementation is shown in figure 8.13 for the example shown in figure 8.11.

![Alternative implementation](image)

Figure 8.12: An alternative implementation

Figure 8.13 shows a PTA that ‘catches’ clock values \(1 < x < 9\) and delays it until 9 seconds have passed before letting the PTA continue.

**Multiple clocks**

The problem has the same problem and possible solutions as listed in the multiple clocks section under alternative semantics 1.

**8.6 New Semantics 4: Overlapping guards**

When two guards overlap there is a probability that a delay is selected that satisfies both guards. For example the guards \([0 \leq x \leq 2]\) and \([1 \leq x \leq 3]\) get both satisfied when a delay gets chosen that falls in the interval \([1 \leq x \leq 2]\). When a delay gets selected that satisfies multiple guards then one of the guards edges gets chosen uniformly to be used to transition to a next location (other distributions are allowed to be used, none others are currently implemented in Uppaal). This reduces the probability that the edge gets selected proportional to how many guards are enabled. Guard gaps are ignored.

A new semantic could be considered where the probability to select an edge does not depend on the number of guards enabled but solely on the range size of the guard and the distribution.

**8.6.1 The uniform distribution**

Consider the following figure:

![Output selection](image)

Figure 8.13: Output selection only depends on the guard ranges
The first delay density function corresponds with the current Uppaal semantics. The second is a delay density function which height depends on the number of guards enabled. When \( n \) guards are enabled the height of the pdf is also \( n \) times higher than normal.

Delay selection can be done by adding up all the guard ranges and increasing the delay density in height in places where multiple guards are enabled as shown in figure 8.13. This can be achieved by splitting the delay distribution function in parts again, one part for each guard:

\[
M_s(K) = \sum_{g \in G_s} M_s^{(g)}(K),
\]

where \( M_s^{(g)}(K) \) denotes the delay distribution function specific to guard \( g \).

In order to supply \( M_s^{(g)}(K) \) we need to calculate the total combined range of all guards, which we denote \( S \):

\[
S = \sum_{g \in G_s} [\omega(g, \nu) - \kappa(g, \nu)]
\]

Using \( S \) we construct:

\[
M_s^{(g)}(K) = \frac{\max[\kappa(g, \nu), \min(b, \omega(g, \nu))] - \min[\omega(g, \nu), \max(a, \kappa(g, \nu))]}{S}
\]

The derivative becomes:

\[
\lim_{\delta \to 0} \frac{M_s([t, t + \delta])}{\delta} = \left\{ \begin{array}{ll}
\frac{|H_s'|}{S} & \text{when } \kappa(l, \nu) \leq v + t \leq \omega(l, \nu); \\
0 & \text{Otherwise.}
\end{array} \right.
\]

With \( s' = (l, \nu + t) \) and \( t \) a delay \( \in \mathbb{R}^+ \).

Note: The above semantics do not work if the model is in a deadlock state since \( S \) would be zero. In this case the new semantics can not be applied and the old semantics should be used. This also applies to the exponential distribution.

### 8.6.2 The exponential distribution

The above approach is also possible for the exponential distribution. The delay areas where \( a \) guards can be enabled have to be increased proportionally just as we explained in the case for the uniform distribution. To supply \( M_s(K) \) we construct anew and different \( S \), which denotes the sum of the probabilities that a delay is selected that satisfies a guard.

\[
S = \sum_{g \in G_s} (e^{-\lambda \times \kappa(g, \nu)} - e^{-\lambda \times \omega(g, \nu)})
\]

Unfortunately this value \( S \) is larger than one if overlapping guards are present, so we need to reduce the total probability that a delay is selected that satisfies a certain guard back to one. This can be achieved by dividing the probability that a guard is selected by \( S \). Using this we construct:

\[
M_s^{(g)}(K) = \frac{e^{-\lambda \times \max(\min(a, \omega(g, \nu)) - \kappa(g, \nu)), 0) - e^{-\lambda \times \min(\max(b, \kappa(g, \nu)) - \omega(g, \nu)), 0)}}{S}
\]

With:

\[
\lim_{\delta \to 0} \frac{M_s([t, t + \delta])}{\delta} = \left\{ \begin{array}{ll}
\frac{\lambda e^{-\lambda(t - \kappa(l, \nu)) \times H_{s'}}}{S} & \text{when } \exists \text{ guard } g \text{ s.t. } (\nu(x) = t \models g) \land t; \\
0 & \text{Otherwise.}
\end{array} \right.
\]

with \( s' = (l, \nu + t) \) the state after increasing all clocks with \( t \).
8.7  Timelock semantics

When guard gaps are present on a location with an invariant a timelock becomes a possibility. A timelock is triggered when time can no longer be spend due to clocks having reached the upperbound of an invariant and the model cannot transition to a new location due to no guards being enabled. Figure 8.14 shows a time lock.

![Figure 8.14: A time lock is present at $x=10$](image)

In figure 8.14 it is possible to draw a delay larger than five. Time must continue to run, this is an assumption. However as soon as the model reaches $[x == 10]$ time can no longer be spend, resulting in a so-called timelock.

In the PTA we have discussed so far in this thesis we assumed that timelocks were forbidden. However we could come up with alternative semantics for timelocks are guard gaps where no guard becomes enabled anymore in the future. We will not go into the details from these new timelock semantics like we did for the alternative semantics we already presented in this chapter since the timelock semantics are comparable to those alternative semantics we presented earlier.

8.7.1  Current timelock semantics

Currently it is assumed that the model is non-zenos. Time must always diverge. It can not get "stuck" at or under a certain constant time value, so it is assumed that timelocks to not happen. If a timelock does happen an error is generated in Uppaal and the simulation is canceled.

8.7.2  New timelock semantics

If we drop the non-zenos assumption it would become possible that a clock time gets stopped. This can result in three new semantics:

1 - The entire model deadlocks when a timelock has been found. This is similar to new semantics 2 we discussed in this chapter.

2 - A specific clock in a single PTA freezes and can no longer be used. Other PTA in the network can continue if needed. This can result in a cascade failure if multiple PTA use the same clock.

3 - Redraw a new delay using the clock values from the point in time when the current location was first entered. This is similar to the new semantics 1 we discussed earlier.
Chapter 9

Rare events & properties

Research question: What are the different kinds of rare events and properties of PTA and which of them are interesting for this project?

In this chapter we will be talking about the different kinds of rare events that can be triggered during simulation of a network of single PTA. We will also be going into the properties (of the form $Pr[\psi]$ and others) that can be specified for PTA. Other properties of PTA can be found in the Simulation models chapter and the Uppaal chapter. We will finish with the properties and rare events that are interesting for this project. The various aspects of this chapter are based on tests from the Uppaal model-checker and for that reason we will mention Uppaal often.

9.1 Rare events

A rare event is an event with a very small probability of occurrence. for the kind of properties that we consider in this thesis, this means: Transitioning via a certain path with a small probability to a location. In PTA there are several kinds of rare events. Each rare event can be triggered in a different way, depending on several different factors. They can be path related, when many non-rare transitions lead to a rare state. They can also be location related, when one single location forces the system to transition to a rare location. A number of different rare events are:

1. Via probabilistic edges/branches. An edge can branch into multiple sub-edges. Each sub-edge has a value attached to it. This value gives the probability of transitioning via the corresponding sub-edge to the next location.
   - If the probability of transitioning to a next location is very low then the location behind the sub-edge becomes a rare location.
   - Via multiple probabilistic branches in a row. Each probabilistic sub-edge might not lead to a rare state. But several in a row might.

   ![Figure 9.1: A rare event via probabilistic edges](image)

2. Via locations. Locations can have a very large invariant or slow rate of exponential. When a location is annotated with an invariant then the simulation delay in that
location is chosen randomly according to the uniform distribution up to the bound of the invariant.

- If the invariant-bound is large and the guard-bound of an outgoing edge is small, then the probability of taking that edge becomes small as well.
- If the rate of exponential is low and the guard-bound of an outgoing edge is small, then the probability of taking that edge becomes small as well.
- When the guard-bound falls within the tail of the exponential distribution the probability of taking the edge from said guard is small.
- Via multiple large invariants/slow rates. The probability of taking an edge with a small guard might not be small enough to make the following location rare. Several in a row might.

![Figure 9.2: A rare event via a small invariant-guard ratio](image)

3. Via edges. If the model contains no guards then the probability that a certain path is taken depends on the number of outgoing edges and/or the distance to the rare location.

- Via a few locations with many outgoing edges. For example when a simulated run goes through two locations, each with 100 outgoing edges, then the probability of taking a path to a certain end state becomes $1/100^2$.
- Via many locations with a each a few outgoing edges. For example when a simulated run goes through ten locations, each with two outgoing edges then the probability of taking a path to a certain end state becomes $1/2^{10}$.
- Via a combination of the former two sub-items. When the distance from the initial location to another location is simply too big a rare location is easily created. For example a location via an edge with only one incoming edge at the far end of the model space.
- Via loops/cycles. A path to a location can have a low probability due to it “eating” up the probability by looping several times before it enters a location. The model might also have to “build” up some time before it can enter a location due to guards.

![Figure 9.3: A rare event via too many edges and locations](image)

4. Via time. It is possible that the model spends too much time in good locations before the possibility arrives to enter a rare fail location. See figure 9.4 below for a better understanding.
- The system can only enter a location when both the guard on the incoming edge and the invariant on the location evaluate to true. When the guard-range or invariant-range is low it might be hard to enter the location, making it a rare location.

- Via cost/time in the property $Pr[\diamond C \leq c \psi]$. The constant $c$ denotes the maximum cost/time before a property $\psi$ must be satisfied. If the constant $c$ is set too low the simulation attempt may rarely reach a location due to spending all the time in preceding states. Setting this time higher could increase the probability of entering the rare location, essentially making the location non-rare.

- Via resets when model-checking the property $Pr[\diamond C \leq c \psi]$. If the clock $C$ never reaches $c$ the property might be rarely satisfied.

- Via stopwatches. A stopwatch can stop a clock while other clocks can still keep running. This can give problems when model-checking the property $Pr[\diamond C \leq c \psi]$. If the clock $C$ never reaches $c$ the property is never satisfied.

- Via a time deadlock (= time lock). If the model spends too much time in one location and causes all attached guards to evaluate to false forever the system deadlocks. The probability of a deadlock depends on the created system model.

- Via normal updates. In normal PTA updates reset a specific clock back to zero which can cause time-related problems. For example an edge resets a clock for a location and in a following location a high guard/invariant-bound is used can cause a location to be rare.

- Via guard conjunctions. Separate single guards might not create a rare transition, guards in conjunction might. For example $(x_1 < 5) \land (x_2 > 6)$.

- Via urgent/committed locations. Urgent locations freeze time, which can cause time-related problems.

- Via different clock rates. Clock rates can differ. Guards like $x_1 - x_2 \geq 0$ are able to be constructed that check whether or not $x_1 \geq x_2$. However when $x_2$ runs much faster than $x_1$ it becomes rare that $x_1 \geq x_2$.

![Figure 9.4: The red location can not be entered when too much time has been spend in the preceding locations](image)

5. Via networks of PTA. When several PTA are being executed at the same time and have work together or share resources this can lead to new rare locations.

- One PTA can cause variable changes to affect other PTA.

- Several PTA can synchronize on one action together. One PTA might have to wait for another PTA to start the synchronization. This can cause time-related problems.

- Via committed locations. When one PTA simulation is in a committed location it forces the PTA to advance from that location. Other PTA can not advance if they are not in a committed location. Having too many committed locations can cause problems.
- Via delay times. For each PTA in the network a delay time gets chosen for the current location of each PTA. The location that chooses the shortest delay time gets to transition to a new location (if possible). For example take a model with 2 PTA. The first PTA has a rate of 10000. The other one a rate of 1. The first PTA almost always chooses a delay smaller than the other PTA. A transition in the second PTA could become a rare event if the system ends as soon as one PTA enters its finish location.

![Figure 9.5: Model A almost always choses a delay shorter than model B](image)

6. Via features related to Uppaal:

- Via abnormal updates. Uppaal allows clocks to be set to any value via updates. Combine this with guards/invariants and a path can become rare. For example the update \( x_2 := 1000 \) could potentially influence all following guards.

- Via abnormal guards. A guard can contain functions, clock conditions, clock differences, all logical operators. A guard example: \((x_1 < 5) \land \text{allTrue(array)} \lor \text{boolValue} \lor \neg(x_1 - x_2 > 6)\), where allTrue is a function that returns true if all values of the array are true. “boolvalue” is a boolean value that can be set during function calls.

- Via Priced Timed Automata. It is possible to specify clock rates in PTA. These clock rates can potentially mess up the model. For example: Clock \( x_1 \) has a rate of 10000. The initial location has the invariant \( x_1 \leq 1 \) and can jump to the final location at all times. In normal PTA the property \( P_{\diamond} [x_1 \leq 1 \text{ final}] \) is always 1, using a clock rate of 10000 the probability to satisfy the property is only 0.01%.

- Via expressions in functions. A part of the model possibly has to be specified using expressions in functions. For example a function could set the time to one million, potentially messing up all guards.

![Figure 9.6: The model time locks when the boolean value isn’t initialized properly or is set to false during a transition.](image)

7. Via a combination of several non-rare situations as listed above. Each of the above individual situations do not necessarily have to create a rare location. However it is possible that a combination of the above situations form a rare location. Multiple of the same situations can also cause a rare event.

9.1.1 Is this list of rare events exhaustive?

During the construction of this thesis we were wondering if the list with all the rare events is exhaustive. Have we listed all types of rare events?
Each model is specified with a number of properties and one or more PTA in a network. Rare events from properties are listed under the time category (4). In a NPTA PTA can influence each other. These NPTA can result in rare events as we have listed under category 5.

For the other categories we have to look at individual PTA. Each PTA can contain locations, edges, branching edges. Each location can have an invariant or a rate of exponential that influences events. We listed these rare events under category 2. Branching edges can contain low probabilities to transition to a specific location making it rare to enter those locations. We listed these rare events under category 1.

Edges can take a wide range of shapes. Each edge can have a guard and a reset. Both of them can be specified in a wide number of shapes unique to Uppaal. Uppaal can also update certain variables when traversing an edge. An edge can only be taken when its guard is true, i.e. when the clocks have the right time values. When an edge is taken it might reset a number of clocks. Both guards and resets are time-related, and because of this we listed them under the time category (4). Updates and guards can be specified in a number of ways unique to Uppaal. Updates can call functions which are built from expressions. Each can trigger a event differently. We listed these situations under category 6.

Edges do not have to have a guard. The number of edges or locations alone could cause certain events to be rare. We listed these rare events under category 3.

All the 6 listed categories contain situations that can cause an event to become rare. Multiple of these situations in a row or combinations of them can also cause events to become rare. This also allows the model to take any size and shape. The rare events these combinations can cause are listed under category 7.

We have gone into all aspects of a model specified in Uppaal. Each aspect has been categorized. We believe that for this reason we have exhausted the number of rare events.

9.1.2 Interesting rare events

When applying importance sampling we are not allowed to alter the model as this could potentially alter the probability to trigger a rare event. The model must stay intact in order to apply IS. We are however allowed to alter the distribution attached to a location or the probability weights of a branching edge. It is also allowed to add extra invariants if they are not smaller than the constant $C$ specified in the value estimation property.

Below we will give reasons why a rare event category could be interesting for this project. As the meaning of the word interesting is rather vague in the context of this project we will translate it to: Not too complex, not too much time-consuming, IS must be able to be automated, it must be able to calculate edge traversal probabilities and the rare event must be important and happen often enough (if a rare event is present).

We aim our focus only on the interesting rare events to make it easier and quicker for us to find ways to implement IS on the rare events.

We do not have to exclude the uninteresting categories from the model, but probabilities to enter the uninteresting rare events might be influenced by the applying IS techniques on the interesting rare events. It is also required that the probability of the uninteresting rare events is never reduced to zero as this would alter the model.
Category 1: Branching edges

Traversing branching edges works similar to traversing edges in DTMC’s. Each edge has a probability attached to it and the probability that the model traverses one of these edges depends on that probability. Importance sampling for DTMC’s already exists in the form of failure biasing, which we researched during earlier projects. Finding a way to apply failure biasing can be relatively easy as it is not a new technique. One of the research questions we formulated also includes this problem. Branching edges are a vital part in PTA. If they would not be present the model would only be timed automata.

Category 2: Guards/invariants

In order to apply importance sampling on this category we would have to force rare events to happen by forcing the system to traverse certain edges. By changing the distributions attached to the location we could force the system to take on a specific clock value and in turn force a certain edge to be taken. The elements of this category, distributions, guards and invariants, are a vital part in PTA. If they would not be present the model would only be a DTMC.

Category 3: Edges/rare paths

Models with lots of edges/locations can be altered into models with branching edges, which we accomplish later on in this thesis. Loops however can not be altered or merged together into a single location. Due to multiple possible resets, special guards or exponential distributions we can not alter or merge them into a single location. Loops do play a vital part in TA and PTA. If they would not be present the model would be in essence just a tree.

Category 4: Time

All the rare events listed in this category would require a different IS approach. This would take too much time to do properly. The property $\Pr[\diamond C \leq c \psi]$ does play a vital role in PTA. It is used in the estimation of the percentage of runs that end in a fail event. Delay biasing techniques could be used in the same manner as in category 2.

Category 5: Networks of PTA

The NPTA models complexity increases exponentially with the number of PTA present in the network. It would take too much time to find a good way to apply IS. We would first have to solve the IS problem on models with only 1 PTA in them and then try to apply those techniques on Networks of PTA. It is possible to combine the several PTA in the network into a single PTA, on which IS could be applied. We talk about merging several PTA together later on in the appendix.

Category 6: Other unique ways

It is almost impossible to apply IS on the rare events in this category. In order apply IS on the rare events in this category it must be possible to alter functions, declarations and abnormal guards. Even if we do not alter the models internal working somehow when altering these items it would take a different approach for each item to apply IS. It would simply take too much time and it would be too complex to apply IS properly.
Category 7: Combinations

A rare event can occur due to a combination of events from the other categories we listed. This does complicate things, however we can not leave combinations out. They are a vital part in PTA.

9.2 Model Properties

During the generation run Uppaal monitors the clock values and current location. If at some point during execution Uppaal can no longer continue to a new location or when a property is made true/false the simulation stops.

There are a wide number of different properties. The properties are based upon temporal logics like CTL and LTL in which location reachability can be expressed. We will not go into regular properties, for more information on them the reader can read [30] or the Uppaal help.

9.2.1 Statistical Properties

UPPAAL can estimate the probability of expression values statistically. There are four types of statistical properties: quantitative, qualitative, comparison and value estimation. We will only go into quantitative and value estimation. The other two can be found in the Uppaal help.

Probability Estimation (Quantitative Model Checking)

\[ \Pr[\text{Clock} \leq c](\{\Diamond \text{ or } \Box\} \psi) \]

Quantitative query estimates the probability of a path expression being true given that the predicate in probability brackets is true. Intuitively the model exploration is bounded by the expression in the brackets: it can be limited by setting the bound on a clock value, model time or the number of steps (discrete transitions). The user does not supply the number of executions to be run, but the confidence that should be reached. The result is an estimated probability and a number of histograms over the values of the variable specified in the probability brackets. Properties of the form \( \Pr[\Box C \leq c \psi] \) can be rewritten into properties of the form \( \Pr[\Diamond C \leq c \phi] \) in the same manner as with regular properties.

Importance sampling could cause problems since it effects the confidence that has to be reached before the property stops.

Value Estimation

\[ E[\text{Clock} \leq c; N](\{\min:\text{ or }\max:\} \psi) \]

Estimates the value of an expression by running a given number of simulations. Depending on the model \( \min(\text{imum}) \) will result in the lowest possible probability that a certain expression is satisfied. In most cases this will be zero if the model can end up in a state that does not satisfy \( \psi \). According to Uppaal tests \( \max(\text{imum}) \) will give a proper estimate. This property works the same as \( \Pr[\Diamond C \leq c \psi] \), but with a user supplied number of executions to be run.

From this point onward we will denote this property \( P[x \leq c](\Diamond \psi) \) and let the user give the number of executions to be run.
9.2.2 State Properties $\psi$ and $\phi$

$\psi$ and $\phi$ are state properties, properties that are true/false in a specific state. State properties are evaluated for the initial state and after each transition. During a transition it is possible that properties are satisfied momentarily, but since there is no evaluation done at that point it is possible to 'miss' a true/false evaluation. The user has to account for this when supplying/creating the model. These state properties can take a wide range of shapes:

**Boolean operators**

Expressions of the form $a \{\leq, <, =, \neq, >, \geq\} b$ can be used to create state properties. $a, b$ are integers. In each location Uppaal will check if the state property is true or not.

**Booleans**

Expressions of the form $red = true$, where $red$ is a boolean variable that can be set when entering a location. Properties like $P[\diamond C \leq c \text{red}]$ can be checked this way.

**Locations**

Expressions of the form $M.loc$, where $M$ is a model and $loc$ is the name of a location, evaluate to true in a state $(l, \nu)$ if and only if $M.loc$ equals the name of $l$.

**Deadlocks**

The state property $\text{deadlock}$ evaluates to true for a state $(L, \nu)$ if and only if for all $d \geq 0$ there is no action successor of $(L, \nu + d)$.

9.2.3 Interesting properties

In order to be able to use properties in this project we must be able to apply importance sampling techniques on the model and then use the property checks to calculate the probabilities of the properties being true.

The regular properties listed in the Uppaal help unfortunately can not be used in this project since they do not estimate the probability that a property evaluates to true. These properties, when model-checked, only give a true or false result, not the probability that one of these is true. Uppaal does not use these properties in simulation. Uppaal checks using different algorithms than simulation to see whether or not the property is true.

Probability estimation estimates the percentage of runs that end in a fail state, but does not give us a useful answer for us. It will only give a confidence interval after running a fixed number of runs to make an accurate guess. The number of runs Uppaal executes in probability estimation depends on the size of the confidence interval and the level of significance.

The statistical property test named value estimation does do what we want. It estimates the probability that a specified property is true as in normal Monte Carlo simulation. We can specify the number of execution runs to be done and Uppaal will calculate, without supplying confidence intervals or other statistical information, the number of times the specified property evaluated to true divided by the total number of executions to be run specified by the user of Uppaal. It does exactly what formula 6.6 does.
9.3 Conclusion: Interesting rare events & properties

Finding a proper solution for applying importance sampling on all rare events in combination with all properties is due to time-constraints impossible. Each rare event category might require a different approach in applying importance sampling which is too time-consuming to deal with for this thesis. Some rare events/properties are rarely used or are not interesting.

Categories 1-3 together with the value estimation property and combinations are interesting for this project.

Categories 1-3 are not the most complex rare events, they happen relatively often as they are a vital part of PTA and would take relatively the least amount of time compared to the other categories. Applying importance sampling on all of these interesting categories is not needed, as some of the rare events can be remodeled into a rare event from another category (which we will try to achieve later on in this thesis). This is also true for the unbounded regular properties, but since they give a true/false answer and not a probability they are not interesting as they are not used in simulation.

The value estimation property is the only property able to be used to estimate the probability that a state property is satisfied and for that reason the only interesting property.
Chapter 10

Failure biasing

Research question: Can we use failure biasing techniques for particular rare events in PTA to create a good change of measure?

The idea behind importance sampling is to change the dynamics of the system in such a way that the system enters a rare state more frequently in order to achieve a variance reduction. Failure biasing is a technique that can be used to achieve this variance reduction.

In this chapter we will look into a number of failure biasing techniques that could potentially be applied on PTA. We will start with an introduction for the user to the techniques behind failure biasing in sections 10.1 and 10.2. Afterwards we will apply those techniques to PTA in section 10.3 and 10.4 in order to create a good change of measure.

Coming back to the research question: We can use failure biasing techniques for certain rare events in PTA to create a good change of measure. In section 10.4 we alter current failure biasing techniques to work for PTA. We also run a number of case studies in this section that show that our form of failure biasing can help reduce the variance of the estimate and increase its accuracy. We present case studies for various situations, both with and without loops and for various changes of measures. In section 10.4 we show some of the results. In section 10.5 we come to a conclusion.

10.1 Introduction

Consider the following problem with failure rate $\epsilon = 10^{-4}$:

![Figure 10.1: A rare event via probabilistic (branching) edges](image)

The chance for the failure transition to occur is very rare ($\epsilon = 10^{-4}$). Because of this rarity, simulation requires several thousands of executions to get an accurate estimate. In failure biasing the total probability of a failure transition (flows towards a failure location) is increased from $\epsilon$ to $p$ and the probability of a repair transition (flows away from a failure location) is changed to $1 - p$. This ensures that the failure transitions are more likely to occur and would in theory result in a variance reduction. Studies have shown that $p$ is best chosen pretty high ($0.5 \leq p \leq 0.9$, see [39, 31, 40]). Running the above example with a value of 50% can give a very accurate result while only a few dozen executions have to
be run which is a significant decrease in the number of simulations to be run. The exact value for $p$ is up to the user.

In this chapter we will discuss a number of old and new importance sampling techniques based upon failure biasing. Using these techniques we might be able to reduce the number of required simulations for an accurate result.

10.2 Known failure biasing techniques

Below we will discuss several failure biasing techniques. Which one should be used depends on the probabilities specified in the model. In [39] Shahabuddin talks about some problems that can show up when the system is not balanced properly. A system is unbalanced when all of the failure rates of the components in the system are not of the same order of magnitude. In those cases it becomes possible that the error in simple failure biasing (SFB) becomes infinitely large, it would be better to use balanced failure biasing (BFB). When the failure rates are in the same order of magnitude then simple failure biasing can be used to achieve importance sampling.

The probabilities for transitioning from a location to another location are given by matrices $B_P$ and $B_Q$. $B_P$ give the original branch probabilities, $B_Q$ the change of measure. Do note that the matrices $B_P$ and $B_Q$ can change when the location or clock values change during an execution. The matrices have to be recalculated after each transition. $B_Q(s, s')$ gives the probability to transition from state $s$ to $s'$.

10.2.1 Simple failure biasing

In [39] the branch matrix $B_Q$ for the embedded DTMC is created as follows using edge set $E$, states $s, s'$ and edge $(s, s') \in E$:

- If $(s, s') \notin E$:
  $$B_Q(s, s') = 0$$

- If $(s, s') \in E$ and $s \in U$ and $s \neq 0$:
  $$B_Q(s, s') = \begin{cases} 
  p \times B_P(s, s')/F_s & \text{If edge } (s, s') \text{ is a fail transition} \\
  (1 - p)B_P(s, s')/R_s & \text{If edge } (s, s') \text{ is a repair transition} \\
  0 & \text{otherwise}
  \end{cases}$$

- If $(s, s') \in E$ and $s \in \Gamma$, or if $(s, s') \in E$ and $s \in U$ and $s = 0$:
  $$B_Q(s, s') = B_P(s, s')$$

$U$ contains all the $up$ states. $\Gamma$ contains all the $fail$ states. The state space $S$ is the combination of $U$ and $\Gamma$ ($S = U \cup \Gamma$). $F_s$ denotes the probability of taking a failure transition at state $s$ under the original distribution. $R_s$ give the probability of taking a repair transition at state $s$. $s = 0$ denotes the initial state.

Unfortunately the simple failure biasing method can only be applied when the failure probabilities are in the same order of magnitude. However in those cases the simple failure biasing technique can give a larger variance reduction then the BFB method.

Note: States in the context of DTMC can be considered equal to locations due to time variables being absent.
10.2.2 Balanced Failure Biasing

For balanced failure biasing the $B_Q$ matrix becomes:

- If $(s,s') \not\in E$:
  
  $$B_Q(s,s') = 0$$

- If $(s,s') \in E$ and $s \in U$ and $s = 0$:
  
  $$B_Q(s,s') = 1/n_{F_s}$$

- If $(s,s') \in E$ and $s \in U$ and $s \neq 0$:
  
  $$B_Q(s,s') = \begin{cases} 
  p/n_{F_s} & \text{If } (s,s') \text{ is a fail transition} \\
  (1-p)B_P(s,s')/R_s & \text{If } (s,s') \text{ is a repair transition} \\
  0 & \text{otherwise}
  \end{cases}$$

- If $(s,s') \in E$ and $s \in \Gamma$:
  
  $$B_Q(s,s') = B_P(s,s')$$

$n_{F_s}$ is the number of failure transitions possible from state $s$. $F_s$ is no longer used in balanced failure biasing as all failure transitions are given equal probability. The new probability of taking a repair transition still depends on the original probability. When a state $s \in \Gamma$ is reached it no longer really matters what probability is assigned to a transition, usually one stops at this point with the simulation. Shahabuddin kept the probability the same.

[39, 31] tell us that Shahabuddin proved that when balanced failure biasing is used that certain performance measure estimates always always have a bounded relative error while simple failure biasing does not when failure rates are of different order of magnitudes. This makes balanced failure biasing a more robust method as it can always be used. Shahabuddin did show that when the system is balanced that the simple failure biasing method does give rise to a bounded relative error.

10.3 PTA failure biasing requirements

In order to create a new failure biasing technique a few problems arise that make it difficult to simply convert one of the known failure biasing techniques to a technique that works for all kinds of PTA. These problems will have to be solved first before a good failure biasing technique could be created for PTA.

We will not consider networks of PTA for this project. NPTA are too time-consuming and more difficult to be able to construct a properly working failure biasing method within the scope of this master thesis project. The same is true for the simple failure biasing method. Since the SFB method does not work in all cases we will not consider it and focus solely on the balanced failure biasing method.

10.3.1 Repair/failure transitions

Failure biasing is a technique that is often applied to reliability models where failure/repair transitions and failure states are present. In other models these transitions and states do not have to be present. We have to find a way to apply failure biasing to these other kinds of models as well.

The idea behind failure biasing is to force the model towards a certain failure state more often to reduce the number of simulations to be run for an accurate estimate. The state we want to reach does not have to be a failure state, it can be any other rare state
we want to apply failure biasing on. Failure transitions flow towards this rare state and repair transitions flow away from this rare state. In other models failure/repair transitions do not have to be present, however they can flow towards/away from a rare state. For this reason we will call transitions (in other models) towards the rare state failure transitions and transitions away from the rare state repair transitions.

Note: In Uppaal branching edges are attached to branchpoints as we can see from the smaller vertex in figure 10.1. Only from these branchpoints failure biasing could be used. From regular locations delay biasing could be used which we will go into in the next chapter.

10.3.2 Loop transitions

Consider figure 10.2 where a loop is shown.

![Figure 10.2: The red state gets entered with a total probability of $2e$](image)

This model has a branching edge with a single failure transition to the red state and a single repair transition to the green state. But we can not classify the selfloop into a repair/failure transition as they can be both. How should we approach these transitions? We can not just ignore them, if the probability that a (self)loop takes place is very high we will not be able to increase the probability of failure transitions by much.

We will have to take these loop transitions into account when performing failure biasing.

10.3.3 Model exploration

In order to apply failure biasing we must be able to know what transitions are repair/failure/loop transitions. This can be achieved by assuming that some form of model exploration has taken place. It would be too time-consuming if we had to assume otherwise and in turn have to create something to discover the types of the transitions.

Unfortunately applying a model exploration algorithm is too time-consuming to be completed during the course of this master thesis project. We will assume that the user that supplied the model also supplies the types of the transitions.

10.3.4 How smart is the user?

Failure biasing can give time-related problems. Consider the following example:
If we model-check $P[\sigma x \leq 10 \text{ red}]$ while using failure biasing on the orange state we will get a variance increase. If the user is smart enough to see that we cannot just apply failure biasing on the orange state there is not a problem.

We will assume that the user has some knowledge of how PTA work and how failure biasing works. Enough to prevent problems as in figure 10.3. If the user did not have the knowledge to prevent failure biasing problems we would have to find a way to automatically prevent failure biasing problems by selecting the right transitions to perform failure biasing on with or without a model exploration.

10.3.5 A good change of measure

Finding the optimal change of measure that minimizes the variance to zero is difficult as it requires knowing the value we are trying to estimate. We have to stick to finding a good variance reduction.

In simple/balanced failure biasing the probability to take a failure transition $p$ is best increased to a value with a range of $[0.5, 0.9]$. This value is explained in [40] and is gotten by testing various probabilities using trial and error. The optimal probability depends on the model and its selection is left up to the user. Unfortunately a range this large leaves a wide range of possible change of measures to choose from. We can theorize over what a good value however.

Consider figure 10.2. This figure shows a single repair transition to the green state with no other ways of getting to the red state. Altering the probability of this transition serves no purpose as reaching the green state would only give a result of zero in simulation. This makes removing the green state from the model (or reducing the probability to zero) and increasing the probabilities of the other 2 transitions a viable option.

If there is a possibility to get to a failure state via a repair transition we cannot reduce the probability to use a repair transition to zero as we would be excluding behavior from the model, which in turn affects the estimate. We would have to reduce the probability of loop and repair transitions (proportionally to the old probability) in order to increase the probability of a failure transition. By experimenting using trial and error we might be able to come up with a good change of measure that works for most models in the same manner as in [40].

10.4 PTA failure biasing

Using the requirements listed in the previous section and the old failure biasing methods we can come up with a failure biasing method for PTA. We present the branch matrix for step failure biasing step $j$:

- If $(e) \notin E$:
  \[
  Q_j^{(B)}(e) = 0
  \]
- If \((e) \in E \text{ and } l \in U \text{ and } l = l_0:\)
  \[ Q_j^{(B)}(e) = \frac{1}{n_{F_s}} \]

- If \((e) \in E \text{ and } l \in U \text{ and } l \neq l_0:\)
  \[ Q_j^{(B)}(e) = \begin{cases} 
  p/n_{F_s} & \text{If } (e, l) \text{ is a fail transition} \\
  q \times P_j^{(B)}(e)/R_s & \text{If } (e, l) \text{ is a repair transition} \\
  r \times P_j^{(B)}(e)/Y(l) & \text{If } (e, l) \text{ is a loop transition} \\
  0 & \text{otherwise} 
  \end{cases} \]

- If \((e, l) \in E \text{ and } l \in \Gamma:\)
  \[ Q_j^{(B)}(e) = P_j^{(B)}(e) \]

- Additionally we require \(p + q + r = 1\) to keep the total probability of leaving a location at one. The symbols \(p, q, r\) represent the new failure, repair and loop transitions under the change of measure;
- We also require the user to select the locations to perform failure biasing on.

\(n_{F_s}\) is the number of failure transitions possible from state \(l\). \(R_s\) is the total probability to take a repair transition and \(Y_s = \bar{Y}_s + \bar{Y}_s\) is the total probability to take a loop transition under the original distributions. We denote \(\bar{Y}_s\) the total probability to take a selfloop (a transition from and to the same location) under the original distribution. We also denote \(\bar{Y}_s\) the total probability to take a loop transition under the original distribution which is not a selfloop. \(l\) is the source location.

10.4.1 The branch likelihood ratio

Using our algorithm we can calculate the branch likelihood (for a single failure biasing step \(j\)):

\[ \frac{P_j^{(B)}(e)}{Q_j^{(B)}(e)} \]

For the branch likelihood to be correct and able to be calculated it is required that the PTA requirements, as listed earlier, are met.

Types of transitions \(p, q, r\)

We assumed earlier in this thesis that guard gaps are filled up using selfloops. This means that it is always possible to do a transition and that the probability to do a transition is one. In PTA failure biasing \(p, q, r\) are the probabilities to take respectively a failure, repair or a loop transition. In order to increase the percentage of runs that end in a failure the value \(p\) is increased. This in turn requires \(p\) and/or \(q\) to be lowered to keep the probability of taking a transition at one.

10.4.2 Case studies

The above method for PrTa failure biasing is an altered version of balanced failure biasing. The altered values for \(p, q, r\) are still left up to the user as is done in simple/balanced failure biasing. By experimenting with these values we can gain a very basic understanding of what a good change of measure should look like.
Case 1: All types of transitions

We simulated the above model with $e = 0.01$ and exact value 0.02 and $n = 1,000,000$ runs. The estimate includes the 95% confidence interval.

<table>
<thead>
<tr>
<th>Method</th>
<th>$p$</th>
<th>$q$</th>
<th>$r$</th>
<th>Estimate</th>
<th>Sample standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC $e = 0.01$</td>
<td>$\frac{1}{2} - 0.01$</td>
<td>$\frac{1}{2}$</td>
<td>$0.01993 \pm 0.00027$</td>
<td>0.140</td>
<td></td>
</tr>
<tr>
<td>IS</td>
<td>0.1</td>
<td>0.44545</td>
<td>0.45455</td>
<td>$0.02000 \pm 0.00008$</td>
<td>0.0427</td>
</tr>
<tr>
<td>IS</td>
<td>0.33</td>
<td>0.33162</td>
<td>0.33838</td>
<td>$0.02004 \pm 0.00006$</td>
<td>0.0274</td>
</tr>
<tr>
<td>IS</td>
<td>$\frac{1}{2}$</td>
<td>0.24747</td>
<td>0.25253</td>
<td>$0.01995 \pm 0.00008$</td>
<td>0.0412</td>
</tr>
<tr>
<td>IS</td>
<td>0.9</td>
<td>0.04949</td>
<td>0.05051</td>
<td>$0.01964 \pm 0.00064$</td>
<td>0.328</td>
</tr>
</tbody>
</table>

We can see that the probability to enter the failure state is highest when we choose a value of 0.33 in this example. Running the example again with $e = 0.001$ results in:

<table>
<thead>
<tr>
<th>Method</th>
<th>$p$</th>
<th>$q$</th>
<th>$r$</th>
<th>Estimate</th>
<th>Sample standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC $e = 0.001$</td>
<td>$\frac{1}{2} - 0.001$</td>
<td>$\frac{1}{2}$</td>
<td>$0.001996 \pm 0.000087$</td>
<td>0.045</td>
<td></td>
</tr>
<tr>
<td>IS</td>
<td>0.1</td>
<td>0.44545</td>
<td>0.45455</td>
<td>$0.001999 \pm 0.000008$</td>
<td>0.0043</td>
</tr>
<tr>
<td>IS</td>
<td>0.33</td>
<td>0.33162</td>
<td>0.33838</td>
<td>$0.001996 \pm 0.000005$</td>
<td>0.00265</td>
</tr>
<tr>
<td>IS</td>
<td>$\frac{1}{2}$</td>
<td>0.24747</td>
<td>0.25253</td>
<td>$0.002001 \pm 0.000011$</td>
<td>0.0057</td>
</tr>
<tr>
<td>IS</td>
<td>0.9</td>
<td>0.04949</td>
<td>0.05051</td>
<td>$0.001958 \pm 0.000063$</td>
<td>0.0322</td>
</tr>
</tbody>
</table>

Again the value of $p = 0.33$ comes out best. The results are comparable to the first simulation, only roughly a factor ten more accurate.

The above results are only for a single model, but do show that for values $p \approx q \approx r \approx \frac{1}{3}$ a good variance reduction can be found.

Case 2: No loop transitions

This model has no loop transitions. Both colored locations are absorbing.

We simulated the above model with $e = 0.0001$ and exact value 0.0001 and $n = 1,000,000$ runs. The estimate includes the 95% confidence interval.

<table>
<thead>
<tr>
<th>Method</th>
<th>$p$</th>
<th>$q$</th>
<th>$r$</th>
<th>Estimate</th>
<th>Sample standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC $e = 0.0001$</td>
<td>0.99999</td>
<td>0</td>
<td>0</td>
<td>$1.00 \cdot 10^{-4} \pm 0.198 \cdot 10^{-4}$</td>
<td>0.01</td>
</tr>
<tr>
<td>IS</td>
<td>0.1</td>
<td>0.9</td>
<td>0</td>
<td>$0.99 \cdot 10^{-4} \pm 0.00588 \cdot 10^{-4}$</td>
<td>0.00030</td>
</tr>
<tr>
<td>IS</td>
<td>0.33</td>
<td>0.67</td>
<td>0</td>
<td>$1.00 \cdot 10^{-4} \pm 0.00277 \cdot 10^{-4}$</td>
<td>1.4 \cdot 10^{-4}</td>
</tr>
<tr>
<td>IS</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
<td>$1.00 \cdot 10^{-4} \pm 0.00196 \cdot 10^{-4}$</td>
<td>1.00 \cdot 10^{-4}</td>
</tr>
<tr>
<td>IS</td>
<td>0.9</td>
<td>0.1</td>
<td>0</td>
<td>$1.00 \cdot 10^{-4} \pm 0.000654 \cdot 10^{-4}$</td>
<td>3.3 \cdot 10^{-5}</td>
</tr>
<tr>
<td>IS</td>
<td>0.99</td>
<td>0.01</td>
<td>0</td>
<td>$1.00 \cdot 10^{-4} \pm 0.00019 \cdot 10^{-4}$</td>
<td>1.0 \cdot 10^{-5}</td>
</tr>
<tr>
<td>IS</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$1.00 \cdot 10^{-4}$</td>
<td>0</td>
</tr>
</tbody>
</table>
The results show that the higher the change of measure the better the result is. This can be expected as there are only two single paths to the failure location. Raising the percentage of failure transitions brings us closer to the optimal change of measure, which is \( p = 1 \). The showed variance for \( p = 1 \) is not zero, but this is because of slight discrepancies in how Java stores real numbers. Since the showed variance is zero we are not able to present a confidence interval.

A \( p \)-value of 0.33 also results in a large variance reduction and could also be used.

It appears that the results confirm that we are able to remove the transition to the green location completely without altering the function of the model. If transitions in the model can not result in a failure anymore we are able to freely remove them when applying importance sampling (if also applying a likelihood ratio). These results also seem to confirm the notion that there exists a bounded error.

**Other cases**

Locations where no repair transitions are possible do not have to be model-checked by using failure biasing. In these cases the model is either in an absorbing state or can only move closer towards a failure state via either a failure transition or a loop transition. In the new location failure biasing could be applied again if needed.

Situations where we are in a failure location do not have to be model-checked as we are already in the location we want to model-check. The same applies to absorbing locations, only now because we can not leave the location any longer.

Locations where only a single transition type is present can not be used in failure biasing as we do not know on which transitions to apply failure biasing on.

**Case 3: Multiple transitions, no loops**

In [21] Kwiatkowska presented a simple communication model. We slightly edited the model and came up with the following:

We check the probability of sending a message that fails three times in a row. Some results are (100 million MC runs, 1 million IS runs):

<table>
<thead>
<tr>
<th>Method</th>
<th>( p )</th>
<th>( q )</th>
<th>Estimate</th>
<th>Sample standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>0.01</td>
<td>0.99</td>
<td>( 0.87 \cdot 10^{-6} \pm 0.18 \cdot 10^{-6} )</td>
<td>( 0.0008 )</td>
</tr>
<tr>
<td>IS</td>
<td>0.5</td>
<td>0.5</td>
<td>( 0.999752 \cdot 10^{-6} \pm 0.0052 \cdot 10^{-6} )</td>
<td>( 2.6 \cdot 10^{-6} )</td>
</tr>
<tr>
<td>IS</td>
<td>0.9</td>
<td>0.1</td>
<td>( 0.999347 \cdot 10^{-6} \pm 0.00120 \cdot 10^{-6} )</td>
<td>( 6.1 \cdot 10^{-7} )</td>
</tr>
<tr>
<td>IS</td>
<td>0.99</td>
<td>0.01</td>
<td>( 0.999908 \cdot 10^{-6} \pm 0.00034 \cdot 10^{-6} )</td>
<td>( 1.75 \cdot 10^{-7} )</td>
</tr>
<tr>
<td>IS</td>
<td>1</td>
<td>0</td>
<td>( 1 \cdot 10^{-6} )</td>
<td>( 0 )</td>
</tr>
</tbody>
</table>

The results show that a very large variance reduction can be obtained. Raising the percentage of failure transitions brings us closer to the optimal change of measure \( p = 1 \).
Case 4: Multiple transitions, loops included

Consider the following models:

The above two models are a representation of a single triple modular redundancy model (TRM). Input goes through 3 separate processors running the same program and then the voter takes a majority vote and decides what should be output (shown in A). Model B is model A turned into a PTA. The up locations indicate the number of processors up. The down location indicates that the voter and in turn the system is down. Values are: $f = 0.001, h = 0.01, g = 0.2, u = 1$. We are model-checking $P[x \leq c](\Diamond \text{DOWN})$, the probability that the system goes down within $c$ seconds. Results for 1 million runs:

<table>
<thead>
<tr>
<th>Method</th>
<th>c-value</th>
<th>Estimate</th>
<th>Sample standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>1000</td>
<td>$0.631988 \pm 0.000945$</td>
<td>0.482</td>
</tr>
<tr>
<td>MC</td>
<td>100</td>
<td>$948.4 \cdot 10^{-4} \pm 5.7 \cdot 10^{-4}$</td>
<td>0.287</td>
</tr>
<tr>
<td>MC</td>
<td>10</td>
<td>$97.8 \cdot 10^{-4} \pm 1.9 \cdot 10^{-4}$</td>
<td>0.0984</td>
</tr>
<tr>
<td>MC</td>
<td>1</td>
<td>$10.46 \cdot 10^{-4} \pm 0.63 \cdot 10^{-4}$</td>
<td>0.0322</td>
</tr>
<tr>
<td>IS</td>
<td>1000</td>
<td>$0.30857 \pm 0.14178$</td>
<td>72.33</td>
</tr>
<tr>
<td>IS</td>
<td>100</td>
<td>$0.09500 \pm 0.00305$</td>
<td>1.56</td>
</tr>
<tr>
<td>IS</td>
<td>10</td>
<td>$99.77 \cdot 10^{-4} \pm 0.65 \cdot 10^{-4}$</td>
<td>0.0333</td>
</tr>
<tr>
<td>IS</td>
<td>1</td>
<td>$10.04 \cdot 10^{-4} \pm 0.16 \cdot 10^{-4}$</td>
<td>0.008</td>
</tr>
</tbody>
</table>

Every edge had a weight of one under the change of measure. This means that an edge has a probability of $\frac{1}{2}$ in UP3 and UP0, a probability of $\frac{1}{3}$ in UP2 and UP1.

The MC results were accurate when compared with the same model in Uppaal. The results show that with a low c-value a good variance reduction and an accurate estimate can be obtained. If the c-value is too high the probability to enter the DOWN location can not be considered rare. In those cases the estimate is possibly wrong as well (which we can see for the $c = 1000$ case).

10.4.3 PTA output failure biasing

Currently after a delay selection if multiple edges are enabled one of the edges get selected according to the discrete uniform distribution. If the selected edge is a branching edge then failure biasing could be used. Consider the following figure:

Figure 10.4: Failure biasing can only be applied 25% of the times
After the delay selection in the initial state there is a 25% probability that the probabilistic branch gets chosen. Failure biasing can only be applied on this branching edge. This makes failure biasing less efficient as it will only be used 25% of the times in this model.

We do know that when multiple edges are enabled an edge gets selected according to the discrete uniform distribution. Knowing this it becomes possible to include the non-branching edges into the failure biasing process. For this we have to move the failure biasing process to just after the delay selection process and before the edge selection process. In essence what happens is shown in figure 10.5

![Figure 10.5: Failure biasing can be applied on all edges](image)

The failure biasing process now includes the edge selection process. The result of this makes it possible to perform failure biasing on more edges at once and thus makes it more efficiently.

### 10.5 Conclusion

Our developed technique based on failure biasing for applying importance sampling does seem to be able to give a variance reduction. We supplied two smaller and 2 larger case studies both with and without selfloops and cycles. Using our results we can determine in which situations we should use specific p, q, r values. Of course these values are not perfect, but should be good enough to grant a variance reduction in most cases. We distinguish several cases, both with and without selfloops (loop where the location stays the same) and with and without repair/failure cycles (loops through multiple locations that lead to a repair/failure location, we assume that selfloops are not part of failure/repair cycles). We supply good p, q, r values (listed as weights) for each location where failure biasing could be used:

<table>
<thead>
<tr>
<th>Selfloops</th>
<th>Fail Cycles</th>
<th>Repair cycles</th>
<th>p</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>98</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Otherwise</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

We have not been able to test very large PTA as we have not been able to find or produce such models. Further research might have to be done to find the right p, q, r values for those very large PTA models. For the smaller models the listed p, q, r could be used to get a nice variance reduction. If one type of transition is missing, the other two probabilities get its probabilities evenly such that the total probability remains one.

Coming back to our research question. We developed a technique based on failure biasing techniques that has the potential of creating a good change of measure. But just like the failure biasing techniques we based our technique on it has some problems. The right values for p, q, r depends largely on the model itself. In order to create a good change
of measure it is best if the user of our technique supplies the right values to use in importance sampling. Even though we supplied good values for $p, q, r$ that can lead to a nice variance reduction for some small and mid-sized models it can become difficult to predict the best $p, q, r$ values for very large models.
Chapter 11

Delay Biasing

Research question: Should our change of measure use different probability densities for the transition times? And if so, what would they look like?

In this chapter we will discuss a technique based on failure biasing to alter the probability densities for the transition times. We will introduce the user to the various probability density functions and its problems in section 11.1. In sections 11.2, 11.3, 11.4 and 11.5 we will apply those techniques to PTA. 11.6 will show some case studies and in 11.7, 11.8 we will conclude on the results and this chapter. In 11.9 and 11.10 we discuss the correctness of our techniques.

Coming back to our research question: In the end we did not truly use different probability densities for the transition times. We kept using the uniform and exponential distribution. Using those distributions we calculated the probabilities that a certain edge toward a failure was selected. We then used failure biasing methods, based on the techniques discussed in the previous chapter, to alter these probabilities where necessary. We also present a number of case studies that show that delay biasing can results in both a variance reduction/accurate estimate and a variance increase/inaccurate estimate depending on whether or not failure biasing is applied correctly or not.

11.1 Introduction

Earlier in chapter 7 we talked about the probability distributions attached to locations. These distributions are often either uniform in nature when an invariant is present or exponential in nature when an invariant is absent. Depending on what guards are attached to the locations the behavior of these distributions can behave weird, for example when guard gaps are present as we have seen in chapter 7. These distributions can cause and influence a number of rare events listed in section 9.1. Consider the following problem:

![Diagram](image)

Figure 11.1: Fail is reached with a probability of 0.01%

When using the uniform distribution to simulate this model it will only end up in the fail location with a probability of $\frac{1}{10000}$. A probability this small could be seen as a rare event.
In importance sampling we increase the number of times that a rare event happens by using a change of measure and afterwards fix the introduced bias by multiplying the result of each simulation run with the likelihood ratio obtained after each simulation run. In order to increase the number of times that the above example results in a failure we would have to increase the number of times the model chooses the upper edge. We would have to 'force' the model to choose a delay such that clock $x < 1$ more often so we can achieve a variance reduction and reduce the number of execution runs that have to be performed to get an accurate result.

In failure biasing we increase the probability that the model chooses a specific edge just like in failure biasing. How much the probability is increased depends on the model and the user. In order to perform failure biasing we must be able to calculate the probability that a specific edge is taken. After calculating these probabilities we increase them in a similar manner to failure biasing so we can force the system towards certain locations more often by selecting an edge according to the increased probabilities. In the end after an edge has been selected a delay is drawn according to the distribution specified on the location limited to the range of the edges guard (so no invalid delays are drawn). We will call this combination of failure biasing and the new delay selection process delay biasing. In essence delay biasing is a form of conditioning: We draw a delay uniformly/exponentially under the condition that a specific edge will be taken.

This chapter

In this chapter we will be talking about how we can apply techniques to force the model to take a specific delay more often. We start with the requirements for creating such a technique and then based on earlier work in failure biasing come up with a new delay biasing technique that we apply on uniform and exponential distributions. We will not go into the specifics of those distributions, as we have done so earlier in sections 7.2.2 and 7.3.1, and only focus on applying delay biasing on these distributions.

11.2 PTA delay biasing requirements

In order to properly construct a good working delay biasing method for PTA we need to overcome a number of problems. In this section we will go into the requirements of the delay biasing method we are trying to construct.

For this project we will not consider networks of PTA. NPTA are too time-consuming and more difficult to be able to construct a properly working delay biasing method for within the scope of this master thesis project. The same applies for model with multiple clocks.

11.2.1 Guard assumptions

When using delay biasing we increase the probability that a certain edge is taken that in turn increases the probability that a certain delay is chosen. This requires picking a delay that falls often within the range of the guard of the transition we want to take.

Earlier we defined the range of the guard $g$ of edge $e$, which equals $G(e)$ we want to take to be $[\kappa(g, \nu), \omega(g, \nu)]$ with $\kappa(g, \nu) \leq \omega(g, \nu)$ and $\kappa(g, \nu), \omega(g, \nu) \in \mathbb{R}_{\geq 0}$. $\kappa(g, \nu)$ is the lower-bound of the guard and $\omega(g, \nu)$ the upper-bound. The range of the invariant, if one is present, will be $[\kappa(l, \nu), \omega(l, \nu)]$ with $\kappa(l, \nu) \leq \omega(l, \nu)$ and $\kappa(l, \nu), \omega(l, \nu) \in \mathbb{R}_{\geq 0}$. $\kappa(l, \nu)$ is usually the current clock value of the invariants location. It can not be bigger than $\omega(l, \nu)$ as that would mean that the invariant does not hold. If no invariant is present $\omega(l, \nu)$ is considered to be infinite.
We assumed that the guard-bounds, both upper and lower, are limited to the current clock value when entering the invariant and the upper bound on the invariant. For example when a guard range is \([10, 30]\), the current clock value 14 and the invariant is \(x \leq 20\), the actual guard range becomes \([14, 20]\).

**Guard gaps**

When guard gaps are present in the model the uniform and exponential distributions are no longer truly uniform/exponential. Delay selection partially depends on the size and location of these guard gaps as we have seen in chapter 7. In section 8.2.2 an alternative for guard gaps was presented. This alternative fills the guard gaps by using self loops to fill up the gaps. By using these selfloops the delay selection also follows the uniform/exponential distribution completely.

We will assume that no guard gaps exist to make it easier to perform delay biasing on specified models. It is easier due to the fact that the distributions are no longer distorted by the guard gaps. Any guard gaps that were originally in the model can be replaced with selfloops by the user.

**Guard and invariant ranges**

The guard and invariant ranges are affected by each other and by the current clock values. For this reason we assume that the guards both lower and upper bounds are cannot be larger than the invariants upper bound or lower than the current clock values. The invariants lower bound will be the same as the current value of the clock used in the invariant.

### 11.2.2 Delay & Edge selection process

Currently in Uppaal a delay is selected according to the uniform/exponential distribution. Afterwards an edge is uniformly selected to transition to a new location. In order to apply a form of failure biasing it is required to revert this process. We first calculate the probabilities that an edge is selected, increase them using importance sampling, select an edge and then select a delay according to the guard range of said edge.

We will make sure that the probabilities that an edge gets selected by the process in reverse will stay the same (when not applying importance sampling).

**The edge selection process**

For the edge selection process it is required that we know what edges are attached to a location, what the guards of said edges, the current clock values are and what the distribution attached to the current location is. Using these values it is possible to calculate the probability that an edge is selected. With this probability it becomes possible to apply a form of importance sampling on the edge selection process and force the model to select an edge more often .

We will assume that all of these values are known at each location. If they would not be known we would not be able to properly calculate the probability that a guard gets satisfied and any attempt at importance sampling would be a waste of time. The probabilities should also be able to be reversed into a distribution if they have been altered for example by applying importance sampling.

We will also assume that we know what the type of all edges are, i.e. repair/failure or loop transitions. The user that supplies the model will have to supply the type of the edges or some form of model exploration has to be done as we talked about in the failure biasing chapter.
Note: Delay biasing can only be performed when the model is currently in a regular location. If the model is in a branchpoint then failure biasing should be used.

The delay selection process

After an edge has been selected a delay will have to be drawn according to the distribution initially attached to the location. This process is a form of conditioning: Draw a delay according to the specified distribution under the condition that a specific edge will always be taken.

11.2.3 Failure biasing approach

The new edge selection process we discussed is similar to the edge selection process for branching edges. For this reason we can use the failure biasing techniques we discussed in chapter 10. It is easier to work with known materials than it is to come up with new approaches to apply delay biasing. We will not discuss the formulas of delay biasing, for that the user can go back to PTA failure biasing in section 10.4.

While failure biasing is a technique that has proved that it can lead to a variance reduction we do not know if the above delay biasing technique is good enough to achieve a variance reduction. We will have to run several case studies to see if it really is a good technique.

11.3 Delay biasing for the uniform distribution

When an invariant is present on a location the uniform distribution is used to draw delays from the range of the invariant. As we assumed that no guard gaps are present it is always possible to draw a delay directly and in turn select an edge. Using the ranges of all guards attached to the location (via an edge) in combination with the attached invariant it is possible to calculate the probability that an edge is selected. Under the change of measure these probabilities are increased.

11.3.1 Calculating the probabilities

The probability that a guard from edge \( e \) gets evaluated to true after drawing a delay is:

\[
\frac{\omega(g,v) - \kappa(g,v)}{\omega(l,v) - \kappa(l,v)}.
\]

Unfortunately the probability that a guard is evaluated to true potentially differs from the probability that a guards edge gets selected for transitioning to a new location. If several guards are overlapping then there is a possibility that multiple edges are enabled after drawing a delay (while only one gets uniformly selected). We also have to take into account that a guard can partially overlap with another edge while another part does not.

It is possible to calculate the probability that a guard gets selected by splitting the guard into several sections. One section for each part that overlaps with another guard and possibly and possibly multiple sections for the parts of the guard that do not overlap. For each guard section the formula \( \frac{S_U - S_L}{\omega(l,v) - \kappa(l,v)} \times \frac{1}{n(S)} \) could be used to calculate the probability that that a delay is drawn that falls in that section and that the edge of the guard is selected for transitioning to a new location. \( S_U \) is the sections upper-bound, \( S_L \) the lower-bound and \( n(S) \) the number of guards enabled in section \( S \). Adding the probabilities of all guard sections together will give the probabilities that said guard is selected.

Consider the following example:
There are two guards. One with the range \([0,2]\), the other with range \([1,3]\) (taking in account the invariant). The guards overlap at \([1,2]\). In order to calculate the probability that guard \([x \leq 2]\) is selected we could split the guard up in sections \([0,1]\) and \([0,2]\) and calculate the probability for those parts separately and then add them together to get the probability that the edge of the guard is used to transition to a new location. For section \([0,1]\) the probability is \(\frac{1-0}{3-0} \times \frac{1}{2} = \frac{1}{3}\) and for section \([1,2]\): \(\frac{2-1}{3-0} \times \frac{1}{2} = \frac{1}{6}\). Together the probability is \(\frac{1}{2}\).

11.3.2 Delay biasing

After the transition probabilities have been calculated they can be altered by using the failure biasing methods we discussed earlier. The \(p,q,r\) under the original distribution are altered in such a way that rare events happen more often. With only these increased probabilities it is not yet possible to draw a delay. To do this we draw a delay uniformly from the attached guards range (after an edge has been selected). Consider the example we used earlier:

It is easy to see that under the original uniform distribution \(F_s = \frac{1}{10000}\), \(R_s = \frac{9999}{10000}\) and \(Y_s = 0\). By using failure biasing on this example we can alter this to \(p = \frac{1}{2}\), \(q = \frac{1}{2}\) and \(r = 0\) and be able to achieve a very good variance reduction which we can see in the case studies section later on in this chapter.

While the delay biasing process shows a slight alteration in the delay selection process, it is still allowed due to the fact that the probabilities of first drawing a delay uniformly and then selecting an enabled edge uniformly to traverse are equal to the probabilities of first selecting an edge to traverse uniformly and then drawing a delay uniformly.

11.4 Delay biasing on the exponential distribution

Locations do not always have an invariant attached to it. In this section we will discuss delay biasing on locations with a rate of exponential is attached to it. With a rate of exponential attached to a location the chosen delay is distributed according to the exponential distribution. This leads to a new set of problems that have to be solved in order to apply delay biasing.

An example:
Due to the lack of an invariant there are no restrictions on the chosen delay. The delay can take on any value and guards do not have to have an upper-bound. The probability that a transition is taken towards a failure state can come from the tail of the exponential distribution as we can see in the example.

For the exponential distribution we will have to calculate the probabilities that a certain edge is taken in a similar manner as for the uniform distribution taking into consideration the problems listed above.

### 11.4.1 Calculating the probabilities

Before applying delay biasing it is possible to use the cumulative probability distribution (cdf) to calculate the probability that a delay is selected between time zero and \( t \). The cdf is given by:

\[
F_P(t) = \begin{cases} 
1 - e^{-\lambda t} & \text{for } t \geq 0; \\
0 & \text{for } t < 0.
\end{cases}
\]

For guards that have a lower-bound \( \geq 0 \) we have to subtract the area under the lower-bound of the guard to get the probability that a delay satisfying the guard.

With \( \omega(g, \nu) \) the guards upper-bound and \( \kappa(g, \nu) \) the guards lower-bound of edge \( e \), assuming both larger than zero. If the guard does not have an upper-bound then infinity will be taken as upper-bound, which gives probability one in the cdf (the entire area under the cdf). The above formulas do not take into consideration overlapping guards. We can alter them in a similar manner as we did for the uniform distribution (without going into the details). By introducing sections and by dividing the probability that the guard is enabled with the number of guards enabled we can calculate the probability that the edge of the guard will be selected for transitioning to the new location. The formula becomes:

\[
(e^{-\lambda \kappa(g, \nu)} - e^{-\lambda \omega(g, \nu)}) \times \frac{1}{n(S)},
\]

for each section \( S \) of the guard to be selected. \( S_U \) is the sections upper-bound, \( S_L \) the sections lower-bound and \( n(S) \) the number of guards overlapping section \( S \). Adding all the sections of the guard together results of course in the probability that said edge is selected.

### 11.4.2 Delay Biasing

After the probabilities of transitioning to a new location via a specific edge have been calculated the values \( p, q, r \) can be altered to apply delay biasing by using the same techniques as listed in the failure biasing chapter. These values can then be used to randomly select an edge to transition to a new location. Afterwards it is required to draw a delay according to the exponential distribution on the range of the selected edges guard by increasing the probability that said delay is chosen from the guards range to 1. This can be achieved by multiplying the pdf and cdf with the factor

\[
\frac{1}{1 - e^{-\lambda \omega(g, \nu)}} - \frac{1}{1 - e^{-\lambda \kappa(g, \nu)}} = e^{-\lambda \kappa(g, \nu)} - e^{-\lambda \omega(g, \nu)}
\]

on the interval \([\kappa(g, \nu), \omega(g, \nu)]\). The pdf on the interval \([\kappa(g, \nu), \omega(g, \nu)]\) becomes:

\[
f_Q(t) = \begin{cases} 
0 & \text{for } t > \omega(g, \nu); \\
\frac{1}{e^{-\lambda \omega(g, \nu)} - e^{-\lambda \kappa(g, \nu)}} \lambda \left[ e^{-\lambda \omega(g, \nu)} - e^{-\lambda t} \right] & \text{for } \kappa(g, \nu) \leq t \leq \omega(g, \nu); \\
0 & \text{for } t < \kappa(g, \nu).
\end{cases}
\]
The cdf becomes:

\[
F_Q(t) = \begin{cases} 
1 & \text{for } t > \omega(g, \nu); \\
\int_0^t \lambda \left[ \frac{1}{e^{-\lambda \omega(g, \nu)} - e^{-\lambda \kappa(g, \nu)}} \right] e^{-\lambda y} dy = e^{-\lambda t} - e^{-\lambda \kappa(g, \nu)} e^{-\lambda \omega(g, \nu)} - e^{-\lambda \kappa(g, \nu)} & \text{for } \kappa(g, \nu) \leq t \leq \omega(g, \nu); \\
0 & \text{for } t < \kappa(g, \nu). 
\end{cases}
\]

The introduced factor in essence removes the probability that a delay is chosen outside of the guards range and adds it to the probability that a delay is chosen inside the guards range. The factor is zero when the delay equals \( \kappa(g, \nu) \) (numerator equals zero) and one when the delay equals \( GU \) (numerator equals denominator). Since the factor is a constant the formulas still behave exponentially.

We assumed that the exponential distribution starts at time zero. If this is not the case the guards will need to be adjusted.

### 11.5 The jump likelihood ratio

To calculate the jump likelihood ratio (for a single delay biasing step) we need to have the probabilities to select edge \( e \) when at state \( s \) under the original distribution and the change of measure. We can calculate these probabilities using the theories we discussed above and we called these probabilities \( P_j^{(J)}(e, s) \) earlier in this thesis. \( Q_j^{(J)}(e, s) \) give the increased probabilities under the change of measure for step \( j \).

The jump likelihood ratio is:

\[
\frac{P_j^{(J)}(e, s)}{Q_j^{(J)}(e, s)}
\]

Do note that in this chapter we skipped the process that reverts the calculated probabilities back to a distribution as it is not required to draw a delay.

For the jump likelihood to be correct and able to be calculated it is required that the PTA requirements, as listed earlier, are met.

### 11.6 Case studies

**Case 1: No loop transitions**

![Diagram](image)

This model has no loop transitions. Both colored locations are absorbing.

We simulated the above model using the uniform distribution in the initial location and for \( n = 1,000,000 \) runs. The estimate includes the 95% confidence interval.
The results are almost completely the same as case 2 in the failure biasing case studies section. This is due to the extreme similarity between the two methods and the fact that the probabilities to transition to the colored locations are the same.

These results also seem to confirm the notion that there exists a bounded error just like in failure biasing. For other reasoning about the results see case 2 in the failure biasing section.

We will not go into the other simple cases as they are similar to the cases we discussed in the failure biasing section.

**Case 2: Guard splitting**

Guard splitting can be used to split up guards into two or more parts. When a guard contains a part that will not satisfy a property or has to pass through a smaller guard (in our example \(x \leq 1\)) guard splitting can result in a greater variance reduction. For more information on guard splitting see section 13.6. The resulting model after guard splitting becomes:

We are model-checking the property \(P[x \leq 1](\text{Bad})\). The results for 1.000.000 runs are:

<table>
<thead>
<tr>
<th>Method</th>
<th>Estimate</th>
<th>Sample standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC1</td>
<td>(38.61 \cdot 10^{-4} \pm 1.22 \cdot 10^{-4})</td>
<td>0.062</td>
</tr>
<tr>
<td>IS1</td>
<td>(37.65 \cdot 10^{-4} \pm 0.89 \cdot 10^{-4})</td>
<td>0.0456</td>
</tr>
<tr>
<td>IS2</td>
<td>(38.15 \cdot 10^{-4} \pm 0.29 \cdot 10^{-4})</td>
<td>0.0146</td>
</tr>
<tr>
<td>IS3</td>
<td>(38.17 \cdot 10^{-4} \pm 0.097 \cdot 10^{-4})</td>
<td>0.005</td>
</tr>
</tbody>
</table>

We used three separate methods (for both Monte Carlo and importance sampling) to achieve a variance reduction:

- MC1: Standard Monte Carlo simulation. \(p, q, r\) are obtained from the model itself.
- IS1: Delay biasing on first 3 locations. Improvement. \(p = q = 1/2, r = 0\) without guard splitting. If a delay > 1 is selected in the first two locations the model will go to the green location from a later location.
- IS2: Delay biasing on first 3 locations. Improvement. \(p = q = 1/2, r = 0\) with guard splitting.
- IS3: Delay biasing on first 3 locations. Improvement. \(p = 9/10, q = 1/10, r = 0\) with guard splitting.

The results show that importance sampling using guard splitting for this example can lead to a greater variance reduction compared to non-guard splitting. Combining guard splitting with a high \(p\)-value leads to an even greater reduction.
Case 3: Loop transitions

Consider the following model:

This model can only go to the red fail state when a delay is selected that satisfies $x \in (90, 91]$. The probability to reach the red state is 0.1. Applying importance sampling on this model for 100,000 runs gives the following results:

For this very simple model importance sampling actually results in a variance increase. The higher we make the probability to transition to the red location under the change of measure (p-value) the higher the variance becomes. This is completely opposite to what we want to achieve.

A possible explanation might be due to the fact that we apply delay biasing on the selfloop. Normally without importance sampling a new delay would be re-choosen and added to the old delay. The result is that it appears that the delay area gets excluded (visually, not internally) from the delay selection process. The user sees that delay gets chosen uniformly from the interval [90,100].

When applying importance sampling on the model a likelihood ratio does get calculated internally for those times that a delay is selected that results in a selfloop. Even when the user eventually sees that a delay gets chosen from the interval [90,100] the likelihood ratio might have seen a significant increase. For example take the probabilities for method IS2. If initially a delay gets chosen that causes a selfloop then this results in a likelihood ratio of $\frac{0.9}{1} = 9$ which can eventually result in a large variance increase.

A possible solution might be to leave these selfloops alone during the importance sampling process and focus solely on the transitions to new locations. The probability of a selfloop does not get altered through the use of importance sampling (it can differ after a selfloop due to new clock values). Results for 100,000 runs are:

We ran the test twice for both Monte Carlo and importance sampling using different ratios between the two non-selfloop transitions. We are unable to supply the $r$-value since the probability of a selfloop alters after a selfloop due to new clock values.

In the MC method we can see from the original model that for 1 failure transition ($p$) 9 repair transitions ($q$) are done.

For the IS1 method we altered the ratio: for 1 failure transition 1 repair transition is done under the change of measure.

The IS2 method has a ratio: for 9 failure transition 1 repair transition is done under the change of measure.

Consider the following model:
This model has a selfloop in the ’middle’. We generated some results in the same manner as the previous test, excluding selfloops:

<table>
<thead>
<tr>
<th>Method</th>
<th>Estimate</th>
<th>Sample standard deviation</th>
<th>Ratio p : q</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>0.010029 ± 0.00063</td>
<td>0.101</td>
<td>1 : 9</td>
</tr>
<tr>
<td>IS1</td>
<td>0.01007 ± 0.00027</td>
<td>0.044</td>
<td>1 : 1</td>
</tr>
<tr>
<td>IS2</td>
<td>0.01003 ± 0.00020</td>
<td>0.032</td>
<td>9 : 1</td>
</tr>
</tbody>
</table>

The results from the last two models do show that excluding selfloops from can lead to a nice variance reduction. If we would take them into account when applying importance sampling it becomes possible to receive a variance increase, something that is unwanted.

**Case 4: Multiple delay biasing in a row**

This is model that represents a train trip through multiple cities. Due to the possibility of trains having a delay in arrival times it is possible for travelers to arrive after having extra delay at their final destination. We model check to achieve the probability to enter location Sne without delay ($P[x <= 132](\diamond Sne)$). There is delay when the model chooses to traverse through a Wait-location. In the simulation we omitted the outgoing edges from the Wait-locations as at that point it is no longer possible to satisfy the property. We ran 100.000 simulations using both Monte Carlo and importance sampling. The results:

<table>
<thead>
<tr>
<th>Method</th>
<th>Estimate</th>
<th>Sample standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>0.02585 ± 0.00099</td>
<td>0.158</td>
</tr>
<tr>
<td>IS1</td>
<td>0.02204 ± 0.01040</td>
<td>1.68</td>
</tr>
<tr>
<td>IS2</td>
<td>0.02648 ± 0.00073</td>
<td>0.118</td>
</tr>
<tr>
<td>IS3</td>
<td>0.02621 ± 0.00063</td>
<td>0.102</td>
</tr>
<tr>
<td>IS4</td>
<td>0.02640 ± 0.00012</td>
<td>0.0191</td>
</tr>
</tbody>
</table>

All transitions towards the Ens location that do not go through a wait location are considered to be failure transitions ($p$). All transitions to a wait locations are repair transitions ($q$), all selfloops are loop transitions ($r$).

MC: Standard Monte Carlo simulation. $p,q,r$ are obtained from the model itself. IS1: Delay biasing on first 4 locations. Wrong results. $p = q = 1/2, r = 0$ on initial location. $p = q = r = 1/3$ on next 3 locations to final location.

IS2: Delay biasing on only the first location. Improvement. $p = 9/10, q = 1/10, r = 0$ on
initial location.
IS3: Delay biasing on first 4 locations. Ratio of $p = 1 : q = 1$ on first 4 locations to final location. Selfloops are ignored.
IS4: Delay biasing on first 4 locations. Ratio of $p = 9 : q = 1$ on first 4 locations to final location. Selfloops are ignored.

By applying importance sampling on selfloops in this model we got some inaccurate results and a large variance increase. When we ignored the selfloops and only applied importance sampling on the transitions to a new locations we were able to get a good variance reduction.

Case 5: A larger model

This is a firewire model. We apply both failure biasing and delay biasing.

<table>
<thead>
<tr>
<th>Method</th>
<th>Estimate</th>
<th>Sample standard deviation</th>
<th>$s, f$</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>0.00273 ± 0.00032</td>
<td>0.052178</td>
<td>$s = 0.9, f = 0.1$</td>
<td>1</td>
</tr>
<tr>
<td>IS</td>
<td>0.002678 ± 9.78 · 10$^{-5}$</td>
<td>0.015783</td>
<td>$s' = 0.5, f' = 0.5$</td>
<td>1</td>
</tr>
<tr>
<td>IS</td>
<td>0.002677 ± 5.25 · 10$^{-5}$</td>
<td>0.008472</td>
<td>$s' = 0.1, f' = 0.9$</td>
<td>1</td>
</tr>
<tr>
<td>MC</td>
<td>2.0 · 10$^{-5}$ ± 2.8 · 10$^{-5}$</td>
<td>0.004472</td>
<td>$s = 0.99, f = 0.01$</td>
<td>2</td>
</tr>
<tr>
<td>IS</td>
<td>2.71 · 10$^{-5}$ ± 9.83 · 10$^{-7}$</td>
<td>1.586 · 10$^{-4}$</td>
<td>$s' = 0.5, f' = 0.5$</td>
<td>2</td>
</tr>
<tr>
<td>IS</td>
<td>2.75 · 10$^{-5}$ ± 4.78 · 10$^{-7}$</td>
<td>7.709 · 10$^{-5}$</td>
<td>$s' = 0.01, f' = 0.99$</td>
<td>2</td>
</tr>
</tbody>
</table>

We estimate the probability to enter the done location before 500 seconds have been spend in total. $s$ gives the probability to start slow. $f$ gives the probability to start fast. Values $s, f$ are altered under importance sampling and represented with $s', f'$. We perform failure biasing on the top 5 locations. We perform delay biasing on the edge from Fast_Fast to Done with probability $\frac{1}{3}$.

For this model we got some nice variance reductions for both tests, which we can see from the standard deviation. The higher we make the probability to start fast the smaller the standard deviation becomes.
11.7 Conclusion

Our developed technique based on failure biasing for applying importance sampling does seem to be able to give a variance reduction. We gave 5 different case studies, each with unique properties, both with and without selfloops and cycles. Using our results we can determine in which situations we should use specific $p, q, r$ values. Of course these values are not perfect, but should be good enough to grant a variance reduction in most cases. We distinguish several cases, both with and without selfloops (loop where the location stays the same) and with and without cycles (loops trough multiple locations) and supply good $p, q, r$ values for each location where failure biasing could be used. We also split $r$ into two parts, namely $\vec{r}$ and $\hat{r}$ such that $r = \vec{r} + \hat{r}$. $\vec{r}$ denotes the probability that the PTA selfloops under the change of measure. $\hat{r}$ denotes the probability under the change of measure that the PTA takes a cycle transition which is not a selfloop. $\vec{Y}_s$ denotes the probability that the PTA selfloops under the original distribution.

<table>
<thead>
<tr>
<th>Selfloops</th>
<th>Cycles</th>
<th>$p$</th>
<th>$q$</th>
<th>$\vec{r}$</th>
<th>$\hat{r}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>$\frac{1-Y_s}{3}$</td>
<td>$\frac{1-Y_s}{3}$</td>
<td>$Y_s^\vec{r}$</td>
<td>$\frac{1-Y_s}{3}$</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>$\frac{1-Y_s}{3}$</td>
<td>$\frac{1-Y_s}{3}$</td>
<td>$Y_s^\vec{r}$</td>
<td>$0$</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>$\frac{1}{3}$</td>
<td>$\frac{1}{3}$</td>
<td>$0$</td>
<td>$\frac{1}{3}$</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>$1$</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

Just like for our failure biasing results we have not been able to find or produce a very large model on which we could apply delay biasing. The best $p, q, r$ values are similar to the ones we found in our failure biasing chapter. We have them in the form of probabilities this time because in the case that selfloops exist the probabilities will have to be left alone for importance sampling to work. The leftover probabilities will have to be distributed according to the attached weights over the other types of transitions.

Coming back to our research question. In the end we did not directly alter the probability distributions for the transition times attached to locations. We actually evaded doing that by just calculating the probabilities to transition to a new location given the current clock values and the edges attached to the current location. After calculating those probabilities we increased them by using a form of failure biasing to gain a variance reduction. It was not needed to revert back to the altered probability distributions for the transition times and thus a waste of time if we did.

11.8 Delay biasing & Failure biasing

In sections 10.5 and 11.7 we have given conclusions and tables when failure biasing and delay biasing can result in a variance reduction for the estimate. In the next section we will take a last look at the correctness of the failure biasing methods we have used and others have used.

11.9 Markovian correctness

Nakayama mentioned in [31] that Shahabuddin in [39] was one of the first to study the asymptotic properties of performance measure estimates obtained through failure biasing techniques. Shahabuddin proved that, when using balanced failure biasing techniques on general (and either balanced or unbalanced) highly reliable Markovian systems where failure rates are either zero or of the form $ce^{\epsilon d}\mathcal{O}(\epsilon^{d})$ with $c > 0, d > 0$ where $\epsilon$ is a rarity parameter, certain performance measure estimates were always bounded by a
relative error. This means that for a fixed number of samples, the expected width of the relative confidence interval for the performance measure estimate remains bounded as the failure rates tend to zero. Highly reliable systems are systems where its components’ failure rates are much smaller than its repair rates.

Shahabuddin also showed that if the failure rates of the system model are of the same order of magnitude that simple failure biasing can also be used to obtain a bounded relative error. If the failure rates are not in the same order of magnitude then it is possible that there is no bounded relative error.

Nakayama showed in [31] more in detail when simple failure biasing methods do and do not give a relative bounded error. However since we based our biasing techniques on the balanced failure biasing method we will not discuss the simple failure biasing method any longer.

11.10 Delay/Failure biasing correctness

From Wikipedia we can find the following: A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present values) depends only upon the present state; that is, given the present, the future does not depend on the past.

Failure biasing was applied and proven correct by Shahabuddin on models with the Markov property, namely DTMCs and CTMCs. PTA and TA however do not have the Markov property so we can not simply say that our form of delay/failure biasing is proven correct. From the case studies in delay biasing we have seen that the estimates were incorrect when selfloops were present. This shows that the relative error is unbounded even when using the balanced failure biasing method.

If we exclude selfloops as we have showed for delay biasing or apply our own form of failure biasing in a correct manner required for IS, we see from our case studies accurate results and the appearance of a relative bounded error. Unfortunately we do were unable to prove the correctness of our techniques during this master project due to time constraints and the difficulty of such a proof.
Chapter 12

The MISS Tool

Research question: Can we construct our own PTA simulator that makes use of importance sampling?

Our original intentions were to add importance sampling to Uppaal. Ultimately we dropped these intentions due to time-constraints and constructed our own simulator which we named MISS: Marco’s Importance Sampling Simulator for PTA.

In this chapter we will discuss the inner working of the MISS tool. We will start with the input, a specially prepared Uppaal PTA saved as an xml document. A few declarations are added to the PTA so the tool can understand where and when to use importance sampling and how long should be simulated. The user of the tool has to select this file in the file chooser, at which it gets parsed and ends up as a workable model the simulator can use. The simulator simulates the parsed model a number of times and produces output in the form of estimates and confidence intervals. The MISS tool is written in Java.

The MISS tool can model check only reachability properties like $P[x \leq c](\Diamond \psi)$, where $\psi$ is a location marked with a red color. The semantics of the models the MISS tool can check are based on the semantics of Uppaal [3] and David [11].

Figure 12.1: Marco’s Importance Sampling Simulator for PTA

Figure 12.1 shows the global workings of the MISS tool. The inner workings of the MISS tool and the meaning of the declarations/model/output are discussed in the next sections in this chapter.
12.1 Input

In order for the MISS tool to simulate a PTA the user is required to create a PTA in Uppaal, which is saved as an xml file, that can be used as input for the MISS tool. Unfortunately Uppaal and its PTA models do not have importance sampling capabilities, so we can not use the standard Uppaal models as input to simulate PTA using importance sampling techniques. We need to modify the Uppaal model slightly to denote where and when importance sampling should be used. Uppaal does allow models to save dummy variables and comments (declarations). Using these dummy variables and comments we can specify in the Uppaal model itself where the MISS tool should use importance sampling techniques during simulation. The dummy variables are also used to store the reachability property the user wishes to simulate. It is vital to the correct working of the tool that these declarations are saved in the model.

A specially prepared model is shown in figure 12.2.

![Diagram of Special Uppaal model](image)

Figure 12.2: A specially prepared Uppaal model able to be used in MISS

A normal PTA specified in Uppaal can be transformed into a model on which the MISS tool can use importance sampling by following a few rules. It is vital that these rules are followed since the MISS tool assumes that the variables in the rules are present in the model. The rules are:

12.1.1 Rule 1: Declarations

In the global declarations window 5 lines must be added which the MISS tool uses to perform importance sampling. These lines, which can be seen in figure 12.1 and 12.2, are:

- "clock x;" – Only a single clock $x$ should be used. No other clocks are allowed in the entire PTA.
- "int IS:=0;" – A dummy variable allowing the $IS$ variable to be used on edges. Used in specify on edges where importance sampling should be used and with which probability.
- "int $f$:=0;" – A dummy variable allowing the $f$ variable to be used on edges. Used to specify on edges where to force the model using delay biasing to use a given probability specified with the $IS$ dummy variable.
- "//n:=10000;" – A comment with a dummy variable that allows the MISS tool to decide the number of simulations to run. The number behind the ’n:=’ can be any positive integer.

- "//max:=100;" – A comment with a dummy variable that allows the MISS tool to decide the maximum time before a simulation ends. The number behind the ’max:=’ can be any positive integer. This number is the same as the upperbound c in the property $P[x \leq c](\diamond \psi)$.

The MISS tool reads the two comments to be able to how long and how often a run needs to be simulated. Uppaal itself does not allow integers larger than 32767. Floating point variables can not be used by Uppaal. Since Java does allow integers larger than 32767 we converted the dummy variables into comments which the MISS tool can read/parse into integers larger than 32767.

12.1.2 Rule 2: Updates/assignments

Branching edges have a probability assigned to them which are used to decide how often a transition is taken. The probability that a regular edge is taken depends on the invariant and the guard. Under failure biasing and delay biasing we alter these probabilities in order to apply importance sampling. To achieve IS we needed a method of specifying the new probability (change of measure) on branching/regular edges. We did this by using the two dummy variables $IS, f$. The MISS tool can read the assignments we make with these dummy variables to see where to apply the change of measure when ordered to use IS. The following assignments, which are placed in the update field of a branching/regular edge in Uppaal, are used:

- "f:=1": Denotes that delay biasing should be used if the model is simulated using importance sampling. Only works when specified on normal edges, not on branching edges.

- "IS:=10": Denotes the change of measure. The 10 denotes the new probability weight and can be any positive value. Can be placed on both branching and normal edges.

- Note: Different assignments should be separated with a comma. i.e.: "x:=0, f:=1, IS:=5" where we denoted that the change of measure probability is 5, that forcing should be used and that clock x should be reset.

12.1.3 Rule 3: Locations

Certain locations must be marked in such a way that the MISS tool knows where to start and stop: These locations are:

- "Init" location – There must be an initial location with the name ”Init”.

- Blue absorbing locations – Absorbing locations should be made blue (not cyan).

- Red failure locations – Failure locations should be made red to allow the tool to model check the property $P[x \leq c](\diamond \text{red})$. 

101
12.2 File chooser

The file chooser is a graphical user interface, containing three buttons and a log window. These elements, which are shown in 12.3, are:

- "Open File + Run" button: Opens a file window where the user can select a file for the MISS tool to use to simulate a model. The model is automatically simulated after loading. Run details are shown in the log window.
- "Info" button: Shows some information in the log window. The information is comparable to the rules we listed earlier.
- "Clear screen" button: Clears the log window.
- Log window: A panel under the three buttons capable of showing information to the user.

In order for the MISS tool to work properly the user is required to select a xml file for the tool which is needed to simulate a PTA with and without importance sampling.

12.3 Parser

In order for the MISS tool to be able to simulate the PTA an xml file needs to be loaded into the tool. The xml file is read and parsed into something the MISS tool comprehends. Uppaal saves it PTA models in the xml file by using special tags and attributes. Some of these are:

- "<declaration>": Both global and local declarations are saved within this tag. For our model this means the declarations from rule 1 are stored.
- "<location>": Contains information about a single location. Has as attributes an id number, an (x,y) location for Uppaal we do not use and a color. Some sub-elements are:
  - "<name>": Contains the name of the location with as attributes the (x,y) location of the name.
  - "<label>": A label attached to the location. Contains either a rate of exponential or an invariant. Has as attributes the (x,y) location of the label and the kind (either "invariant" or "exponentialrate" in this case).
- "<branchpoint>": A special location where outgoing edges are branching edges. Has as attributes an id and a (x,y) location.
- "<transition>": Denotes an edge from and to a location. Has the following sub-elements:
  - "<source>": Denotes the source location id. Stored as an attribute.
- "<target>" Denotes the target location id. Stored as an attribute.
- "<label>" A label that can contain either a probability, a guard or an assignment. Each label has as attributes the (x,y) location of the label and the kind (either "probability", "guard" or "assignment" in this case).

The parser of the MISS tool reads the contents of the tags and attributes and constructs the location set $L$ and the edge set $E$. By looking at the color and names of the locations the parser sees what location is an initial, absorbing or failure location. By looking at the assignments attached to transitions the parser sees what the change of measure is and where delay biasing should be used.

### 12.4 Simulator

After the PTA model from Uppaal has been parsed and the edge set and location set have been created the model can be simulated. We have created an activity diagram of the simulation process, which is shown in figure 12.4. The workings of the activity diagram is explained below.

![Figure 12.4: Simulator activity diagram](image)

After the model has been parsed the models is send to the simulator via a function call. Its parameters are the locations $V$, edges $E$, the number of runs to simulate $n$, the initial location $init$, the maximum time $max$ before the execution ends with a non-failure (which
equals the $c$-value in the property $P[x \leq c](\diamond \psi)$ and the boolean variable $useIS$ that
denotes whether or not importance sampling should be used during simulation.
The symbols $P$ and $Q$ denote the probability distributions used for simulating the PTA. $P$
is the original probability distribution, $Q$ the change of measure.

The PTA model is simulated $n$ times where $n$ was specified as a comment in the decla-
rations area of the PTA. During each simulation a loop is started where the model starts
in the initial location and ends when the model has either entered an absorbing/failure
location or when the time-bound $c$ in the property $P[x \leq c](\diamond \psi)$ has been reached.

During each iteration of the model the next edge $e$ is chosen depending on probability
distribution $P$ if no importance sampling is used and change of measure $Q$ if importance
sampling should be used. What kind of importance sampling is used depends on the
current location: is the current location a branchpoint then failure biasing is used and if
the current location is a regular location delay biasing is used. After an edge has been
selected the likelihood ratio (denoted $L$ in the above figure; location set is denoted $V$) is
updated if importance sampling was used, otherwise it remains one. Afterwards clock $x$ is
increased with the drawn delay and the current location is updated to the location which
edge $e$ has just entered. The next iteration of the loop starts again at this point unless
the model may no longer continue due to having entered a failure/absorbing location or
reaching the time-bound $c$. If the model stops due to reaching the time-bound $c$ or ends
in an absorbing location the likelihood ratio is set to zero.

After the run has finished the the likelihood ratio is saved and reset back to one for
the next run. After $n$ runs have been completed the model calculates the results and ends
the simulation.

**Delay/Failure biasing**

The delay/failure biasing steps showed in the picture follow the techniques discussed in
the delay/failure biasing chapters.

For failure biasing the algorithm in section 10.4 can be used by the user of the tool
to calculate the new probabilities. These new probabilities and the old probabilities are
attached to the branching edges in the model. The likelihood ratio at step/iteration $i$
becomes $\frac{P_{i}^{(B)}(e,l)}{Q_{i}^{(B)}(e,l)}$. $P_{i}^{(B)}(e,l)$ denotes the original probability that edge $e$ branches location
$l$ and $Q_{i}^{(B)}(e,l)$ the change of measure for edge $e$, which is denoted with the expression $IS :=$.

For delay biasing the MISS tool uses the techniques discussed in the delay biasing
chapter to calculating the probabilities under the original distribution to jump from one
location to another. If importance sampling should be used the user of the MISS tool
attaches the new probabilities to the regular edges which the user obtain from using
the failure biasing techniques from section 10.4. The likelihood ratio at step/iteration $i$
becomes $\frac{P_{i}^{(J)}(e,l)}{Q_{i}^{(J)}(e,l)}$. $P_{i}^{(J)}(e,l)$ denotes the original probability, which is calculated by the
MISS tool, and $Q_{i}^{(J)}(e,l)$ the change of measure attached to the regular edge denoted with the
expression $IS :=$.

**12.5 Output**

After the MISS tool has finished simulating the PTA the estimate, confidence interval and
standard deviation are calculated and shown to the user both for simulations with and
without importance sampling.

Some results for the model shown in figure 12.2 are shown in figure 12.5.
Figure 12.5: The output from simulating a PTA $n$ times

The output window shows which file has been opened and parsed and how many runs have been simulated. Two simulations are performed: one Monte Carlo simulation and one Importance Sampling simulation. The estimates follow formulas ???. The 95% confidence intervals and the sample standard deviation are shown. The $1/2$ Confidence difference is the upper bound of the confidence interval minus the lower bound divided by two. This value, and the standard deviation, was used by us to measure the differences between the two simulation techniques.
Chapter 13

Conclusion

In this chapter we conclude on the research questions we tried to answer during the course of this master thesis.

13.1 The Uppaal semantics

At the start of this master project we did not foresee any problems with using the uniform or exponential distribution as is done in Uppaal. However after research we came to the conclusion that PTA were not well-defined. Different people used different semantics for PTA. The creators of Uppaal did not discuss what would have to be done with guard gaps, which are areas in the probability distribution where no delay can be selected, both for the uniform and the exponential distribution.

We looked at the true semantics of both Uppaal and the definition of PTA given by David, one of Uppaal’s creators and came to the conclusion that when they use the uniform or the exponential distribution that the selection of delays is not happening as we normally expect. Delays that fall in guard gaps can not be selected at all, and the probability that a delay gets selected that falls within a guard gap gets added to the remainder of the probability distribution. This results in that values after the guard gap are more likely to be selected than values before the guard gap, something which contradicts the nature of the uniform distribution. This did not seem logical to us, we expected the functions to use the regular uniform distribution.

We researched the delay density functions, which we supplied in mathematical notations, Uppaal indirectly uses to transition to select delays and to transition to new locations. These delay density functions are supplied in such a manner that they can also be used in simulation of the timed automata defined by Baier.

13.2 Alternative semantics

After we came to the conclusion that guard gaps can negatively influence the true probability distribution for selecting delays we started thinking about better solutions. We came up with a number of different semantics for PTA which we presented in the semantics chapter. For most alternative semantics we supplied a number of examples, formulas and implementable alternatives for Uppaal.

The first alternative semantics we supplied we simply excluded guard gaps from the set of possible delays that could potentially be selected to do a delay transition. This way the selection of a delay from all possible delays that could potentially be selected was truly uniform/exponential. These semantics also solve the problem with time-locks, they are simply excluded from the delay density function.
The second alternative semantics simply let models deadlock when a delay was drawn that does not satisfy a guard. Two alternative time-locks semantics were also given in which models can also just deadlock when a time-lock has been triggered.

The third alternative semantics simply waits until a guard becomes enabled before taking a transition. Boarding times for trains and coaches are real-life examples of this alternative.

Two additional semantics, for overlapping guards and timelocks, were also added. The semantics for overlapping guards were altered in such a way that the probability to select an enabled edge does not depend any longer on the number of enabled edges but solely on the size of the enabled guard’s ranges.

The new timelock semantics that were also given are based upon the first three alternative semantics we proposed.

These alternative semantics we supplied seem more logical than what is currently implemented in Uppaal. Some of them can already be implemented in Uppaal while still making use of the current Uppaal semantics. For larger models or when multiple clocks are present the alternatives become more difficult to implement.

The creators of Uppaal, David et al., Baier all allow delay density functions other than the uniform and exponential distribution to be used. Our newly discussed semantics can be used to simulate probabilistic timed automata if needed. Which distribution is best used depends completely on the situation the user wants to model.

13.3 Rare events

As we have seen from the contents of the rare event chapter there are a lot of different kinds of rare events that can be triggered. Each different kind of rare event requires a different way to apply IS. Finding a properly working technique for applying IS on all different kinds of rare events would take up a lot of time and effort. For these reasons we limited ourselves to only the most interesting rare events during the course of this project. With interesting rare events we mean the ones that can be solved, are not too time-consuming for us, not too complex and can be simulated together with our developed tool. The interesting rare events were rare events through branching edges and time-related rare events (small guards/large invariants). We developed and applied techniques called failure biasing and delay biasing to perform IS on the listed interesting rare events. The non-interesting rare events can still be modelchecked if they are altered in such a way that delay biasing or failure biasing can be applied on them as we discussed in this thesis. A number of model alterations for rare events are discussed in the model alteration and improvements section in the appendix.

We also limited ourselves to properties of the form $P[\Diamond_{x\leq c} \text{Fail}]$ where the user supplies the number of executions to be run. Unbounded properties cannot be used in simulation, while the other statistical properties possible are unusable since the user can not supply the number of executions to be run. Rare events due to low $c$ values can be modelchecked by applying a technique called guard splitting that allows failure biasing to be used. This is also discussed in appendix A.

13.4 Failure biasing

Failure biasing is a technique used to increase the likelihood of reaching a certain failure state by increasing the probabilities of the transition toward that state. We have developed a technique based on failure biasing techniques for DTMCs that has the potential of creating a good change of measure. But just like the DTMC failure biasing techniques
we based our technique on it has some problems: it requires that proper new probability values are chosen to be able to achieve a variance reduction. The failure biasing papers we based our technique themselves do not supply good values, they merely list that increasing the probability $p$ of a failure transition to $p \in [0.25, 0.9]$ is most of the times acceptable to achieve a variance reduction. In our case a $p \in [0.25, 0.9]$ is most of the times acceptable as well, but we also supplied a number of better than acceptable probability values, obtained through testing a number of different case studies, in certain situations that the user can use to create a good change of measure.

13.5 Delay biasing

During the start of our master thesis we were discussing methods for applying importance sampling. We came up with a method for altering the probability distributions attached to certain locations in order to increase the likelihood of reaching certain hard-to-reach locations. In the model-checker Uppaal it is standard practice to either use the uniform or the exponential distribution, however in theory any distribution can be used to simulate a model.

In the end we did not directly alter the probability distributions for the transition times attached to locations like we mentioned in our research question. We actually evaded doing that by just calculating the probabilities to transition to a new location given the current clock values and the edges attached to the current location. These probabilities can then be used to determine how often to jump to the next location. It was not needed to revert back to the altered probability distributions for the transition times and thus a waste of time if we did.

In order to apply importance sampling we increased the probabilities that certain transitions, which we calculated using the probability distributions, were taken in a similar manner as we did for failure biasing. We named this technique delay biasing due to lacking a better name.

13.6 Modeling limitations

In one of our research questions we wondered whether or not we had to restrict our models and properties in some way to be able to construct a well-performing change of measure. During the course of this project we already limited ourselves to only the interesting rare events and properties due to time constraints. We also assumed a number of requirements to be able to perform IS in the delay biasing and failure biasing chapters. In order for IS to work it is required that all PTA models follow these requirements.

An additional restriction is that we only allow regular PTA, not the version of PTA implemented in Uppaal. Uppaal allows functions to be called that might influence the probability that a transition is made, something that contradicts one of the requirements of delay biasing and failure biasing. Networks of PTA are also restricted since in networks of PTA, PTA can influence the probabilities that a transition is made of other PTA in the network. The use of regular PTA also ensures that invariants and guards behave exactly as we defined in chapter 3, they can no longer contain multiple clocks.

It might be possible for other rare events and properties to be model-checked using a form of model-checking we have not discussed. Due to time constraints we did not go into these other rare events/properties. We have however supplied a number of alternative model implementations and improvements in the appendix that could be used to improve certain modeling techniques discussed in this master thesis.
13.7 The MISS tool

During the start of this master thesis project we also had a goal to implement our IS techniques in Uppaal. Due to our focus switch towards the semantics of PTA we no longer had the time to implement the IS techniques in Uppaal. We did however create our own simulator named MISS that can simulate standard PTA (not Uppaal's version of PTA) using both Monte Carlo simulation and importance sampling. This simulator is fully working and was used to create all the results from the case studies listed in the delay biasing and failure biasing chapters.
Appendix A

Model alterations/improvements

Some of the rare events and properties we did not discuss can still be model-checked using IS if they are added in an altered form that we discuss further below. By altering the model we can simplify/change a rare event giving us the ability to apply IS on them.

Alterations include networks of PTA, timelocks, deadlocks, properties with time-bounds, urgent/committed locations, guard gaps. A number of ways for the user to improve his/her models are also discussed.

A.0.1 Networks of PTA

Most networks of PTA can be reduced to a single PTA by using product construction. In figure A.1 this process is showed. Model B is the product of models A1 and A2. A problem with product construction is that state-space explosion can occur when the number of locations is large or when the number of PTA is large. The total number of locations can be calculated with: \( \prod_{i=1}^{n} (|L_i|) \), where \( n \) is the number or PTA in the network. \( |L_i| \) is the total number of locations model \( i \) has. We can see that the number of locations in the product of several models grows exponentially with the number of PTA.

\[ \prod_{i=1}^{n} (|L_i|) \]

Note: When PTA are merged invariants might contain multiple clocks. In this case the invariants no longer follow the definition of PTA.

A.0.2 Time locks

Models with time lock errors can be replaced by models without time lock errors without having to change the underlying model. This requires adding a new location, called
“timelock”, and adding a new edge from each location, that can time lock in the original model, to the new “timelock” location. The new edge must have a guard with the same range as that the original locations time lock time. The new timelock state must have an exponential rate attached to it to prevent the state itself from timelocking. See figure A.2 for a better understanding of the situation.

![Diagram](image)

**Figure A.2**: Adding a timelock state can prevent the model from giving timelock errors.

Whether the guard on any edge to the timelock location should be exactly the maximum value of the preceding invariant or the complete range of the time that the preceding location can no longer transition is up to the user. For example in figure A.2 the $x_1 = 10$ guard on the timelock transition could be replaced with a $x_1 > 5$ guard.

A bonus for adding timelock locations is that they can be modelchecked. With properties like $P[\diamond x_1 < c \text{ Timelock}]$ we can check the percentage of times the system ends up in a timelock before $x_1$ has reached the value $c$.

**A.0.3 Dead locks**

Just like timelock locations deadlock locations can be added as well. An extra “Deadlock” location has to be added together with edges from locations that could potentially deadlock in the original model. This has to be done in the same way as this is done for time locks, only this time for locations with a rate of exponential. New edges from locations with a rate of exponential need to have a guard with the same lower bound as the maximum upper bound of all the attached edges combined. Final locations that do no have an invariant or rate of exponential have to be turned into urgent locations and be given an edge to the deadlock state. See figure A.3 for a better understanding of the situation.

![Diagram](image)

**Figure A.3**: Adding a deadlock location can help modelchecking deadlocks.

A bonus for adding deadlock locations is that deadlocks can be modelchecked just like we have seen with timelocks. Properties like $P[\diamond x_1 < c \text{ Deadlock}]$ can be checked.

Uppaal already has a test build in for checking if a deadlock has happened.

**A.0.4 Urgent locations**

Urgent locations can be replaced with a new regular location. A new clock $x_2$ must also be added and may only be used in the new location. The new location must have the invariant $x_2 \leq 0$ to freeze time and any incoming edges must reset clock $x_2$ to zero. All of this together is needed for the new regular location to ensure that it works exactly like an urgent location. This is shown in figure A.9 where the urgent location in model A is replaced, resulting in model B.
Figure A.4: Both models are equal

A.0.5 Committed locations

Committed locations work just like urgent locations when only one PTA is present. For networks of PTA it is possible that the system is in an committed location for one PTA and in a normal location for another PTA. A new clock \( x_3 \) in combination with guards and a regular location, next to the replacements for urgent locations, could be used to replace committed locations. Any incoming edges into the regular location would have to reset clock \( x_3 \). All edges not attached to the regular location will need the guard \( x_3 > 0 \). This process can be see in figure A.5.

Figure A.5: Both models are equal

The above process only works when the next location after the committed/new regular location does not freeze time. If the next location does freeze time then more clocks, guards and resetting edges are required to make sure the model remains equal to the model with committed locations. Replacing multiple committed locations is best done iteratively, replacing one committed location each iteration. Replacing a model with many committed locations can be quite cumbersome. It might be better to restrict the model to not contain any urgent/committed locations at all.

A.0.6 Branching edges

For locations with guard-lacking-edges the probability of taking an edge is the same for all edges. All the edges can be combined into a single probabilistic branching edge. The probabilities of jumping to a next state remain the same. Modeling with a branching edge can help us when applying IS because then we can increase the probability that a certain edge is taken. This is shown in figure A.6.

Figure A.6: Multiple edges can give problems when using delay biasing

Edges that might have guards can also partially be replaced with probabilistic branching edges. When multiple edges can be enabled at the same time it is possible to create a branching edge on the interval that those edges are enabled together. An example is shown in figure A.7. On model A only delay biasing can be applied, which still only gives a probability of \( 1/3 \) to enter the red location. On model B we can use both failure biasing and delay biasing. This way we can increase the probability to enter the red location to any value \( < 1 \) when using importance sampling.
When the guards are not equal, but do overlap somewhat it is also possible to change the model. This is shown in figure A.8 where model A can be transformed into model B.

The above process can also be reversed, but only if the probabilities of the branching edge are rational. This is shown in figure A.9.

**A.0.7 Merging locations**

For path-related problems it might be useful to merge several locations together while keeping the number of different paths the same. This can be done by merging several locations together and transforming all paths consisting of multiple edges into a single edge. This is shown in figure A.10 for a model containing a loop. Time properties are left out in this model. The user does have to account for the fact that these time properties can change as well when merging several locations together.

Merging non-important locations. If a system is in a location that can not lead to a specific rare location anymore then we do not have to execute the model any longer. It might be useful to merge all these locations together into a single location and stop the execution of a run as soon as the location is entered. Merging reduces the number of states, which in turn can help reduce simulation times.

**Low properties**

When model-checking properties with a small range like $P[\diamond x<1 \text{ Bad}]$ all locations and edges that make use of clock-values larger than one can be removed from the model. This can lead to a more efficient model.
A.0.8 Other distributions

Currently in Uppaal delay selection can only be performed according to the uniform or exponential distribution. However each model can be altered to function with other pdfs, that are only partially based upon the uniform or exponential distribution. These new distributions give us more ways to model a system. Below we have constructed 2 models. Each with a different probability distribution.

Model A follows:

\[
\begin{align*}
\text{Pick } x_1 &= 5 \text{ with probability } 0.25; \\
\text{Pick } x_1 &= 10 \text{ with probability } 0.25; \\
\text{Pick uniformly from } [x_1 < 10 \land x_1 \neq 5] \text{ with probability } 0.50
\end{align*}
\]

Model B follows:

\[
\begin{align*}
\text{Pick uniformly from } [0, 10] \text{ with probability } 0.33; \\
\text{Pick uniformly from } [10, 20] \text{ with probability } 0.67
\end{align*}
\]

Figure A.11: 2 models that do not follow the uniform distribution directly.

The user does have to be careful with the clock value for \(x_1\) when entering the first location.

A.0.9 The uniform/exponential distribution

The situation where no guards are enabled for the uniform distribution can be altered to a model where it is not possible to have inactive guards. This is shown in figure A.12. Model A can be transformed into model B by the user.

Figure A.12: A non-uniform model can be reversed to a uniform model.

During the transformation the user creates a selfloop on every location where guard gaps are possible. These selfloops must have the same guard range as the guard gaps. In essence we are filling the guard gaps up with the selfloop so the delay selection is really uniform. The model behavior remains the same.

This improvement for the uniform distribution also works for the exponential distribution.

A.0.10 Guard splitting

Sometimes it is possible that delay biasing is not enough to ensure that a good delay is chosen without causing other problems with delay biasing later on during a simulation.
run. For example take the following problem:

If we apply delay biasing on the $x < 4$ guard and get a delay of $2+$ seconds we will no longer be able to apply delay biasing on the next two guards. However if we split the first two guards in to two guards that go to the same location we could apply more efficient delay biasing. We would have to split the guards in a $x < 1$ guard and a $x >= 1$ guard that also includes the upper-bound of the original guard. This results in the following model for our problem:

Using guard splitting can result in an even greater variance reduction if applied properly. Do note that the new guard we do not use delay biasing on never gets used. It is only used to keep the model the same.

For a case study see 11.6.

**Low properties**

If properties like $P[\diamond x <= 1 \text{ Bad}]$ have to be model-checked then guard-splitting can also be used on the model to potentially achieve a greater variance reduction. An example:

On the first model delay biasing can not be applied while it is possible to perform delay biasing on the second model. An automatic process or the user will have to tell where and with what parameters delay biasing has to be applied.

**Clock constraint conjunctions**

It is possible to split guards containing a conjunction of 2 clock constraints into guards with only a single clock constraint. We will explain this process using an sample PTA.
Model A shows the original model with 3 edges of which two have guards containing a conjunction of two clock constraints.

Model B shows the PTA where guard gaps have been filled up with selfloops. The results is two more edges with both a conjunction of two clock constraints. Filling the selfloops is required to be able to split clock constraint conjunctions.

Model C shows a PTA where the conjunction with the lowest lower bound, has been split. The result is four new urgent locations and several new edges. The other edges coming out of location L1 have all been copied to the new urgent locations N2, N3, N4 (original edge removed). If the guard of the edge can never be satisfied at this point the edge can be removed again.

Model D shows a PTA where the above process is applied for the last guard with a conjunction of clock constraints.

The complete algorithm:

1. Fill all guard gaps with selfloops.
2. Split the clock constraint conjunction \((x >, \geq a \land x <, \leq b)\) with the lower lower bound in two parts, as followed:
   2a. Add four extra urgent locations. To make things easier we name these new locations N1, N2, N3, N4. Old locations are named L1, L2.
   2b. Delete the original clock constraint conjunction \((x >, \geq a \land x <, \leq b)\).
   2c. Add an edge from L1 to N1 with guard \(x >, \geq a\).
   2d. Add an edge from N2 to N4 with guard \(x <, \leq b\).
   2e. Add an edge from L1 to N2 with the compliment of the guard \(x >, \geq a\), which is \(x <, \leq a\).
   2f. Add an edge from N1 to N3 with the compliment of the guard \(x <, \leq b\), which is \(x >, \geq b\).
   2g. Add an edge from N4 to L2 with no guard.
3. All edges other than the one that was split gets copied and added to the locations N2, N3, N4. If the edge can never be enabled the edge does not have to be added.
(4) If all guards with conjunctions of clock constraints have been split then end algorithm, otherwise go back to step 2.

We do not provide a proof that this algorithm works.
Appendix B

Discrete-Event Simulation

Note: This is a chapter from the research topics report.

Discrete-event simulation (DES) is about the modeling of a system as it evolves over time. Systems are usually modeled with timed state-transition graphs. The state of the system is a set of variables that are necessary to describe the system at a certain point in time. The transitions in the graph give the different relationships between the various states. Systems are usually categorized in two types, namely discrete time and continuous time systems. Discrete systems change their states at fixed points in time without spending time on a transition jump, only in the states. Continuous systems change continuously in time. Systems can be simulated by executing a model of the system and tracking its various state changes over time. Deterministic systems can also be deterministic or stochastic in nature. Deterministic systems are systems in which the output is determined as soon as the input has been given. There are no random elements in deterministic systems. Non-deterministic systems one may jump, without a fixed probability, from one state to several possible next states. If systems do have random probabilistic elements then they are called stochastic systems. Simulation of stochastic systems, where time can run discretely or continuously, can also be called Discrete-Event Simulation.

In DES states change at separate discrete points in time, namely at those points in time where an event happens. For example think about a garage in which a state is represented by the number of cars waiting to be repaired and by whether the repairman is idle or busy. Car arrival (car broken) and departure (service completion/car fixed) are two examples of events. During an arrival either the waiting car count gets increased by one or the repairman starts immediately and becomes busy. During a departure the waiting car count gets decreased or the repairman becomes idle.

In DES the system model jumps from state to state by events. The order and the relative timing of the events is what we are usually interested in. The timing itself is based on either timed simulation, in which time is repeatedly increased by a constant where 0 or more events happen simultaneously or by event-based simulation, in which one event happens and time gets increased by the event-time.

More information on DES can be found in [16] and [27].
Appendix C

Statistical Model Checking

Note: This is a chapter from the research topics report.

In this chapter we will overview several SMC techniques. We assume that we have a system $S$ that we want to check properties $\varphi$ for. Normally $S$ is a whitebox system, which are systems in which the internal workings and the state transitions can be observed, but SMC techniques for (semi) black-box systems, in which the internal workings cannot be observed (only input and output), also exist. The property $\varphi$ is a bounded property, which means that the property can be defined on finite sample executions. $\varphi$ is usually defined in a temporal logic language like Continuous Stochastic Logic (CSL) or Probabilistic Computation Tree Logic (PCTL). The syntax from CSL and other information on CSL can be found in [47]. For SMC it is a requirement that we can get the result of any experiment in a finite amount of time. Otherwise the system could potentially be unsolvable and SMC would be worthless in that case.

The main SMC approaches in testing whether or not a property holds are based on hypothesis testing. More info can be found in [28], [46]. Hypothesis testing is the use of statistics to determine the probability that a given property is true or not. A hypothesis test uses data obtained from a number of executions, the sample set, to determine that probability.

C.1 Hypothesis testing

Consider a stochastic system $S$ and a property $\varphi$. An execution of a system $S$ is a possibly infinite sequence of states $s_i$, with times $t_i$ and events $e_i$ of $S$:

$$s_0 \xrightarrow{t_0, e_0} s_1 \xrightarrow{t_1, e_1} s_2 \xrightarrow{t_2, e_2} \ldots$$

Now let $B_i$ be a discrete random variable with a Bernoulli distribution of parameter $p$. This variable can only take on 2 values, namely 0 and 1 with:

$$Pr[B_i = 1] = p \quad \text{and} \quad Pr[B_i = 0] = 1 - p$$

The outcome for $B_i$, denoted $b_i$, is 1 if the sample execution path satisfies $\varphi$ and 0 otherwise. Another often used notation, with the same meaning, is the indicator function:

$$1_{\{b_i \text{ satisfies } \varphi\}} = \begin{cases} 1 & \text{if } b_i \text{ satisfies } \varphi \\ 0 & \text{otherwise} \end{cases}, \quad \text{with } Pr[1_{\{b_i \text{ satisfies } \varphi\}} = 1] = p.$$ 

In hypothesis testing one is interested in testing whether the parameter $p$ of the Bernoulli distribution is above or below some given threshold. We know that $Pr[B_i = 1] = p$, to determine whether or not $p \geq \theta$, we can test $H_0$: $p \geq \theta$ against $H_1$: $p \leq \theta$. $\theta$ is a, by the user, supplied value. However this can give problems when the probability $p$ is very close to or equal to $\theta$ because it would require a gigantic number of samples before a conclusion can be made. A test-based solution would not be able to guarantee a correct result, that is
why (among other reasons we will not go into) we have to introduce an indifference region of width \(2 \cdot \delta\). With this indifference region the hypothesis test will change to a test for checking \(H_0 : p \geq \theta + \delta\) against \(H_1 : p \leq \theta - \delta\). If the value of \(p\) is between \(\theta + \delta\) and \(\theta - \delta\) (the indifference region), then we say that the probability is so close to \(\theta\) so that we are indifferent with respect to which of the two hypotheses \(H_0\) or \(H_1\) is accepted. If \(p\) does not fall inside the indifference region then we will accept either \(H_0\) (when \(p \geq \theta\)) or \(H_1\) (when \(p \leq \theta\)).

Since even with indifference regions it is not possible to completely guarantee that a property holds when using simulation, there is always a chance that an error is made. There are 2 possible kinds of errors, usually denoted with \(\alpha\) and \(\beta\). \(\alpha\) is the probability of accepting \(H_1\) when \(H_0\) holds (Type-1 error, or error of the first kind, or false negative). \(\beta\) is the probability of accepting \(H_0\) when \(H_1\) holds (Type-2 error, or error of the second kind, or false positive). Mathematical notation:

\[
\Pr[H_0 \text{ holds } | \text{ accept } H_1] \leq \alpha \\
\Pr[H_1 \text{ holds } | \text{ accept } H_0] \leq \beta
\]

\(\alpha\) and \(\beta\) combined give the strength of a test. Generally it is assumed that both \(\alpha\) and \(\beta\) are less than \(\frac{1}{2}\). If they were both at least \(\frac{1}{2}\), the chance for an error would become too large for any test to be meaningful. The probabilities for correctly determining whether or not a property holds or is inconclusive are given by:

\[
\Pr[H_1 \text{ holds } \lor \text{ indifference } | \text{ accept } H_1] \geq 1 - \alpha \\
\Pr[H_0 \text{ holds } \lor \text{ indifference } | \text{ accept } H_0] \geq 1 - \beta
\]

### C.2 Single Sampling Plan

In [46] Younes showed a technique called the single sampling plan to check whether or not a property holds. In this fixed size sample test we test \(H_0 : p \geq \theta + \delta = p_0\) against \(H_1 : p \leq \theta - \delta = p_1\). To do this we specify a constant \(c\) and a sample size \(n\). If

\[
f(b_1, \ldots, b_n) = \sum_{i=1}^{n} b_i > c
\]

then we accept \(H_0\) else we accept \(H_1\). An indifference hypothesis does not exist, so after \(n\) samples we know that either \(H_0\) or \(H_1\) holds. The probability of \(f(b_1, \ldots, b_n)\) being at most \(c\) is given by the cumulative binomial distribution function:

\[
F(c; n, p) = \sum_{i=0}^{c} \binom{n}{i} p^i (1 - p)^{n-i}
\]

So with probability \(F(c; n, p)\) we accept hypothesis \(H_1\) using parameters \((n, c)\), and with probability \(1 - F(c; n, p)\) we accept hypothesis \(H_0\) (with the same parameters \((n, c)\)).

When \(n\) is large the binomial distribution can be approximated by using the normal distribution with mean \(np\) and variance \(np(1 - p)\). \(H_1\) is accepted when \(\sum_{i=1}^{n} b_i \leq c\), which can be calculated using the standard normal cumulative distribution:

\[
\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt
\]  
(C.2)

The approximate probability of accepting \(H_1\) becomes:
\[ F(c; n, p) \approx \Phi \left( \frac{c - np}{\sqrt{np(1-p)}} \right) \] (C.3)

The approximate probability of accepting \( H_0 \) is one minus the probability of accepting \( H_1 \).

The hardest part of the single sampling plan (SSP) is finding the parameter pair \((n, c)\), also called the single sampling plan. For the SSP to have strength \((\alpha, \beta)\) it is required to find \((n, c)\) such that:

1. \[ F(c; n, p_0) \leq \alpha \]
2. \[ 1 - F(c; n, p_1) \leq \beta \]

It is generally preferred to have sampling plans that use small sample sizes over those that use large sample sizes, for efficiency reasons obviously. So we want to minimize \( n \) while (1) and (2) hold. Unfortunately this minimization problem does not have a simple solution. Younes did however give an algorithm for finding a good single sampling plan in [46] when supplied with \( p_0(=\theta + \delta), p_1(=\theta - \delta), \alpha \) and \( \beta \).

Another Fixed Sample Size Test

In [35] another fixed size sample test was given. However this test was slightly different and had three different hypotheses:

- \( H_0: p = p_0 \).
- \( H_{+1}: p > p_0 \).
- \( H_{-1}: p < p_0 \).

In this test a \( p_0 \) is supplied. The purpose of the test is to see if the actual probability \( p \) of the system is larger (accept \( H_{+1} \)) or smaller (accept \( H_{-1} \)) than \( p_0 \). \( H_0 \) stands for indifference and is used when neither \( H_{+1} \) nor \( H_{-1} \) can be accepted. The test boils down to the following:

- If \( \sum_{i=0}^{c} \binom{n}{i} p^i(1-p)^{n-i} \geq 1 - \frac{\alpha}{2} \) accept \( H_{+1} \).
- If \( \sum_{i=0}^{c} \binom{n}{i} p^i(1-p)^{n-i} \leq \frac{\alpha}{2} \) accept \( H_{-1} \).

These acceptance bounds can be approximated with the normal distribution function from (C.2):

- If \( \Phi \left( \frac{\sum_{i=0}^{c} b_i - Np_0}{\sqrt{Np_0(1-p_0)}} \right) \geq 1 - \frac{\alpha}{2} \) accept \( H_{+1} \).
- If \( \Phi \left( \frac{\sum_{i=0}^{c} b_i - Np_0}{\sqrt{Np_0(1-p_0)}} \right) \leq \frac{\alpha}{2} \) accept \( H_{-1} \).

The test creates an upper and a lower bound for accepting \( H_{+1} \) or \( H_{-1} \) respectively:

- \( u(N) = \Phi^{-1}(1 - \frac{\alpha}{2})\sqrt{Np_0(1-p_0)} \)
- \( l(N) = \Phi^{-1}(\frac{\alpha}{2})\sqrt{Np_0(1-p_0)} \)

The difference between the two papers is that Younes in [46] indirectly assumes that the probability \( p \) cannot fall in the indifference region initially specified. Either \( H_0 : p > \theta + \delta = p_0 \) or \( H_1 : p < \theta - \delta = p_1 \) is accepted after \( N \) samples have been tested. Younes tries to find a proper \((n, c)\) via an algorithm using \( p_0, p_1, \alpha, \beta \) in such a way that the test obeys strength \((\alpha, \beta)\). Reijsbergen in [35] does not assume that \( p \) cannot fall inside an
indifference region. His test either accepts $H_{+1} : p > p_0$ or $H_{-1} : p < p_0$ and when neither of these is accepted $H_0 : p = p_0$, which shows indifference, remains valid. Younes' test will give false results when $p \in [p_1, p_0]$.

C.3 Sequential Tests

It is often inefficient to do a fixed number of tests to determine whether or not a hypothesis should be accepted or not. Take for example the single sampling plan. If we reach $c$ after $n-5$ samples we will know that $H_0$ can be accepted without having to do more any more tests, even if the 5 tests are negative. An algorithm could easily be created, which Younes does in [46], for this sequential single sampling plan. One would only have to check after each test to see whether or not the value $c$ has been reached. Do note that this does not work with the second fixed sample size test from Reijsbergen showed in [35].

![Figure C.1: The Sequential Single Sampling Plan. $m = n.$ From [46].](image)

C.3.1 SPRT

In [41] Abraham Wald created a test called the sequential probability ratio test (SPRT) that works similarly to the sequential single sampling plan. The SPRT is efficient because it minimizes the number of samples needed for determining whether to accept a hypothesis $H_0$ or $H_1$. $H_0$ is accepted when $p \geq p_0$ and $H_1$ is accepted when $p \leq p_1$. Usually $p_0 = \theta + \delta$ and $p_1 = \theta - \delta$ are chosen for $p_0$ and $p_1$.

Basically the SPRT creates two boundaries. An upper boundary for $p_0$ and a lower boundary for $p_1$. Next to the 2 boundaries we have a quantity function where after $n$ executions $x_1, \ldots, x_n$ the following is calculated:

$$p_{1n} \quad p_{0n} = \prod_{i=1}^{n} \left[ \frac{X_i = x_i | p = p_1}{X_i = x_i | p = p_0} \right] = \frac{p_1^{d_n} (1 - p_1)^{n - d_n}}{p_0^{d_n} (1 - p_0)^{n - d_n}}$$

Here $d_n$ equals the number of times a formula $\varphi$ is satisfied, or $d_n = \sum_{i=1}^{n} x_i$. $H_0$ is accepted when:

$$\frac{p_{1n}}{p_{0n}} \leq \frac{(1 - \beta)}{\alpha}$$

And $H_1$ is accepted when:

$$\frac{p_{1n}}{p_{0n}} \geq \frac{\beta}{(1 - \alpha)}$$

For when the test gives an approximate strength of $(\alpha, \beta)$. When neither $H_0$ nor $H_1$ is accepted another sample is generated and the quantity function is recalculated for $n + 1$ sample executions.
Nested Probabilistic operators

The above test can be used for determining formulas \( \varphi \) that contain a single probabilistic operator \( (\varphi = \Pr_{\geq \theta}(\rho)) \) without error (beside \( \alpha \) and \( \beta \)). When \( \varphi \) contains nested probabilistic operators, so \( \rho \) also contains a probabilistic operator, then the above SPRT formula does not work any longer and needs to be altered slightly. More information on nested operators and the details on the altered formulas can be found in [47].

For systems in which the transition probabilities, \( p_0 \) and \( p_1 \) are unknown and no assumptions about the underlying transition structure are made, formulas can be found in [38].

C.3.2 Supergaussian Sequential Tests

In [35] two tests are discussed that are also based on sequential testing. These 2 tests are called the Azuma test and the Darling test. Both test generate an upper and a lower boundary for testing the null hypothesis \( H_0 \) against 2 alternative hypotheses \( H_+1 \) and \( H_{-1} \). As soon as the upper \( (U) \) boundary is crossed \( H_{+1} \) is accepted and as soon as the lower \( (L) \) boundary is crossed \( H_{-1} \) is accepted. If the test statistic stays between the 2 boundaries, i.e. when it falls in the non-critical area \( (NC) \), another sample is generated and the test is retried. Figure C.3 shows the shape of both tests. \( Z_N(= \sum_{i=1}^{n} b_i - N p_0) \) is the normalized test statistic.

The Azuma test has the following non-critical area:

\[
[-a(N + k)^{\frac{3}{4}}, a(N + k)^{\frac{3}{4}}]
\]

The Darling test has the following non-critical area:

\[
[-\sqrt{a(N + 1) \log(N + k)}, \sqrt{a(N + 1) \log(N + k)}]
\]

Figure C.2: The Sequential Probability Ratio Test. \( m = n \). From [46].

Figure C.3: Acceptance graph. From [35].
\( a \) can be seen as a parameter that increases the width of the non-critical area. Looking at the non-critical areas from both tests we can see that when \( n = 0 \) or close to 0 the only value that matters is \( k \). A high value of \( k \) makes the boundaries start further away from \( Z_N = 0 \), which basically looks like as if the boundaries have slightly shifted to the left. So a high value of \( k \) makes it harder to reject at the beginning. Since the two tests both depend on both \( a \) and \( k \), \( a \) and \( k \) have to be chosen in such a way that the two tests still obey strength \((\alpha, \beta)\). Due to this strength \( a \) and \( k \) depend on each other. This gives us one degree of freedom to freely choose either \( a \) or \( k \) the way we want.

\( k \) can be chosen with:

\[
\begin{align*}
  k_{Azuma}(a, \alpha) &= \left(\frac{\log \frac{1}{\alpha}}{4a^4}\right)^2 \\
  k_{Darling}(a, \alpha) &= \left(\frac{(a(\alpha-1))}{2\sqrt{2}}\right)^{-\frac{1}{\alpha-1}} - 1
\end{align*}
\]

Formula’s for \( a \) can be obtained by rewriting the above \( k \)-formulas. For more information see [35].

Figure C.4 shows the acceptance boundaries for all four tests we have discussed. The gauss test is the fixed size sample test we discussed from [35]. All tests are normalized.

![Acceptance graph for all 4 tests. From [35.]](image)
Bibliography


[2] The model checker PRISM; Parker, D. and Norman, G. and Kwiatkowska, M.


