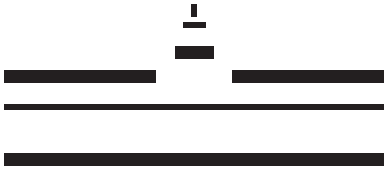


Westfälische
Wilhelms-Universität
Münster

Vergleich und Implementierung von Hypothesentests für das statistische Model Checking

Bachelorarbeit
zur Erlangung des akademischen Grades
Bachelor of Science

Westfälische Wilhelms-Universität Münster
Fachbereich Mathematik und Informatik

Betreuung:
Prof. Dr. Anne Remke

Eingereicht von:
Fabian Edenfeld

Münster, April 2017

Inhaltsverzeichnis

1 Einführung

In der heutigen Zeit der weltweiten Vernetzung und der fortschreitenden Automatisierung durch hoch entwickelte Technologie sind geordnete Systemstrukturen zu einer absoluten Notwendigkeit geworden deren Komplexitätsgrad stetig ansteigt. Daher ist es wichtig, dass diese Systeme effizient und fehlerfrei arbeiten beziehungsweise möglichst selbständig mit Fehler umgehen können. Dies erfordert eine entsprechend detaillierte Planung und Modellierung. Die sogenannten *Petri-Netze* bieten dabei ein graphentheoretisches Werkzeug zur Modellierung solcher Systeme, welches sich leicht an unterschiedlichste Anforderungen anpassen lässt und selbst das detaillierte Modellieren von hochgradig komplexen Systemen ermöglicht. Beispiele dafür sind Modellierungen von Wasserversorgungssystemen ([2]), Fertigungssystemen ([1]) oder die Nutzung von Modellierungen für die Systemanalyse ([11]). Die Simulation solcher Modellierungen ist eine effektive Methode, um diese auf ihre Effizienz, Korrektheit und eventuelle Schwachstellen zu prüfen oder die allgemeine Arbeitsweise zu analysieren. Im Kontext dieser Systemprüfungen werden unter anderem auch sogenannte *Model Checks* angewendet. Das Verfahren des Model Checkings beschreibt dabei die automatisierte Prüfung eines Modells bezüglich der Erfüllung einer modellspezifischen Eigenschaft. Wenn man für die Auswertung solcher Model Checks statistische Methoden verwendet, um einen höheren Grad an Sicherheit bezüglich der Resultate des Verfahrens zu erzielen, spricht man vom *statistischen Model Checking*. Eine dieser statistischen Methoden ist das Durchführen von *Hypothesentests*. Mit Hilfe eines Hypothesentests kann zum Beispiel geprüft werden, ob die Wahrscheinlichkeit, dass das Modell einen bestimmten kritischen Zustand erreicht, möglicherweise über einer strikten Toleranzgrenze liegt.

Diese Arbeit befasst sich nun mit der Nutzung von Hypothesentests im Rahmen des Statistical Model Chekings. Ziel der Arbeit ist es, drei Hypothesentestalgorithmen darzustellen und in ein bestehendes Tool zur Simulation sogenannter *hybriden Petri-Netzen* und der Durchführung von Model Checks zu integrieren.
Zunächst folgt in Kapitel 2 eine Darstellung von Petri-Netzen, insbesondere hybrider Petri-Netze. Es wird darauf eingegangen, worum es sich bei einem solchen Petri-Netz handelt und es werden die für die Simulation relevanten Eigenschaften präsentiert. In Kapitel 3 wird dann die generelle Funktionsweise der Simulation erläutert und zudem die Thematik des statistischen Model Checkings näher dargestellt und gezeigt, wie solche Model Checks mit Hilfe des Tools durchgeführt werden können. Danach folgt in Kapitel 4 eine grundlegende Erklärung von Hypothesentests im Allgemeinen sowie die Präsentation der für diese Arbeit gewählten Hypothesentestalgorithmen. In Kapitel 5 wird die Implementierung dieser Algorithmen dargelegt. Anschließend werden die Hypothesentestestalgorithmen in Kapitel 6, durch mehrere Testdurchläufe auf ihre Performance geprüft

und die beobachteten Ergebnisse zusammengefasst.
Das Simulationstool entstand im Rahmen von [8]. Die Grundlagen der Hypothesentestalgorithmen sind [10] entnommen, das Kapitel bezüglich Petri-Netze entstammt [3].

2 Petri-Netze

Bei Petri-Netzen handelt es sich um gerichtete, gewichtete, bipartite Graphen, welche zur Modellierung in vielerlei Bereichen Anwendung finden, wie zum Beispiel in der Leistungsbewertung verschiedener Systeme oder der Modellierung von Informationsverarbeitungssystemen ([7]). Dabei erlauben Petri-Netze sowohl die Modellierung paralleler Abläufe, als auch den Einsatz verschiedener Systemcharakteristika in Form verschiedener grafischer Komponenten. Die für diese Arbeit relevanten Komponenten, beziehungsweise Arten von Petri-Netzen, insbesondere hybride Petri-Netze mit stochastischen Transitionen, werden in diesem Kapitel vorgestellt.
Zunächst werden in 2.1 und 2.2 anhand des diskreten und stetigen Modells die allgemeine Funktionsweise der Modellierung mit Petri-Netzen erläutert. Danach folgt die Präsentation der hybriden Petri-Netze in 2.3 mit stochastischen Transitionen, welche detaillierter ausfällt, da es sich um das im Tool verwendete Modell handelt. Im Anschluss folgen einige Definitionen, welche für die Modellierung mit Petri-Netzen von Bedeutung sind.
Der Inhalt dieses Kapitels beruht hauptsächlich auf [3], jedoch gibt es in Bezug auf die Modellierung der hybriden Petri-Netze einige Unterschiede zum Tool, welche hervorgehoben werden.

2.1 Diskrete Petri-Netze

Ein diskretes Petri-Netz ist ein gerichteter, gewichteter, bipartiter Graph. Bei den Knoten eines diskreten Petri-Netzes unterscheidet man zwischen Zuständen und Transitionen. Jedem Zustand des Graphen wird dabei eine nicht negative Anzahl an sogenannten Tokens zugewiesen, welche in der grafischer Darstellung als schwarze Punkte auf den jeweiligen Zuständen liegen. Die Gesamtheit der Verteilung der Tokens auf die Zustände des Graphen wird als *Markierung* bezeichnet. Im Verlauf des Modellierungsprozesses kann der Graph beziehungsweise die aktuelle Markierung des Graphen verändert werden. Dies geschieht über die Transitionen, welche über gerichtete, gewichtete Kanten mit einzelnen Zuständen verbunden sind. Im Zuge der Modellierung sind diese Transitionen in der Lage, zusätzliche Tokens für verbundene Zustände zu generieren und bereits vorhandene Tokens zu entfernen, wodurch sich die Verteilung der Tokens, das heißt die Markierung des Graphen, ändert. Die Transitionen selbst können jedoch keine Tokens halten. Diesen Prozess nennt man *Schaltung*. Auf welche Art Tokens hinzugefügt und entnommen werden, ist durch die Kantengewichtung und Ausrichtung vorgegeben. Ist bei einer Kante, welche eine Transition mit einem Zustand verbindet, die Transition der Startknoten und der Zustand der Endknoten, so werden gemäß des korrespondierenden Kantengewichts dem Zustand neue Tokens hinzugefügt. Handelt es sich andererseits bei der schaltenden Transition um den Endpunkt und bei dem Zustand um den Startpunkt,

werden stattdessen Tokens entfernt, ebenfalls gemäß des Kantengewichts. Damit eine Transition aber überhaupt einen solchen Schaltprozess beginnen kann, muss sie erst aktiviert sein. In der Modellierung mit diskreten Petri-Netzen gilt eine Transition genau dann als aktiviert, wenn die Zustände, welche über eine hinführende Kante mit der Transition verbunden sind, mindestens so viele Tokens halten, wie das Kantengewicht der jeweiligen Kante vorgibt. Allerdings wird der Schaltprozess nicht bei jeder Transition mit der Aktivierung eingeleitet, viel mehr unterscheidet man zwischen drei Arten von Transitionen. Die *direkten Transitionen*, welche mit der Aktivierung schalten, die *deterministischen Transitionen*, welche nach der Aktivierung erst nach Ablauf einer bestimmten Zeitspanne schalten und die *stochastischen Transitionen*, welche ähnlich wie die deterministische Variante zeitgesteuert schalten, die Zeitspanne wird jedoch durch eine stochastische Verteilungsfunktion mit jeder Aktivierung neu berechnet.
Die Definition eines solchen diskreten Petri-Netzes ist dabei durch folgende Definition gegeben:

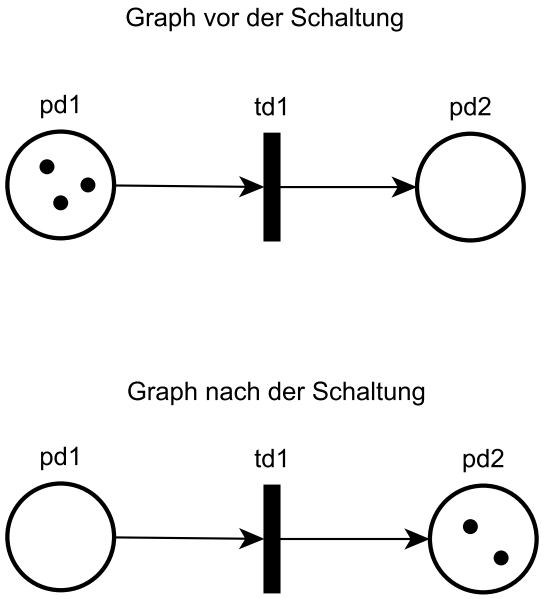
Definition. Ein diskretes Petri-Netz ist ein Tupel $PN^d = (P^d, T, A^d, m_0, \phi_w^A)$. Dabei ist P^d die Menge aller diskreten Zustände. Die Menge der Transitionen ist gegeben durch $T = T^I \cup T^D \cup T^G$, wobei (T^I) die Menge der direkten, (T^D) die Menge der deterministischen und (T^G) die Menge der stochastischen Transitionen darstellt. A^d ist die Menge aller diskreten Kanten. Die initiale Verteilung der Tokens auf die Zustände des Petri-Netzes ist durch die initiale Markierung m_0 angegeben. Jeder Kante $A_i^d \in A^d$ wird über die Funktion ϕ_w^A ein Kantengewicht zugewiesen.

(Anmerkung: diese Definition basiert auf [7] S.543, wobei die Variablennamen aus [3] verwendet wurden)

Im Folgenden soll an einem Beispiel verdeutlicht werden, wie sich die Schaltung einer Transition auf ein diskretes Petri-Netz auswirken kann.

Beispiel. Sei $PN = \{P^D, T, A^d, m_0, \phi_w^A\}$ ein Petri-Netz mit den diskreten Zuständen $P_1^d, P_2^d \in P^d$ (in Abbildung 2.1 dargestellt durch pd1, pd2), der direkten Transition $T_1^I \in T$ (in Abbildung 2.1 dargestellt durch td1) und den Kanten $A_1^d, A_2^d \in A^d$, für die gilt: $A_1^d = (P_1^d, T_1^I)$, $A_2^d = (T_1^I, P_2^d)$ sowie $\phi_w^A(A_1^d) = 3$, $\phi_w^A(A_2^d) = 2$. Sei weiterhin gegeben, dass unter der aktuellen Markierung m_0 drei Tokens auf P_1^d und null Tokens auf P_2^d liegen. Wenn die Transition T_1^I nun schaltet, bedeutet dies, dass gemäß des Kantengewichts von A_1^d drei Tokens von P_1^d entnommen und, gemäß des Gewichts von A_2^d, zwei Token auf P_2^d gelegt werden.
Somit entsteht ein neuer Graph, mit der Markierung m_1

Abbildung 2.1: Grafische Darstellung von 2.1 Quelle: Eigene Darstellung



Wenn eine Transition keine eingehenden Kanten besitzt, spricht man von einer *Zufluss-Transition*, besitzt sie keine ausgehenden Kanten von einer *Abfluss-Transition*. Eine *Zufluss-Transition* kann dabei lediglich Tokens generieren, während eine *Abfluss-Transition* lediglich Tokens entfernen kann.

2.2 Stetige Petri-Netze

Die Modellierung mit einem stetigen Petri-Netz ist in ihren Grundzügen der diskreten Variante recht ähnlich, dennoch gibt es einige signifikante Unterschiede. So halten die Zustände eines stetigen Petri-Netzes keine ganzzahlige Anzahl an Tokens, sondern verfügen stattdessen über einen nicht negativen Wert an sogenannter *Flüssigkeit*, welche als reelle Zahl angegeben wird. Dabei können stetige Zustände sowohl limitierte wie auch unlimitierte Flüssigkeitskapazitäten besitzen. Die Markierung des Graphen beschreibt daher die aktuellen Flüssigkeitsstände der Zustände. Ein weiterer Unterschied besteht in der Funktionsweise von Transitionen. Während man bei diskreten Transitionen von einzelnen Schaltprozessen spricht, entsteht durch eine stetige Transition ein konstanter Zu- oder Abfluss an Flüssigkeit, dessen Stärke von der Flussrate der Transition und dem Gewicht der jeweiligen Kante bestimmt wird. Wenn eine stetige Transition über eine stetige Kante mit einem stetigen Zustand verbunden ist und die Transition den Startknoten bildet, so handelt es sich um einen Zufluss an Flüssigkeit zum verbundenen Zustand hin, stellt sie den Endknoten dar, fließt Flüssigkeit vom Zustand ab.

2.3 Hybride Petri-Netze

Die folgende Definition basiert größtenteils auf [3] (S.25-26), jedoch existieren einige Unterschiede zu der in [8] verwendeten Modellierung. Diese Unterschiede werden genauer hervorgehoben.
Besitzt ein Petri-Netz sowohl diskrete als auch stetige Elemente, so spricht man von einem *hybriden Petri-Netz*. Ein solches Petri-Netz ist ein Tupel $(P, T, A, m_0, x_0, \phi)$ mit der Definition

Definition.

$$\begin{aligned} P &= P^d \cup P^c \\ T &= T^I \cup T^D \cup T^C \cup T^G \\ A &= A^d \cup A^c \cup A^h \cup A^t \\ m_0 &= \{m_1, m_2, ..., m_{n_d}\} \\ x_0 &= \{x_1, x_2, ..., x_{n_c}\} \\ \phi &= (\phi_b^P, \phi_w^T, \phi_p^T, \phi_d^T, \phi_f^T, \phi_g^T, \phi_w^A, \phi_s^A, \phi_p^A) \end{aligned}$$

P ist die Menge der diskreten (P^d) und der stetigen Zustandsmengen (P^c).

$T = T^I \cup T^D \cup T^F \cup T^G$ umfasst die Mengen der direkten ($T^I = \{T_1^I, T_2^I ..., T_{n_I}^I\}$), deterministischen ($T^D = \{T_1^D, T_2^D ..., T_{n_D}^D\}$), stetigen ($T^F = \{T_1^F, T_2^F ..., T_{n_F}^F\}$) und stochastischen Transitionen ($T^G = \{T_1^G, T_2^G ..., T_{n_G}^G$). Um die weitere Definitonsaufbereitung anschaulicher zu gestalten, sind die direkten, deterministischen und stochastisch verteilten Transitionen als die Menge der *Standard Transitionen* $T^S = T/T^F$ zusammengefasst.

A beinhaltet die Mengen A^d, A^f, A^h, A^t. Dabei ist $A^d \subseteq ((P^d \times T^S) \cup (T^S \times P^d))$ die Menge der aus-, und eingehenden Kanten der diskreten Zustände, analog dazu ist $A^f \subseteq ((P^c \times T^F) \cup (T^F \times P^c))$ die Menge der aus-, eingehenden Kanten der stetigen Zustände. Bei A^t und A^h handelt es sich um sogenannte *Test-* und *Hemm*-Kanten, zusammenfassend auch als *Wächterkanten* bezeichnet. Eine Test-Kante verbindet eine beliebige Art von Zustand mit einer beliebigen Art von Transition. Eine so verbundene Transition gilt dann als aktiviert, wenn die Markierung oder der Flüssigkeitsstand des korrespondieren Zustands eine durch das Kantengewicht der *Test*-Kante vorgegebene Bedingung erfüllt (zum Beispiel, dass ein verbundener, diskreter Zustand eine bestimmte Anzahl Tokens hält). Die Funktion einer Hemm-Kante fällt ähnlich aus, allerdings darf für die Aktivierung der verbundenen Transition die korrespondierende Bedingung nicht erfüllt sein. Die Definitionen der Mengen der Wächterkanten sind gegeben durch $A^t \subseteq (P \times T)$ und $A^h \subseteq (P \times T)$. Die Test- und Hemm-Kanten dienen dazu, den diskreten und den stetigen Teil des Petri-Netzes miteinander zu verbinden, indem sich so der Fluss stetiger Transitionen über den diskreten Teil betreffende Bedingungen starten, beziehungsweise stoppen lässt.

Wie auch bei den diskreten und stetigen Petri-Netzen geben m_0 die initiale Markierung

und x_0 die initialen Flüssigkeitsstände an.

Zuletzt findet durch das Tupel $\phi = (\phi_b^P, \phi_w^T, \phi_p^T, \phi_d^T, \phi_f^T, \phi_g^T, \phi_w^A, \phi_s^A, \phi_p^A)$ die Verteilung der Parameter des Graphen statt. Die Funktion $\phi_b^P : P^c \to \mathbb{R}^+$ gibt für jeden stetigen Zustand $p_i^c \in P^c$ einen maximalen Grenzwert für den Flüssigkeitsstand an, wobei dieser auch unendlich sein kann.

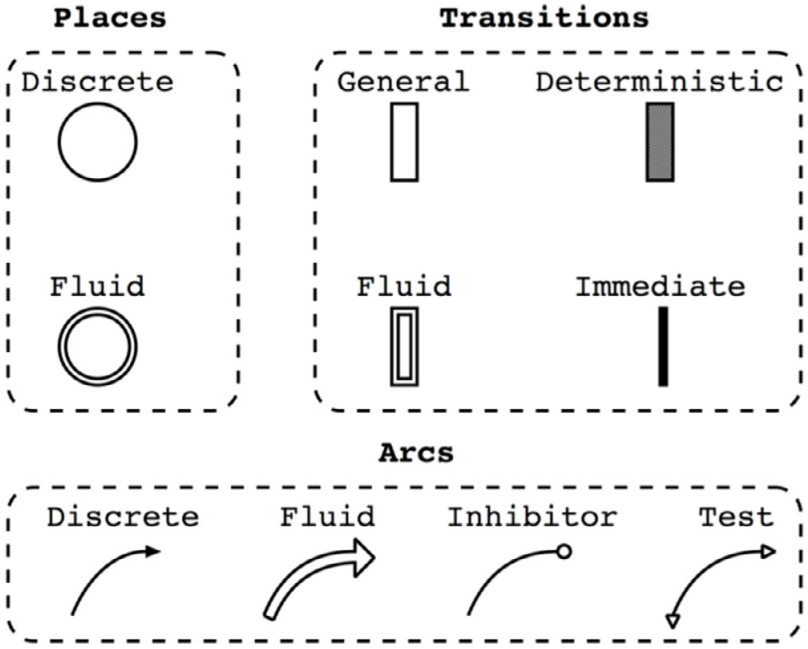
Bei den Transitionen der Menge T^S des hybriden Petri-Netzes kann es dazu kommen, dass mehrere Schaltungen für den gleichen Zeitpunkt vorgesehen sind. Während der Simulation wird allerdings immer nur eine Schaltung gleichzeitig bearbeitet (siehe 3.1). Um solche Schaltungskonflikte aufzulösen, werden Gewichtungen, beziehungsweise Prioritäten für diese Transitionsarten eingefügt. Die Funktion $\phi_w^T : T^S \to \mathbb{R}^+$ teilt den Transitionen eine Gewichtung und $\phi_p^T : T^S \to \mathbb{N}$ eine Priorität zu. Es gilt zu beachten, dass in [3] dabei nur deterministische und direkte Transitionen betrachtet werden, die in [8] verwendete Modellierung sieht es aber vor, dass auch stochastische Transitionen mit entsprechenden Werten versehen werden. Wie diese Konfliktlösung im Laufe der Simulation durchgeführt wird in 2.3.2 näher beschrieben.

Die Schaltzeiten für deterministische Transitionen wird über $\phi_d^T : T^F \to \mathbb{R}^+$ angegeben, die Stärke des Zu- oder Abflusses einer stetigen Transition wird über $\phi_f^F : T^F \to \mathbb{R}^+$ berechnet.

Wie in 2.1 dargestellt, schalten die stochastischen Transitionen nach einer Zeit, welche durch eine Verteilungsfunktion bestimmt wird. Die Funktion $\phi_g^T : (T^G \times \mathbb{R}^+) \to [0, 1]$ gibt dabei die Wahrscheinlichkeit an, mit der eine stochastisch verteilte Transition $t_i^G \in T^G$ vor einem Zeitpunkt τ schaltet. Dabei ist zu beachten, dass die in [3] beschriebenen stochastisch verteilten Transitionen nur einmalig schalten können, während die im Tool verwendete Modellierung auch das mehrfache Schalten einer solchen Transition zulässt ([8], S.6). Näheres wird in 2.3.1 beschrieben.

Die Funktion $\phi_w^A : A \to \mathbb{R}^+$ gibt die Kantengewichte der einzelnen Kanten des Petri-Netzes an. Das Gewicht einer stetigen Kante $a_i^f \in A^f$ ergibt dabei, multipliziert mit dem Wert der Flussrate $\phi_f^F(t_i^F)$ einer verbundenen Transition t_i^F, die tatsächliche Stärke des korrespondierenden Abflusses, beziehungsweise des Zuflusses an. Zudem erhalten stetige Kanten zusätzlich noch eine *share*-Funktion $\phi_s^A : A^f \to \mathbb{R}^+$ und eine Prioritätsfunktion $\phi_p^A : A^f \to \mathbb{N}$, welche dann genutzt werden, wenn ein stetiger Zustand bezüglich des Flüssigkeitsstand seine Maximal- oder Mindestgrenze erreicht hat. Die Behandlung solcher Fälle wird in 2.3.3 kurz beschrieben.
In Bild 2.2 sind die Komponenten der hybriden Petri-Netze abgebildet.

Abbildung 2.2: Komponenten hybrider Petri-Netze. Quelle: [3], S.25



Erläuterung: Discrete = diskreter Zustand, Fluid = stetiger Zustand,
General = stochastische Transition, Deterministic = deterministische Transition,
Fluid = stetige Transition, Immediate = direkte Transition.

2.3.1 Zustände hybrider Petri-Netze

Der aktuelle Zustand eines hybriden Petri-Netzes (nicht zu verwechseln mit den einzelnen Zuständen als Knoten des Graphen) ist ein Tupel $\Gamma = (m, x, c, d, g, e)$. Die einzelnen Variablen repräsentieren dabei Vektoren, welche zur Sicherung der Parameter genutzt werden, die sich im Laufe der Modellsimulation verändern. Wie in 2.3 dargelegt, beinhaltet m die Markierungen der Zustände des diskreten Teils des Petri-Netzes und x die Flüssigkeitsstände der Zustände des stetigen Teils.

In $c = (c_1, c_2, ..., c_{n_D})$ mit $c_i \in \mathbb{R}_0^+$ werden die sogenannten *Uhren* abgespeichert. Eine solche Uhr $c_i \in c$ steht dabei für die Zeit, die seid der Aktivierung der zugehörigen deterministischen Transition $T_i^D \in T^D$ vergangen ist. Sobald $\phi_d^T(T_i^D)$ der Uhr c_i entspricht, schaltet T_i^D, worauf die Uhr zurückgesetzt wird. Sollte eine deterministische Transition wieder deaktiviert werden, so bleibt die Uhr erhalten und läuft erst nach der Reaktivierung weiter.

In dem Vektor $d = (d_1, d_2, ..., d_{n_c})$ sind die *Drifts* der einzelnen stetigen Transitionen abgespeichert. Ein solcher Drift $d_i \in d$ mit $d_i \in \mathbb{R}^+$ steht dabei für die Veränderung des Flüssigkeitsstandes von $T_i^F \in T^F$ pro Zeiteinheit.

Der Vektor $g = (g_1, g_2, ..., g_{n_G})$ beinhaltet die Aktivierungszeiten der stochastischen Transitionen. Das Prinzip ist gleich dem der Uhren, das heißt $g_i \in g$ ist die Zeit

seid der Aktivierung von $T_i^G \in T^G$, wobei die Messung bei Deaktivierung von T_i^G angehalten und mit der Schaltung von T_i^G zurückgesetzt wird. Aufgrund der *one-shot*-Eigenschaft der in [3] vorgestellten stochastischen Transitionen sind die Aktivierungszeiten mit $g_i \in \mathbb{R}_0^+ \cup \{-1\}$ definiert, wobei $g_i = -1$ bedeutet, dass T_i^G bereits geschaltet hat und demnach zu keiner weiteren Schaltung freigegeben werden kann. Wie bereits erwähnt, können die im Simulationstool verwendeten Transitionen jedoch auch mehrfach schalten, weshalb stattdessen die Definition $g_i \in \mathbb{R}_0^+$ genutzt wird ([8], S.9).

Zusätzlich ist zu beachten, dass die über die entsprechende Verteilungsfunktion bestimmte Schaltzeit nach jeder Schaltung wieder verworfen und mit der Reaktivierung neu berechnet wird. Für den Fall, dass eine Transition deaktiviert und mit fortschreitender Laufzeit der Simulation wieder reaktiviert wird, unterscheidet man drei verschiedene Vorgehensweisen, wie man mit der in g abgelegten Aktivierungszeit und der aktuell berechneten Schaltzeit umgeht. Diese Vorgehensweisen lauten *Resume, Repeat identical* und *Repeat different.*
Bei *Resume* wird die Aktivierungszeit gespeichert und mit der Reaktivierung die vorherige Schaltzeit wieder aufgenommen.
Bei *Repeat identical* wird die Aktivierungszeit zurückgesetzt, die vorherige Schaltzeit wird wie bei *Resume* weiter verwendet.
Im Falle von *Repaet different* wird mit der Reaktivierung eine neue Schaltzeit berechnet.

(Anmerkung: Um das Verständnis dieses Abschnitts zu sichern sei an dieser Stelle noch einmal explizit gesagt, dass die Wiederaufnahme der zuvor berechneten Schaltzeit bei *Repeat identical* und *Resume* sich auf den Fall einer Reaktivierung bezieht. Sobald Transitionen, die mit diesen Vorgehensweisen arbeiten schalten, wird eine neue Schaltzeit berechnet) Zuletzt beinhaltet der Vektor $e = (e_1, e_2, ..., e_i)$ den Status jeder Transition bezüglich seiner Aktivierung. Für $e_i \in e$ gilt $e_i \in \{0, 1\}$. Ist $e_i = 0$, so ist $T_i \in T$ deaktiviert, beziehungsweise bei $e_i = 1$ aktiviert.
Zu Beginn der Simulation, wird der initiale Modellzustand durch $\Gamma_0 = (m_0, x_0, c_0, d_0, g_0, e_0)$ angegeben, wobei $c_0 = g_0 = 0$.

2.3.2 Konfliktlösung

Während der Simulation kann immer nur eine Schaltung gleichzeitig abgearbeitet werden. Wenn nun aber mehrere Schaltungen zum selben Zeitpunkt stattfinden sollen, kommen die in 2.3 genannten Transitionsgewichtungen und Prioritäten zum Einsatz. Unter den Transitionen mit gleichem Schaltzeitpunkt wird die Transition mit der höchsten Priorität bevorzugt. Sollte es unter den Kandidaten mehrere Transitionen mit gleicher Priorität geben, so werden gemäß der verschiedenen Varianten von Transitionen sogenannte *Konfliktmengen* gebildet.
Sei nun $C^I(\Gamma)$ die Konfliktmenge der direkten Transitionen T^I gleicher Priorität in einem Modellzustand Γ. Dann ist die relative Schaltwahrscheinlichkeit einer direkten Transition

$T_i^I \in C^I(\Gamma)$ nach [3] (S.28) gegeben durch:

$$P(T_i^I) = \frac{\phi_w^T(T_i^I)}{\sum\limits_{\forall T_i^I \in C^I(\gamma)} \phi_w^T(T_i^I)}$$

Diese Formel kann gleichermaßen auch für deterministische und stochastische Transitionen beziehungsweise für deren entsprechenden Konfliktmengen $C^D(\Gamma, t)$ und $C^G(\Gamma, t)$ verwendet werden, wobei t den jeweiligen Schaltzeitpunkt angibt.

2.3.3 Rate adaption

Wenn für einen stetigen Zustand $P_i^c \in P^c$ eine obere Grenze $\phi_b^P(P_i^c) \neq \infty$ angegeben ist, kann es im Verlauf der Simulation zu einem *Überfluss* kommen, wenn dieser Grenzwert durch einen vorhandenen Zufluss erreicht wird und die Zuflussrate unverändert bleibt. Ebenso kann es auch einen *Unterfluss* geben, falls durch einen vorhandenen Abfluss die Untergrenze null erreicht wird. Um dem entgegenzuwirken, soll eine Anpassung des Zuflusses an den Abfluss, beziehungsweise des Abflusses an den Zufluss vorgenommen werden, wodurch der Flüssigkeitsstand des betroffenen Zustands unverändert bleibt. Dieses Verfahren wird mittels des in [3] (S.28-30) vorgestellten *Rate adaption*-Algorithmus durchgeführt (Wichtig: Die folgende Verfahrensbeschreibung ist stark vereinfacht ausgedrückt, da der detaillierte Ablauf der Prozedur für den eigentlichen Inhalt dieser Arbeit keine maßgeblich Rolle einnimmt, aber zugunsten der Vollständigkeit dennoch mit einbezogen wurde). Der Algorithmus iteriert über alle stetigen Zustände, welche eine der Grenzen erreicht haben, und passt die Flussraten entsprechend an. Dabei wird zunächst die *verfügbare Gesamtflüssigkeit* eines Zustandes $P_i^c \in P^c$ berechnet, das heißt die Summe der Kantengewichte aller Kanten, über die P_i^c mit aktivierten (stetigen) Transitionen verbunden ist.
Im nächsten Schritt werden zuerst die Transitionen betrachtet, bei denen die Kanten, mit denen sie mit P_i^c verbunden sind, höchste Priorität haben. Dann wird die *benötigte Flüssigkeit* des Zustandes P_i^c berechnet. Dieser Wert setzt sich zusammen aus der Summe der Flussraten ϕ_f^F der betrachteten Transitionen, jeweils multipliziert mit dem Kantengewicht der korrespondierenden Kante. Dabei werden allerdings nicht alle Transitionen, beziehungsweise Kanten in Betracht gezogen. Für den Fall, dass P_i^c Unterfluss droht, werden nur von dem Zustand abfließende Kanten betrachtet und für Überfluss nur zufließende. Sollte P_i^c zum Beispiel seine Untergrenze erreicht haben (es droht also Unterfluss) und die Transition $T_i^F \in T^F$ ist über eine zufließende Kante höchster Priorität mit P_i^c verbunden, so wird das Gewicht dieser Kante nicht mit in die Rechnung einbezogen.
Anhand dieser beiden Werte erfolgt nun die Anpassung der Flussraten ϕ_f^F der verbundenen Transitionen. Wenn der Wert der verfügbaren Gesamtflüssigkeit größer als die benötigte Flüssigkeit ist, so bleiben die die Flussraten der in diesem Schritt betrachteten Transitionen erhalten. Sobald aber die Menge der verfügbaren Gesamtflüssigkeit nicht mehr für alle Transitionen der aktuell betrachteten Priorität ausreicht, so wird der Rest der verfügbaren Flüssigkeit gemäß der *share*-Funktionswerte ϕ_s^A verteilt.

Dieses Verfahren wird solange durchgeführt, bis der Drift aller stetigen Zustände, die ihre Obergrenze erreicht haben, kleiner oder gleich null ist, beziehungsweise größer oder gleich null, für den Fall, dass die Untergrenze erreicht wurde.

2.3.4 Ereignisse

Die Entwicklung des Modells (beziehungsweise die Veränderungen des zugehörigen Petri-Netzes) im Laufe der Simulation kann als eine zeitliche Abfolge sogenannter *Ereignisse* betrachtet werden. Dabei kann ein Ereignis das Schalten einer Transition, das Erreichen einer der Grenzen eines stetigen Zustands oder die Aktivierung einer Transition über eine *Test-* oder *Hemm*-Kante sein, welche den aktuellen Modellzustand $\Gamma_i = (m_i, x_i, c_i, d_i, g_i, e_i)$ in den veränderten Zustand $\Gamma_{i+1} = (m_{i+1}, x_{i+1}, c_{i+1}, d_{i+1}, g_{i+1}, e_{i+1})$ versetzt. Zwischen zwei solcher Ereignisse ändert sich nur der stetige Teil des Modells durch die (eventuell) vorliegenden Drifts, der diskrete Teil des Modells bleibt unverändert.
Ein Ereignis ist ein Tupel $\Upsilon = (\Delta\tau, e)$ ([3], S30-31), wobei $\Delta\tau$ die Zeit, ausgehend von dem Modellzustand Γ, bis zum Ereigniseintritt darstellt und $e \in P \cup T \cup A$ das Modellelement nennt, welches das Ereignis verursacht. Sei nun $E(\Gamma) = E^T(\Gamma) \cup E^P(\Gamma) \cup E^A(\Gamma)$ die Menge aller Ereignisse, die von Γ aus erreicht werden können. Genauer ist $E^T(\Gamma) = E^{T^D}(\Gamma) \cup E^{T^G}(\Gamma) \cup E^{T^I}(\Gamma)$ die Menge der Ereignisse, bei denen eine der vorhandenen Transitionen schaltet, $E^P(\Gamma)$ die Menge der Ereignisse, bei denen ein stetiger Zustand eine Grenze erreicht und $E^A(\Gamma) = E^{A^G}(\Gamma) \cup E^{A^D}(\Gamma) \cup E^{A^I}(\Gamma) \cup E^{A^F}(\Gamma)$ die Menge der Ereignisse, bei denen eine Transition (repräsentiert durch eine anhängende Kante) durch eine Test- oder Hemm-Kante aktiviert wird. Anhand dieser Mengen kann die Modellentwicklung genauer analysiert werden, indem für Γ das Ereignis $\Upsilon \in E(\Gamma)$ gesucht wird, welches in der chronologischen Reihenfolge als nächstes eintritt. Sei $E^{min}(\Gamma) = \left\{\Upsilon = (\Delta\tau_i, e_i) \in E(\Gamma) | \nexists \Upsilon = (\Delta\tau_j, e_j) \in E(\Gamma) : \Delta\tau_j < \Delta\tau_i\right\}$ die Menge der Ereignisse mit minimaler verbleibender Zeit. Es gilt nun insgesamt drei Fälle zu unterscheiden. Wenn $|E(\Gamma)| = \emptyset$, befindet sich das Modell im sogenannten *absorbing state* ([3], S.32), in dem sich die Markierungen nicht weiter verändern werden. Im Fall $|E^{min}(\Gamma)| = 1$, ist die Suche nach dem nächsten Ereignis mit minimaler, verbleibender Zeit trivial. Sollte allerdings $|E^{min}(\Gamma)| > 1$ gelten, so wird das nächste Ereignis gemäß der in 2.1 vorgegebenen Ordnung gewählt.

Tabelle 2.1: Ordnung und Bearbeitungsschritte der möglichen Ereignisse. Quelle: [8], S.11

Randordnung	Art des Ereignisses	Auswirkungen
I	Schaltung einer direkten Transition: $\Upsilon_k^{T^I} = (\Delta\tau, T_K^I) \in E^{T^I}(\Gamma)$	1. Auflösen direkter Konflikte 2. Schaltung einer direkten Transition T_k^I 3. Ausführung von *rate adaption*
II	Wächterkante einer direkten Transition ist aktiviert: $\Upsilon_k^{A^I} = (\Delta\tau, A_K^I) \in E^{A^I}(\Gamma)$	1. Anpassung der Flüssigkeitsstände von $\Delta\tau$
III	Schaltung einer deterministichen Transitionen: $\Upsilon_k^{T^D} = (\Delta\tau, T_K^D) \in E^{T^D}(\Gamma)$	1. Anpassung der Flüssigkeitsstände von $\Delta\tau$ 2. Auflösen deterministischer Konflikte 3. Schalten einer deterministischen Transition T_k^D 4. Ausführung von *rate adaption*
IV	Stetiger Zustand erreicht Grenze: $\Upsilon_k^{P^c} = (\Delta\tau, P_k^c) \in E^{P^c}(\Gamma)$ Wächterkante stetiger Transition aktiviert: $\Upsilon_k^{A^C} = (\Delta\tau, A_k^C) \in E^{A^C}(\Gamma)$	1. Anpassung der Flüssigkeitsstände von $\Delta\tau$ 2. Ausführung von *rate adaption*
V	Wächterkante deterministischer Transition aktiviert: $\Upsilon_k^{A^D} = (\Delta\tau, A_k^D) \in E^{A^D}(\Gamma)$ Wächterkante stochastischer Transition aktiviert: $\Upsilon_k^{A^G} = (\Delta\tau, A_k^G) \in E^{A^G}(\Gamma)$	1. Anpassung der Flüssigkeitsstände von $\Delta\tau$
VI	Schaltung einer stochastischer Transition: $\Upsilon_k^{T^G} = (\Delta\tau, T_k^G) \in E^{T^G}(\Gamma)$	1. Anpassung der Flüssigkeitsstände von $\Delta\tau$ 2. Auflösen stochastischer Konflikte 3. Schalten einer stochastischen Transition T_k^G 4. Ausführung von *rate adaption*

3 Hauptfunktionen des Tools- Grundlagen

In diesem Kapitel werden die Grundzüge der wesentlichen Funktionen des Tools dargelegt. Dabei handelt es sich einerseits um die ereignisorientierte Simulation hybrider Petri-Netze und zum anderen um die auf diesen Simulationen basierende Durchführung von Model Checks.
In diesem Kontext wird zunächst in 3.1 das verwendete Simulationsverfahren erläutert, welches auf [4] basiert. Im Anschluss folgt in 3.2 eine Übersicht über die Thematik des Model Checkings (beruhend auf [8]), mit Fokus auf die Art und Weise, wie diese mit dem Simulationstool durchgeführt werden können.

3.1 Simulation diskreter Ereignissysteme

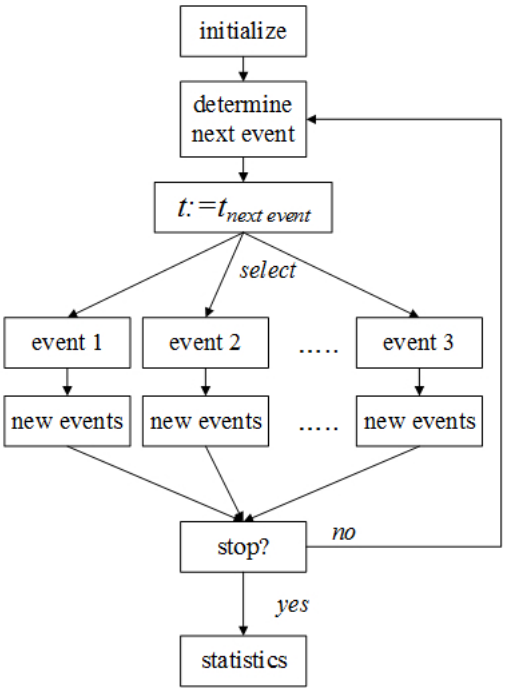
Von einem *diskreten Ereignissystem* spricht man, wenn es sich um ein System handelt, welches sich im Laufe der Ausführung systeminterner Funktionen durch das Eintreten verschiedener Ereignisse verändert([4], S.412 - 413). Petri-Netze gehören zu dieser Art von System, wobei die Art von Ereignissen, welche innerhalb der Modellierung auftreten können, in 2.3.4 bereits genauer dargestellt wurden.
Im Hinblick auf die Simulation solcher diskreter Ereignissysteme unterscheidet man zwei verschiedene Herangehensweisen [4]. Die *zeitbasierte Simulationsvariante* sieht es vor, dass innerhalb einer Kontrollschleife, ausgehend von einem Zeitpunkt t, geprüft wird, ob in dem zeitlichen Intervall $[t, t + \Delta]$ ein oder mehrere Ereignisse stattgefunden haben. Wenn dies der Fall ist, werden diese Ereignisse entsprechend abgearbeitet und die Kontrollschleife wiederholt sich mit $t = t+\Delta$ bis ein maximales t erreicht wird. Die Nachteile dieser Methodik sind allerdings, dass zum einen die einzelnen Ereignisse ungeordnet und eigenständig betrachtet werden. Wenn Δ zu groß gewählt wird, läuft man Gefahr, dass zwei gegenseitig abhängige Ereignisse als unabhängig aufgefasst werden. Daher muss Δ einen sehr kleinen Wert annehmen (das heißt es werden nur sehr kleine Zeitintervalle betrachtet), was wiederum bedeutet, dass es voraussichtlich viele Kontrollschleifen geben wird, in denen keine Ereignisse stattfinden.
All dies macht die Art der zeitbasierten Simulation recht ineffizient. Viel häufiger wird hingegen die zweite Simulationsvariante genutzt, die *ereignisorientierte Simulation*, welche im Folgen näher erklärt wird.

3.1.1 Ereignisorientierte Simulation

Bei der ereignisorientierten Simulation werden anstatt bestimmter zeitlicher Intervalle die einzelnen Ereignisse in chronologischer Reihenfolge betrachtet. Da durch ein Ereignis das Modell verändert wird, sorgt dessen Eintreten dafür, dass sich die Menge der

Ereignisse, welche zukünftig passieren könnten, ebenfalls verändert. Das Verfahren sieht nun vor eine Liste zu berechnen und diese stets so anzupassen, dass das erste Element immer das als nächstes eintretende Ereignis ist. In einer Kontrollschleife wird das Modell gemäß dieses Ereignisses angepasst und die darauf folgenden Ereignisse in der Liste abgelegt und ihre Eintrittszeiten kalkuliert. Das bearbeitete Ereignis wird anschließend aus der Liste entfernt. Diese Kontrollschleife wird bis zum gewünschten Simulationsende ausgeführt. 3.1 skizziert den Ablauf des Simulationsvorgangs in grafischer Darstellung.
Der Vorteil dieser Vorgehensweise ist die optimale Länge der Zeitintervalle zwischen den Ereignissen, da die Ereignisse stets einzeln und in chronologischer Reihenfolge betrachtet werden, wodurch es keine Zeitintervalle gibt, in denen kein Ereignis vorkommt. Dadurch entstehen zudem keine Konflikte in Bezug auf Abhängigkeiten.

Abbildung 3.1: Grafische Darstellung der ereignisorientierten Simulation. Quelle: [4], (S.414)



3.2 Statistisches Model Checking

Unter Model Checking versteht man im Allgemeinen das Verfahren der automatisierten Analyse von reaktiven Systemen. Die Arbeitsweise des Model Checkings besteht aus der Modellierung eines Systems, der Spezifizierung einer oder mehrerer Eigenschaft, welche es innerhalb des Systems zu prüfen gilt, und schließlich dessen Verifizierung [6].
Für den Fall, dass die betrachtete Modellierung jedoch stochastische Elemente, in diesem Fall stochastische Transitionen, enthält, werden für das Modell Checking nach [5]

zwei Herangehensweisen unterschieden. Zum einen das numerische und zum anderen das statistische Verfahren. Das numerische Verfahren kann zwar generell eine hohe Genauigkeit ausweisen, ist allerdings zum Teil äußerst rechenintensiv. Das zweite Verfahren ist das *Statistische Model Checking*. Die Technik des statistischen Model Checkings sieht vor, dass zunächst anhand des Modells und der zu prüfenden Eigenschaft mehrere Simulationen durchgeführt werden, wobei die Resultate dieser Durchläufe dokumentiert werden. Im Anschluss werden statistische Methoden genutzt, um die so gesammelten Daten auszuwerten und eine Aussage bezüglich der Erfüllung der Eigenschaft innerhalb des Modells aufzustellen.
Neben der Simulation hybrider Petri-Netze gehört die Durchführung solcher Model Checks zu den Hauptfunktionen des Tools. So können zu eingelesenen Modellierungen hybrider Petri-Netze Eigenschaften spezifiziert und auf ihre Korrektheit geprüft werden. Auch hier werden zunächst einige Simulationsdurchläufen ausgeführt. Die dadurch generierten Daten geben approximativ Auskunft darüber, mit welcher Wahrscheinlichkeit die eingegebene Eigenschaft erfüllt wird. Über diese Daten kann dann mittels besagter statistischer Methoden wie das Berechnen von Konfidenzintervallen für die näherungsweise Bestimmung der tatsächlichen Wahrscheinlichkeit und der Durchführung von Hypothesentests ein Resultat erzielt werden.

3.2.1 STL-Syntax

Der *Stochastic Time Logic* Formalismus, kurz *STL*, ermöglicht es modellbezogene Eigenschaften hybrider Petri-Netze für die Durchführung der Model Checks zu formulieren. Dieser Formalismus wird im Folgenden kurz beschrieben.
Die *STL*-Syntax ist nach [8] (S.13-14) gegeben durch folgende Definition:

Definition 1. Sei (P, T, A, m_0, ϕ) die Modellierung eines hybriden Petri-Netzes.
Eine *STL* Formulierung für ein solches Modell ist definiert als:

$$\varphi ::= P_{\bowtie p}(\Psi),$$

$$\text{mit } \bowtie \in \{<, \leq, >, \geq\},\ p \in [0,1] \text{ und } \Psi ::= tt \mid AP \mid \neg\Psi \mid \Psi \wedge \Psi \mid \Psi U^{[t_1,t_2]\Psi}$$

$$\text{wobei } t_1, t_2 \in \mathbb{R}_0^+ \text{ mit } t_1 \leq t_2 \text{ und}$$

$$AP ::= x_P \sim a \mid m_P \sim b \mid c_T \sim c \mid d_P \sim d \mid g_T \sim e \mid e_T,$$

$$\text{mit } \sim \in \{=, <, \leq, >, \geq\},\ a, c, e \in \mathbb{R}_0^+ \text{ und } b \in \mathbb{N} \text{ und } d \in \mathbb{R}$$

Bei AP handelt es sich um sogenannte *Atomic properties*, welche beschrieben sind durch:

- $x_p \sim a$: Vergleich des Flüssigkeitsstandes x_P eines stetigen Zustandes P mit einem Wert a.
- $m_p \sim b$: Vergleich einer Zustandsmarkierung m_P eines diskreten Zustands P mit einem Wert b.
- $c_T \sim c$: Vergleich des Clock Wertes c_T einer deterministischen Transition T mit einem Wert c.

- $d_P \sim d$: Vergleich des Drifts d_P eines stetigen Zustands P mit einem Wert d.
- $g_T \sim e$: Vergleich der Aktivierungszeit g_T einer stochastischen Transition T mit einem Wert e.
- e_T: Überprüfung des Aktivierungsstandes einer Transition T.

Die *STL* Syntax umfasst den Fall *logisches Oder* zwar nicht direkt, allerdings kann dieser über *De Morgan's law* hergeleitet werden: $(\Psi_1 \vee \Psi_2) := \neg(\neg\Psi_1 \wedge \neg\Psi_1)$.

Die Semantik des *STL*-Formalismus umfasst die Definitionen der Bedingungen, welche erfüllt sein müssen, damit eine in *STL* formulierte Modelleigenschaft in Hinsicht auf die zugehörige Modellierung als erfüllt gilt. Anhand der Semantik kann das Simulationstool daher besagte Modelleigenschaften verifizieren und somit die für das Model Checking benötigten Daten sammeln. Im Rahmen dieser Arbeit ist ein genaueres Verständnis dieser Definitionen allerdings nicht von besonderer Bedeutung (die Semantik ist einsehbar in [8], S.14-16). Stattdessen folgt nun eine Übersicht darüber, wie eine *STL*-Formel im Tool eingegeben werden kann.

3.2.2 Eingabe von *STL*-Formulierungen

Dieser Abschnitt basiert auf [8] (S.33-34).
Die Formulierungsvorschrift einer *STL* Formulierung für Hypothesentests ist durch folgenden Term gegeben:

$$\mathbf{T: P \sim x\ (\Phi)}$$

Dabei ist **T** der Zeitpunkt, zu dem die folgende Formulierung geprüft werden soll. Dieser Zeitpunkt muss als Dezimalwert angegeben werden.
$\sim$ wird durch einen Vergleichsoperator ersetzt, das heißt $\sim \in \{<, \leq, >, \geq\}$.
x ist eine Dezimalzahl, welche für den Wert p_0 steht, demnach gilt $\mathbf{x} \in [0, 1]$
Φ wird durch einen der folgenden Formeln ersetzt:

- True: **'tt'**
- Negation: '$!\Psi_1'$
- Konjunktion: **'AND**(Ψ_1, Ψ_2)**'**
- Disjunktion: **'OR**(Ψ_1, Ψ_2)**'**
- *Until formula*: **'U**$[t_1, t_2]$ (Ψ_1, Ψ_2)**'**
- *Simple atomic property*

Dabei sind Ψ_1 und Ψ_2 weitere Formeln.

Die Menge der *Atomic properties* ist definiert mit:

- **‘fluidlevel(‘id’) ∼ x’**
  Vergleich des Flüssigkeitsstand eines stetigen Zustands mit einem Wert **x**.
- **‘drift(‘id’) ∼ x’**
  Vergleich des Drifts eines stetigen Zustands mit einem Wert **x**.
- **‘uboundary(‘id’)’**
  Überprüfung, ob ein stetiger Zustand seine obere Grenze erreicht hat.
- **‘lboundary(‘id’)’**
  Überprüfung, ob ein stetiger Zustand seine untere Grenze erreicht hat.
- **‘tokens(‘id’) ∼ y’**
  Vergleich der Anzahl von Token eines diskreten Zustand mit einem Wert **x**.
- **‘enabled(‘id’)’**
  Überprüfung des Aktivierungsstatus einer beliebigen Transition.
- **‘clock(‘id’) ∼ x’**
  Vergleich des Clock Werts einer deterministischen Transition mit einem Wert **x**.
- **‘enablingtime(‘id’) ∼ y’**
  Vergleich der Aktivierungszeit einer stochastischen Transition mit einem Wert **x**.
- **‘condition(‘id’)’**
  Überprüfung, ob die Bedingung einer Test- oder Hemm-Kante erfüllt ist.

Wobei **id** die ID einer Transition (zum Beispiel **‘it1’**), eines Zustands (zum Beispiel **‘pd1’**) oder einer Test-, beziehungsweise Hemm-Kante ist (zum Beispiel **‘ga1’**), **x** ist ein Dezimalwert, **y** ein Integer Wert und es gilt $\sim \in \{<, \leq, >, \geq\}$.

Beispiele.

8.0:P>0.5(tokens(‘pd1’)>0)
Testet, ob die Wahrscheinlichkeit, dass bei **T = 8.0** der diskrete Zustand mit **id = ‘pd1’** mindestens ein Token hält, größer als **0.5** ist.

10.0:P>0.5(AND(condition(‘ga1’),drift(‘pc2’)<100.0))
Testet, ob die Wahrscheinlichkeit, dass bei **T = 10.0** sowohl die Bedingung der Kante mit **id = ‘ga1’** erfüllt, sowie der Drift des stetigen Zustands mit **id = ‘pc2’** kleiner als **100.0** ist.

4 Hypothesentest

In diesem Kapitel werden die für die Implementierung ausgewählten Algorithmen präsentiert, beziehungsweise deren mathematischen Grundlagen aufbereitet.
Zunächst wird in 4.1 eine allgemeine Darstellung des Aufbaus und der Funktionsweise von Hypothesentests (nach [10]) angegeben. Dann werden die in [10] vorgestellten Klassifikationen von Testalgorithmen erläutert in 4.2. Zudem werden die zur Auswahl stehenden Algorithmen aufgelistet in 4.3 und die getroffene Wahl kurz begründet. Anschließend werden diese Hypothesentestalgorithmen in 4.4 detailliert dargestellt.

4.1 Hypothesentests Grundlagen

Bei einem Hypothesentest gilt es eine Vermutung bezüglich einer unbekannten, reellen Wahrscheinlichkeit $p \in [0, 1]$ auf Korrektheit zu überprüfen. Diese Wahrscheinlichkeit bezieht sich dabei auf eine Aussage ϕ über einen zufallsbasierten Vorgang. Im Rahmen dieser Arbeit bezeichnet ϕ eine gemäß 3.2.1 formulierte Eigenschaft. Die Vermutung wird so modelliert, dass ein reeller Wert $p_0 \in [0, 1]$ für die Wahrscheinlichkeit angegeben und mit einem Vergleichsoperator in Relation zu p gesetzt wird. Nun gilt es, zwei gegensätzliche Hypothesen bezüglich dieser Angaben aufzustellen und zu prüfen, welche davon zutrifft. Dazu werden für die Hypothesen Akzeptanz- und Ablehnungsbereiche konstruiert, sowie eine Teststatistik anhand von $N \in \mathbb{N}$ Stichproben aufgestellt. Eine Teststatistik ist eine Funktion in Abhängigkeit der Stichprobengröße, welche Auskunft darüber geben soll, welche Hypothese zu akzeptieren ist. Die in dieser Arbeit verwendete Teststatistik wird im Folgenden noch näher erläutert, kurz umschrieben handelt es sich dabei um einen Wert, welcher mit jeder gezogenen Stichprobe steigt oder fällt. Je nachdem, in welchen Bereichen die Teststatistik landet, kann eine der Hypothesen akzeptiert (Teststatistik liegt in deren Akzeptanzbereich) und die andere verworfen werden (Teststatistik liegt in deren Ablehnungsbereich). In [10] (S.379) werden die Hypothesen als H_{+1} und H_{-1} bezeichnet mit den Definitionen:

$$H_{+1} : p > p_0$$
$$H_{-1} : p < p_0.$$

Des Weiteren existiert noch eine dritte Hypothese $H_0 : p = p_0$ genannt Nullhypothese. Diese kann jedoch mit statistischen Mitteln nicht bewiesen werden, da es selbst für sehr große N nicht ausgeschlossen werden kann, dass ein geringer Wert ϵ existiert, sodass $p = p_0 + \epsilon$ gilt ([10]). Zur Beobachtung der Stichproben wird in [10] die sogenannte

verschobene Teststatistik Z_N verwendet, welche folgendermaßen definiert ist ([10], S.379):

$$Z_N \mathrel{\widehat{=}} S_N - Np_0$$
$$\text{mit}^* \, S_N(X) = \sum_{i=0}^{N} X_i.$$

X_i ist dabei das Ergebnis einer einzelnen Stichprobe und $S_N(X)$ die Anzahl aller Stichproben mit positivem Ergebnis gemäß der Eigenschaft ϕ im Rahmen der zugrundeliegenden Modellierung. Demnach ist X_i definiert mit ([10], S.379):

$$X_i \mathrel{\widehat{=}} \mathbf{1}_\phi(\omega_i) = \begin{cases} 1, & \text{wenn } \phi \text{ in } \omega_i \text{ erfüllt ist} \\ 0, & \text{sonst,} \end{cases}$$

dabei ist ω_i der Simulationsdurchlauf der i-ten Stichprobe.

Aus dieser Definition folgt, dass für H_{+1} der Wert von Z_N (weit) größer 0 und für H_{-1} (weit) kleiner 0 ausfällt. Wenn zum Beispiel $S_N < Np_0$ gilt, so verweist dies potentiell auf $p < p_0$ und die H_{-1}-Hypothese, andersherum ist $S_N > Np_0$ ein Hinweis auf $p > p_0$ und die H_{+1}-Hypothese. Wie aus der Berechnung hervorgeht, ist die Größe von Z_N und damit die Wahrscheinlichkeit ein konkretes Ergebnis zu erhalten maßgeblich von $|p - p_0|$ abhängig. Je größer dieser Wert ist, desto schneller und deutlicher wächst Z_N in Richtung einer der beiden Akzeptanzbereiche.
Hier folgt nun ein Beispiel zum Ablauf eines so definierten Hypothesentests:

Beispiel. Sei ϕ eine zufallsbasierte Aussage, die mit einer Wahrscheinlichkeit p zutrifft und $p_0 = 0{,}4$. Sei weiterhin $N = 100$ und die Modellierung des Tests gegeben durch $P_{>p_0}(\phi)$. Die Annahmebereiche sind gegeben durch {9, ..., 100} für H_{+1} und {-9, ..., -100} für H_{-1} (diese Bereiche sind nicht berechnet, sondern zu Veranschaulichungszwecken frei gewählt). Wenn nun $Z_N = 10$ gilt, so wird die H_{+1}- Hypothese akzeptiert und die H_{-1}-Hypothese verworfen. Sollte Z_N hingegen in dem Bereich {-8, ..., 8} liegen, so ist dies ein Indiz für $p = p_0$, was allerdings, wie oben bereits erwähnt, nicht bewiesen werden kann. Der Test endet somit ergebnislos.

Zur grafischen Veranschaulichung des Testprozesses werden in [10] (S.379-380) vier Bereiche als Teilmengen von $\mathbb{R}^2$ auf einem Graphen mit der Ordinate Z_N und der Abszisse N definiert. Die Bereiche U und L stellen die Akzeptanzbereiche für H_{+1} und H_{-1} dar. Sobald Z_N einen ausreichend großen, beziehungsweise kleinen Wert erreicht hat und somit einen der Bereiche betritt, kann die entsprechende Hypothese akzeptiert werden. Sollte Z_N den dritten Bereich I erreichen ohne eine ausreichende Größe erreicht zu haben, so wird der Test ohne abschließendes Resultat beendet. Der vierte Bereich NC ist der Rest des Graphen. Befindet sich Z_N in NC, kann aufgrund dessen unzureichender Größe noch kein Ergebnis berechnet werden. Die ersten drei Bereiche werden als *kritisch* bezeichnet, da der Test endet, sollten sie erreicht werden. Analog dazu ist NC ein *nicht kritischer* Bereich.
Bei einem Hypothesentest gilt es zu beachten, dass kein Test eine absolute Garantie auf Korrektheit des Ergebnisses hat. Man spricht in diesem Kontext vom *Fehler erster Art*,

Fehler zweiter Art und dem *Konfidenzniveau*. Der Fehler erster Art ist die Wahrscheinlichkeit, mit der ein falsches Ergebnis aufgrund entsprechend ausfallender Teststatistik als korrekt bezeichnet wird. Der Fehler zweiter Art ist hingegen die Wahrscheinlichkeit, ein korrektes Ergebnis als falsch aufzufassen. Diese Wahrscheinlichkeiten stellen demnach Ergebnisunsicherheiten dar, weswegen zu hohe Werte fatale Folgen für die Qualität eines Tests bedeuten. Generell möchte man einen Hypothesentest also so konstruieren, dass beide Fehlerwahrscheinlichkeiten unter einem gewissen Schwellenwert gehalten werden. Für den Fehler erster Art ist dieser Schwellenwert mit α betitelt, für den Fehler zweiter Art mit β. Die in dieser Arbeit vorgestellten Tests unterliegen nach [10] ebenfalls bestimmten Bedingungen bezüglich α und β, welche bei deren Konstruktion stets zu beachten sind. Diese Bedingungen werden im Nachfolgendem näher erläutert.
Sei A_i mit $i \in \{-1, 1\}$ das Ereignis, dass eine der beiden Hypothesen akzeptiert wird ([10], S.380). Dann soll für α und β gelten

$$\mathbb{P}(A_{+1}|\neg H_{+1}) \leq \alpha_1 \tag{4.7}$$
$$\mathbb{P}(A_{-1}|\neg H_{-1}) \leq \alpha_2 \tag{4.8}$$

$$\mathbb{P}(\neg A_{+1}|H_{+1}) \leq \beta_1 \tag{4.9}$$
$$\mathbb{P}(\neg A_{-1}|H_{-1}) \leq \beta_2, \tag{5.0}$$

$$\text{dabei gilt}\, \alpha = \alpha_1 = \alpha_2 \,\text{und} \beta = \beta_1 = \beta_2.$$

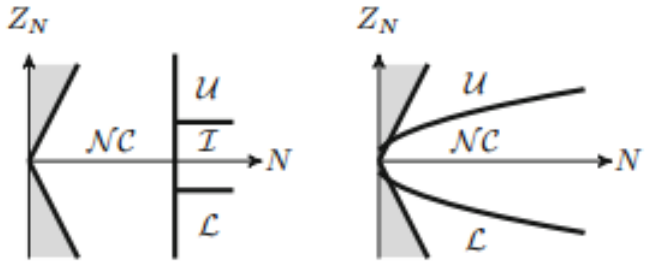
Nähere Erläuterung: die Gleichungen 4.7 und 4.8 besagen, dass die Wahrscheinlichkeit ein falsches Ergebnis zu erzielen durch α begrenzt sein soll. Die Gleichungen 4.9 und 5.0 besagen, dass die Wahrscheinlichkeit, ein falsches oder gar kein Ergebnis zu erzielen durch β begrenzt sein soll.
Zuletzt gibt das $1 - \alpha$ eines Hypothesentests an, mit welcher Wahrscheinlichkeit das Ergebnis des Tests korrekt ist.

4.2 Klassifikation von Hypothesentests

Generell unterscheidet man zwischen sequentiellen Algorithmen und denen mit fester Stichprobengröße. Sequentielle Tests prüfen nach jeder gezogenen Stichprobe, ob mit den vorhandenen Werten ein Ergebnis vorliegt. Ist dies nicht der Fall, so wird die Stichprobengröße so lange erweitert, bis ein Ergebnis feststeht oder der Testdurchlauf abgebrochen wird. Dieser potentielle Abbruch hängt allerdings nicht von der Definition des Tests ab, sondern unterliegt in der Regel externen Kriterien, wie zum Beispiel die Vorgabe einer maximalen Stichprobengröße N, die nicht überschritten werden darf.
Tests mit fester Stichprobengröße prüfen einmalig mit den Werten, die aus der zuvor berechneten Stichprobenanzahl hervorgehen. Sollten die Werte nicht für ein konkretes Resultat ausreichen, so endet der Test ergebnislos. In Bezug zu den oben genannten vier grafischen Bereichen U, L, I und NC gilt es zu beachten, dass sequentielle Tests nicht

über den Bereich I verfügen, da sie ja nur im Falle des "manuellen"Abbruchs kein Ergebnis erzielen können und dass die kritischen Bereiche U, L und I für Tests mit vorgebender Stichprobengröße durch eben diese Größe auf horizontaler Ebene begrenzt sind, da ja nur über diesen Wert allein ein Ergebnis berechnet werden soll. Abbildung 4.1 zeigt die die beschriebenen Bereiche

Abbildung 4.1: Abbildung der vier Bereiche *U, L, I* und *NC*. Links für Tests mit fester Stichprobengröße, rechts für sequentielle Tests. Quelle: [10], S.380



In [10] werden drei Kriterien zur Bewertung von Hypothesentestalgorithmen angegeben. Das erste Kriterium ist die *Korrektheit* des Tests. Die Wahrscheinlichkeit des Tests ein falsches Ergebnis zu berechnen, sollte durch α begrenzt sein (siehe 4.7, 4.8), beziehungsweise sollte die Wahrscheinlichkeit, mit der der Test korrekt liegt, dem Konfidenzniveau entsprechen. Als zweites Kriterium wird die *Mächtigkeit* des Tests betrachtet. Ein Test sollte möglichst so konstruiert sein, dass die Wahrscheinlichkeit, am Ende einer Berechnungsphase ein Resultat erzielen zu können, angemessen hoch ist. Das dritte Kriterium befasst sich mit der *Effizienz* eines Tests, das heißt die durchschnittliche Anzahl der Stichproben, die gebraucht werden, bis ein Ergebnis angegeben werden kann.
Bei dem Entwurf von Hypothesentestalgorithmen kommt es zwischen diesen Kriterien zu klassischen Trade-offs. Ein Test mit hoher Korrektheit und Mächtigkeit läuft Gefahr viele Stichproben für ein Ergebnis zu benötigen, wohingegen ein Test mit hoher Korrektheit und Effizienz mit erhöhter Wahrscheinlichkeit keine finale Aussage treffen kann und so weiter. Unabhängig von seiner Beschaffenheit gilt, dass ein Test in mindestens einem der Kriterien negativ beeinflusst wird, wenn der Wert $|p - p_0|$ sehr klein ist, da es schwieriger und aufwendiger wird ein (korrektes) Ergebnis zu erzielen.

Gemessen an der Art dieser Beeinflussung werden in [10] (S.381) drei Arten von Tests klassifiziert. Die erste Klasse umfasst Tests, deren Wahrscheinlichkeit, ein falsches Ergebnis zu berechnen, α für kleine $|p - p_0|$ übersteigt. Dies liegt daran, dass die Teststatistik Z_N durch kleine $|p - p_0|$ Werte ebenfalls nur eine geringe Größe annimmt. Es ist nun statistisch zu erwarten, dass Z_N sich dennoch in Richtung einer der Grenzen entwickeln wird. Aufgrund der geringen Größe von Z_N und der damit verbundenen geringen Tendenz zu einer der Grenzen, ist die Wahrscheinlichkeit relativ groß, beziehungsweise größer als für große $|p - p_0|$, dass Z_N in Richtung der falschen Grenze, beziehungsweise

in Richtung des falschen Annahmebereichs wächst.
Tests der zweiten Klasse laufen Gefahr, dass die Wahrscheinlichkeit, zu keinem Schlussergebnis zu gelangen β für kleine $|p-p_0|$ übersteigt. Dies ist wiederum dadurch begründet, dass Tests dieser Klasse (in der Regel) mit einer festen Stichprobengröße arbeiten (Anmerkung: dies wird vom Autor von [10] nicht explizit erwähnt, die Abbildung 4.1 lässt diesen Schluss jedoch eindeutig zu). Für Tests dieser Klasse gibt es also einen bestimmten Punkt, an dem der Testvorgang abgebrochen wird. Wenn Z_N nun aufgrund eines relativ kleinen $|p - p_0|$ Wertes eine unzureichende Größe angenommen hat, kann nach Erreichen des Abbruchpunktes kein Ergebnis präsentiert werden.
Tests der dritten Klasse benötigen für kleine $|p - p_0|$ potentiell eine große Anzahl an Stichproben, um eine Lösung berechnen zu können, allerdings akzeptieren diese Tests stets die richtige Hypothese (im Rahmen eines eines entsprechenden Konfidenzniveaus).
Für jede dieser Klassen beziehungsweise den zugehörigen Testalgorithmen existiert ein spezieller Eingabeparameter, welcher gerade im Hinblick auf die Implementierung der Tests eine größere Rolle spielt (näheres dazu wird in Kapitel 5 erläutert). Für Tests der ersten und zweiten Klasse symbolisieren diese Parameter ein Niveau bezüglich $|p-p_0|$ ab dem der Nutzer indifferent gegenüber der Korrektheit (für die erste Klasse) beziehungsweise der Mächtigkeit (für die zweite Klasse) der Testalgorithmen ist. Das heißt wenn sich $|p-p_0|$ unterhalb dieses Parameters befinden sollte, ist es nicht mehr von Interesse, ob er Test korrekt ist, beziehungsweise mit einem Ergebnis endet. Der Parameter der ersten Klasse ist mit δ, der Parameter der zweiten Klasse mit ζ angegeben. Bei der dritten Klasse von Tests wird durch den Parameter γ eine Vermutung bezüglich des konkreten Werts von $|p - p_0|$ ausgedrückt. An dieser Stelle sei schon im voraus angemerkt, dass in den Kapiteln 5 und 6 diese Parameter durch einen einheitlichen Wert *guess* ausgedrückt werden. Dies hat die Begründung das, obwohl die Bedeutungen unterschiedlich sind, sich alle Parameter auf den gleichen Wert nämlich $|p - p_0|$ beziehen und sich daher nur in der theoretischen Interpretation dieses Wertes unterscheiden.

4.3 Auswahl der Hypothesentests

Gemäß der Klassifikation aus 4.2, werden in [10] sieben Hypothesentestalgorithmen vorgestellt. Es folgt nun eine Auflistung dieser Algorithmen mit einer Begründung, welche Tests für die Implementierung in dieser Arbeit ausgewählt wurden. 4.1 bietet eine grafische Präsentation aller Algorithmen mit entsprechender Klassenzugehörigkeit.

Da die verschiedenen Klassen von Hypothesentests unterschiedliche Vor- und Nachteile bieten, wurden sie maßgeblich in die Auswahl der Testalgorithmen einbezogen. Ziel war es, möglichst einen Repräsentanten jeder Klasse in das Tool zu integrieren, um das Potential der unterschiedlichen Vorzüge auszuschöpfen.
Der *Sequentiell Propability Ratio Test* (kurz *SPRT*), gehört zur ersten Klasse von Tests, dieser wurde aber bereits im Tool implementiert und steht daher nicht zur Auswahl. Der zweite Vertreter dieser Klasse, der *Gauss Single Sampling Plan-* Algorithmus, kurz *Gauss- SSP*, schneidet performancetechnisch im Schnitt deutlich schlechter ab als der *SPRT* und ist deshalb ein tendenziell wenig interessanter Kandidat.

In der zweiten Klasse befinden sich drei verschiedene Algorithmen, der *Gauss-Confidence-Intervall-*, der *Chow Robbins-* und der *Chernoff-Confidence-Intervall* Test. Die Qualität des Ergebnisses ist beim *Cernoff-Confidence-Intervall* Test (kurz *Cernoff-CI*) zwar minimal besser als beim *Gauss-Confidence-Intervall-* (kurz *Gauss-CI*) und *Chow-Robbins* Test, allerdings auch unter dem Einsatz einer erheblich größeren Menge an Stichproben. Zudem wird [10] angegeben, dass der *Chow-Robbins* Test unter bestimmten Bedingungen (welche in 6.1.2 näher erläutert werden) schneller oder auch langsamer als der *Gauss-CI* Test ein Ergebnis erzielen kann. Da ein abschließender Vergleich der implementierten Hypothesentests ebenfalls Teil dieser Arbeit sein soll, wäre es ein interessant zu beobachten, inwieweit sich diese Gegebenheit in der Praxis widerspiegelt. Aus diesen Gründen wurden der *Gauss-CI* sowie der *Chow-Robbins* Test in die engere Auswahl einbezogen.
Die dritte Klasse umfasst die Tests *Azuma* und *Darling-Robbins*. Tests dieser Klasse kommen, wie in 4.2 beschrieben, stets zu dem richtigen Resultat. Die Entscheidung, welcher der beiden Tests dieser Klasse für die Implementierung interessanter ist, ist also nicht von der Qualität der Ergebnisse abhängig zu machen. Außerdem ähneln sich beide Tests sehr stark in ihrer Arbeitsweise, tatsächlich unterscheiden sie sich nur in der Parameterbestimmung. Die Wahl fiel deshalb letztlich auf den *Azuma* Test, da dieser in Bezug auf die Anzahl der benötigten Stichproben eine bessere Leistung erzielt.
Die Gesamtauswahl fällt also auf die Tests *Chow-Robbins*, *Gauss-CI* und *Azuma*, welche nun im nachfolgenden Kapitel detailliert präsentiert werden.

Tabelle 4.1: Übersicht über Hypothesentests und Klassifizierungen. Quelle: [10], S.381

	Klasse I	Klasse II	Klasse III
Risiko bei $p \approx p_0$	Korrektheit: falsches Ergebnis, d.h. Fehler erster Art (& Effizienz)	Mächtigkeit: kein Ergebnis, d.h. Fehler zweiter Art (& Effizienz)	Effizienz: Sehr große Stichprobengröße und lange Laufzeit
Parameter	Korrektheitsindifferenz Level δ	Mächtigkeitsindifferenz Level ζ	Vermutung γ
Tests mit fester Stichprobengröße	Gauss-SSP	Gauss-CI CHernoff-CI	
Sequentielle Tests	SPRT		Azuma Darling
Gemischte Tests		Chow-Robbins	

4.4 Grundlagen der gewählen Hypothesentests

Nun werden die mathematischen Grundlagen und Voraussetzungen im Detail erläutert, beginnend mit dem *Gauss-Confidence-Intervall*-Test. Anschließend folgen der *Chow-Robbins-* und *Azuma*-Algorithmus. Die präsentierten Grundlagen und Verfahrensweisen entstammen dabei [10], Kapitel 3.

4.4.1 Gauss-Confidence-Intervall

Der Gauss-Confidence-Intervall Algorithmus gehört zu den Tests mit fester, vor dem eigentlich Test berechneter Stichprobengröße. Ziel des Algorithmus ist, mittels Approximation ein Intervall zu konstruieren, welches mit einer Wahrscheinlichkeit von 1 - α

die tatsächlich vorliegende Wahrscheinlichkeit p enthält. Dadurch kann anhand der Lage von Z_N in Relation zu der unteren, beziehungsweise oberen Schranke des Intervalls festgestellt werden, welche Hypothese sicher akzeptiert werden kann. Liegt Z_N oberhalb der oberen Schranke, wird die H_{+1}-Hypothese akzeptiert, liegt Z_N unterhalb der unteren Schranke, die H_{-1}-Hypothese. Für den Fall, dass Z_N unterhalb der oberen und oberhalb der unteren Schranke liegt, endet der Test ergebnislos. Die zu bestimmenden Parameter des Algorithmus sind die Intervallgrenzen l und u und die Stichprobengröße N. Die Wahl von N bestimmt dabei maßgeblich über die Mächtigkeit des Tests. Wird N klein gewählt, so verbessert sich zwar die Effizienz, allerdings steigt das Risiko, kein Resultat erzielen zu können, wenn $|p - p_0|$ einen geringen Wert annimmt (siehe 4.2). Damit gehört der Gauss Confidence Intervall Algorithmus zur zweiten Klasse von Hypothesentests. Wird N hingegen sehr groß gewählt, so entsteht ein potentiell hoher Verlust an Effizienz zu Gunsten eines unverhältnismäßig kleinen Gewinns an Mächtigkeit, sollte $|p - p_0|$ nicht allzu gering ausfallen. Für diesen Test wird ein N gesucht, sodass die Vorgaben 4.9 und 5.0 erfüllt sind.
Sei $p - p_0$ gegeben durch $\zeta > 0$, so kann ein N berechnet werden, welches sowohl die gewünschten Eigenschaften nicht verletzt, als auch für eine bessere Effizienz minimal ausfällt. Für ein ausreichend großes N kann $\widehat{p} = S_N/N$ via Normalverteilung approximiert werden, mit

$$\begin{aligned}\mu &= p_0 + \zeta \\ \sigma^2 &\widehat{=} (p_0 + \zeta)(1 - p_0 - \zeta)/N.\end{aligned}$$

Sei nun weiterhin

$$\begin{aligned}\xi &= \phi^{-1}(1 - \alpha) \\ \sigma_{H_0}^2 &\widehat{=} (p_0)(1 - p_0)/N,\end{aligned}$$

dann ist die Wahrscheinlichkeit, dass auch nach N Stichproben H_{+1} verworfen wird gegeben durch

$$\begin{aligned}&\mathbb{P}(\widehat{p} \leq p_0 + \xi\,\sigma_{H_0}) = \mathbb{P}\Big(\frac{\widehat{p}_N - p_0 - \zeta}{\sigma} \leq \frac{\xi\sigma_{H_0}}{-}\zeta\sigma\Big) \\ &= \phi\left(\frac{\xi\sigma_{H_0} - \zeta}{\sigma}\right) \\ &= \phi\left(\frac{\xi\sqrt{p_0(1-p_0)} - \zeta\sqrt{N}}{\sqrt{(p_0+\zeta)(1-p_0-\zeta)}}\right).\end{aligned} \tag{4.1}$$

Umstellen nach β und Auflösen nach N ergibt folgende Gleichung

$$N_G = \left(\frac{\xi\sqrt{p_0(1-p_0)} - \phi^{-1}(\beta)\sqrt{(p_0+\zeta)(1-p_0-\zeta)}}{\zeta}\right)^2.$$

Führt man die gleiche Rechnung für $p = p_0 - \zeta$ durch, so erhält man einen zweiten, ebenfalls validen Wert für N. Für den Test wird der größere dieser beiden Werte gewählt.

Dadurch wird garantiert, dass für $|p - p_0| > \zeta$ 4.9 und 5.0 erfüllt sind.
Aus der Formel für N ist direkt ersichtlich, dass der Wert von p_0 ein wichtiger Faktor für die Größe darstellt. Durch das Einsetzen verschiedener Werte ergibt sich, dass N größer ausfällt, wenn p_0 nahe bei 0,5 liegt beziehungsweise kleiner, je näher p_0 an 0 oder 1 heranreicht.

4.4.2 Chow-Robbins

Der *Chow-Robbins Test* arbeitet sowohl sequentiell, als auch mit einer festen Stichprobengröße. Ziel ist es, ein Konfidenzintervall um den Wert $\widehat{p} = S_N/N$ zu bilden und anhand dessen Lage zu p_0 ein Ergebnis zu bestimmen. Dafür werden sequentiell Stichproben gezogen, bis das Intervall eine vorher berechnete Größe erreicht hat. Die Berechnung der Breite des Intervalls während der Simulation ist von den Ergebnissen der einzelnen Stichproben abhängig. Es kann also zu einer Varianz der benötigten Stichprobenanzahl kommen, welche jedoch durch die Vorgabe der maximalen Intervallgröße nach oben beschränkt ist. Nachdem diese Intervallgröße von 2ϵ erreicht ist, kann die H_{+1}- Hypothese akzeptiert werden, wenn $p_0 < \widehat{p}_N - \epsilon$, beziehungsweise H_{-1} für $p_0 > \widehat{p}_N + \epsilon$ gilt. Für $\widehat{p}_N - \epsilon < p_0 < \widehat{p}_N + \epsilon$ kann hingegen keine der Hypothesen akzeptiert werden und der Test endet ergebnislos. Damit gehört der *Chow-Robbins*-Algorithmus zur zweiten Klasse von Hypothesentests.
Bei einer Stichprobengröße N und bei einem Konfidenzniveau von $1 - 2\alpha$ ist die Breite des Intervalls gegeben durch

$$2\phi^{-1}(\alpha)\sqrt{Var(\widehat{p}_N)}$$
$$mit\, Var(\widehat{p}_N) = \widehat{p}_N(1 - \widehat{p}_N)/N.$$

Um das Intervall konstruieren zu können, gilt es einen Wert für dessen Halbbreite ϵ zu finden, sodass 4.9 und 5.0 erfüllt sind. Dabei kann $\widehat{p}_N$ durch eine Normalverteilung approximiert werden mit

$$\mu = p_0 + \zeta$$
$$\text{mit}\, \zeta = |p - p_0| \sigma = \epsilon/\phi^{-1}(1 - \alpha) = -\epsilon/\phi^{-1}(\alpha).$$

Die H_{+1}- Hypothese wird für $\widehat{p}_N \leq p_0 + \epsilon$ verworfen. Deshalb kann 4.1 benutzt werden, wobei $\xi\sigma_{H_0}$ durch ϵ substituiert wird. Umstellen nach β ergibt

$$\epsilon = \frac{\zeta}{1 + \phi^{-1}(\beta)/\phi^{-1}(\alpha)}.$$

Durch das Einsetzen von verschiedenen Werten für $\widehat{p}_N$ wird ersichtlich, dass die Intervallbreite für $\widehat{p}_N = 0,5$ maximal ist. Dadurch ist die tatsächlich vorliegende Wahrscheinlichkeit p ein bedeutender Faktor bezüglich der Effizienz des Tests. Ein Ergebnis kann schneller erzielt werden, wenn p und damit auch $\widehat{p}_N$ näher an 0 oder 1 heranreicht beziehungsweise, wenn p weit von 0,5 entfernt liegt.

4.4.3 Azuma

Der *Azuma*-Algorithmus gehört zu den sequentiellen und zur dritten Klasse von Hypothesentests, also jenen Tests, welche unter Einsatz einer potentiellen hohen Menge an Stichproben stets das korrekte Ergebnis bestimmen. Der Test arbeitet, ähnlich wie der *Gauss-CI* Algorithmus, mit einem Intervall und nutzt die relative Position der Teststatistik Z_N zu diesem Intervall, um ein Ergebnis zu erzielen. Der Unterschied besteht darin, dass der *Azuma*-Algorithmus die Intervallgrenzen nicht einmalig, sondern über zwei von der Stichprobengröße N abhängige Funktionen bestimmt. Die untere Intervallgrenze ist mit $l(N)$, die obere Grenze mit $u(N)$ betitelt. Es wird angenommen, dass Symmetrie vorliegt, das heißt es gilt $u(N) = -l(N)$. Damit ist es ausreichend, einen geeigneten Term für eine der beiden Funktionen zu finden. Im Folgenden wurde dafür $u(N)$ gewählt.
Für das Aufstellen der Funktion ist es wichtig, dass sie asymptotisch schneller wächst als$\sqrt{N}$, um Fehler erster Art für kleine $|p - p_0|$ zu vermeiden. Dies ist damit begründet, dass Z_N proportional zu$\sqrt{N}$ wächst. Wenn nun die H_0 Hypothese korrekt ist, das heißt $p = p_0$, dann ist die Wahrscheinlichkeit, dass eine der beiden Grenzen von Z_N überschritten wird, gleich 1 ([10], S.384). Wenn nun $Z_N > u(N)$ gilt, so wird die H_{+1}-Hypothese und für $Z_N < l(N)$ die H_{-1}-Hypothese akzeptiert.
Sei nun

$$u(N) = a(N + k)^b$$
$$mit\, a > 0,\, k > 0,\, b \in (\frac{1}{2}, 1).$$

Wählt man

$$\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = e^{-8(3b-2)a^2k^{2b-1}}, \tag{4.2}$$

so werden die Eigenschaften 4.7, 4.8, 4.9 und 5.0 erfüllt.
Es gilt nun geeignete Werte für die Parameter a, b und k zu finden. Die Parameter a und k beeinflussen dabei die Breite des nicht kritischen Bereichs NC, also der Bereich, für den gilt, dass, sollte sich die verschobene Teststatistik Z_N darin befinden, noch keinerlei Ergebnis vorliegt und weitere Stichproben gezogen werden müssen. Bei einem groß gewähltem a erhöht sich der Zuwachs an der Breite von NC im Verlauf des Tests, das heißt für steigendes N. Sollte der Wert für a also zu hoch ausfallen, braucht man umso mehr Stichproben, da die Bereiche U und L, für die ein Ergebnis gegeben werden kann, sehr klein werden und ein entsprechend großes Z_N benötigt wird. Bei einem groß gewählten Parameter k vergrößert sich die initiale Breite von NC, das heißt für kleine N- Werte. Es wird dadurch schwieriger zu Beginn des Tests eine Entscheidung zu treffen, was sich negativ auf die Effizienz auswirken kann, wenn es sich zum Beispiel um einen relativ eindeutigen Test handelt und die Teststatistik Z_N deshalb schnell in Richtung des korrekten Annahmebereichs wächst. Für den Fall, dass der Parameter b groß gewählt wird, nähern sich die Grenzen von NC der Form einer Geraden an, wodurch der Bereich asymptotisch extrem breit wird.

Der Parameter k kann von 4.2 abgeleitet werden

$$k_{Azuma}(a, \alpha, b) = \left(\frac{log(\alpha)}{8a^2(2-3b)} \right)^{\frac{1}{2b-1}}.$$

Die Wahl des Parameters b beeinflusst den Test in verschiedenen Kriterien. Ein klein gewähltes b bedeutet, dass bei einer gut gewählten Vermutung für $|p - p_0|$ gegeben durch γ, eine größere Stichprobenanzahl N benötigt wird, die Testperformance allerdings weniger abhängig von γ ist. Ein groß gewähltes b bedeutet eine engere Anbindung an die Wahrscheinlichkeit des Fehlers erster Art α, im Gegenzug wächst allerdings der Bereich NC. In [10] werden eine Reihe von Werten exemplarisch auf ihre Eignung geprüft, mit dem Schluss, dass $b = \frac{3}{4}$ eine gute Balance zwischen den beschriebenen Auswirkungen von b darstellt. Für die Bestimmung des verbleibenden Parameters a wurden (ähnlich wie für b) mehrere Werte für γ und b geprüft und folgender Term approximativ aufgestellt

$$a_{Azuma} \approx (0.25 - 0.144\alpha^{0.15})\sqrt{\gamma/0.0234}.$$

Dass die Parameter a und b nur näherungsweise bestimmt werden, bedeutet im Rahmen des Tests lediglich eine potentielle Beeinträchtigung in dessen Effizienz, nicht in der Qualität der Aussage.

Das folgende Theorem aus [**reijsbergen2013efficient**], zeigt im Rahmen der beschriebenen Funktion die oberen Schranken der Wahrscheinlichkeiten dafür, dass eine der beiden Hypothesen für ein endliches N akzeptiert wird.

Theorem 4.4.1. *Sei Z_N definiert wie in* **??** *und $u(N) = a(N+k)^b$ mit $a > 0$, $k > 0$, $b \in \left(\frac{2}{3}, 1\right)$. Weiterhin sei die Stichprobengröße N durch einen der beiden Terme definiert: $N = min\{n > 0 : Z_N \notin (l(N), u(N))\}$ oder $N = \infty$, falls $Z_N \in (l(N), u(N))$ für alle $N > 0$. So gilt:*

$$\mathbb{P}(N < \infty \ \textit{und} \ Z_N \geq u(N)) \leq \exp(-8(3b-2)a^2k^{2b-1}), \tag{4.3}$$

$$\mathbb{P}(N < \infty \ \textit{und} \ Z_N \leq l(N)) \leq \exp(-8(3b-2)a^2k^{2b-1}). \tag{4.4}$$

Die Beweise für 4.3 und 4.4 werden an dieser Stelle nicht aufgeführt, sind jedoch in [9] einsehbar.

5 Umsetzung

Das Hauptziel dieser Arbeit ist die Implementierung der in 4.4 vorgestellten Hypothesentests. In diesem Kapitel wird diese Implementierung thematisiert, jedoch nicht in Bezug auf die Funktionalitäten der Tests, da dies bereits im Detail in 4.4 präsentiert wurde. Viel mehr wird zunächst in 5.1 die programmiertechnische Basis dargelegt. In 5.2 wird die eigentliche Implementierung vorgestellt. Diese umfasst sowohl die Einbettung der Testalgorithmen in den bestehenden Code, als auch die vorgenommenen Erweiterungen, welche für die erfolgreiche Integration benötigt wurden.

5.1 Programmiertechnische Basis

Quelle dieses Abschnitts: [8] (S.27). Die gewählten Hypothesentestalgorithmen sollen in den bestehenden Code des Simulationstools integriert werden. Dieser wurde im Rahmen von [8] auf Basis der Java Bibliothek *libhpng* implementiert, welche unter anderem die zum Einlesen der XML Modellierungen hybrider Petri-Netze notwendige Funktionalität bereit stellt. Die Wahl Javas als verwendete Programmiersprache wurde mit dessen hohen Grad an Plattformunabhängigkeit und der Vielseitigkeit bereits existierender Java-Bibliotheken begründet.
Für Programmierung wurde die IDE *Eclipse* genutzt

5.2 Implementierung

Da es sich bei dem Programmieranteil dieser Arbeit um eine Integration neuer Algorithmen in eine bestehende Codierung handelt, sollte sich die Implementierung dieser Algorithmen an dem vorliegenden Programmierstil orientieren. Zudem sollte das Tool an entsprechenden Stellen sinnvoll erweitert werden, sodass der vorhandene Grad an Bedienbarkeit erhalten bleibt.
Im Folgenden werden zuerst die vorgenommenen Erweiterungen aufgelistet und deren Funktionen näher erklärt. Im Anschluss wird auf die, für die eigentliche Implementierung getroffenen Schritte eingegangen.

5.2.1 Erweiterung der Befehlsliste

Im Simulationstool wurde ursprünglich nur ein einzelner Hypothesentest implementiert. Durch das Hinzufügen weiterer Testalgorithmen ist nun ein wichtiger Aspekt, dass der Nutzer möglichst einfach zwischen den einzelnen Algorithmen wechseln kann. Dafür wurde der Befehl *set algorithm* der Befehls-Klasse *commands* hinzugefügt, welcher über

die Eingabe eines entsprechenden Kürzels (SPRT für *Sequentiell Probability Ratio Test*, GCI für *Gauss-Confidence-Intervall*, CR für *Chow-Robbins* und Azuma für *Azuma*) diesen Wechsel ermöglicht. Für die initiale Nutzung des Programms wurde der *Sequentiell Probability Ratio Test* als Standard angegeben. Die benötigten Kürzel für die einzelnen Test lassen sich über den *help*-Befehl abrufen.

Des Weiteren wird in jedem der implementierten Algorithmen ein manuell einzugebender Parameter *guess* benötigt. Dessen Bedeutung wurde bereits in 4.2 erläutert, für ein besseres Verständnis wird diese Thematik allerdings hier noch einmal aufgegriffen. Der Parameter *guess* wird für den *Azuma*-Algorithmus als eine näherungsweise Bestimmung des Wertes von $|p - p_0|$ interpretiert. Für den *Chow-Robbins*- und den *Gauss-CI*-Test steht er dagegen für ein Niveau, ab dem die Mächtigkeit dieser Tests nicht mehr von belang ist. Im Folgenden wird zugunsten der Übersicht jedoch auch für diese beiden Algorithmen die Interpretation des *Azuma*-Tests verwendet. Diese Angabe ist nicht falsch, da sie so interpretiert werden kann, dass für eine höhere Mächtigkeit der Tests der Parameter so nahe wie möglich an den eigentlichen Wert von $|p - p_0|$ angelegt werden soll. Es sei an dieser Stelle nur erwähnt, dass dies eine leicht abgewandelte Formulierung der eigentlichen Bedeutung ist. Der Parameter *guess* ist für alle Tests von großer Wichtigkeit (insbesondere für den *Gauss-Confidence-Intervall*- und den *Chow-Robbins*-Test), da er maßgeblich über die Effizienz und Qualität der Algorithmen entscheidet. Der Wert von *guess* ist standardmäßig auf 0.1 gesetzt, was für eine bessere Effizienz der Tests sorgt, allerdings für kleine $|p - p_0|$ möglicherweise zu einer schlechteren Ergebnisqualität führt. Wenn der Nutzer präzisere Tests durchführen möchte, sollte der Wert von *guess* verringert werden (was gerade bei kleinen $|p - p_0|$ ratsam ist). Dies führt bei allen Algorithmen allerdings auch zu einer schlechteren Effizienz. Je näher *guess* an $|p - p_0|$ heranreicht, desto besser fällt die allgemeine Testperformance aus. Daher wurde der *change guess*-Befehl implementiert, über den es möglich ist, im Falle eines Testergebnisses, welches eher unterhalb eines erwarteten Niveaus liegt, den Parameter entsprechend anzupassen um so ein besseres Resultat zu erzielen.

Letztlich wurde die Möglichkeit implementiert, mehrere Hypothesentests hintereinander laufen zu lassen, wodurch empirische Daten zu einem Testdurchlauf gesammelt werden können. Über den Befehl *change testruns* lässt sich die Anzahl der Tests ändern. Diese wurde standardmäßig auf 1 gesetzt. Mit dem Abschluss eines Testdurchlaufs, werden die Resultate präsentiert. Diese umfassen zum einen die prozentualen Anteile der Tests, in denen die eingegebene Eigenschaft erfüllt, beziehungsweise nicht erfüllt wurde und der ergebnislosen Tests. Zum anderen werden die Größen der minimalen, maximalen und durchschnittlich benötigten Stichprobenanzahl angezeigt, was Auskunft über die Varianz dieser Größe gibt.

5.1 zeigt eine Abbildung der Konsole mit den aufgeführten Befehlen, 5.2 zeigt die Ausgabe der Konsole nach einem Testdurchlauf.

5.2.2 Implemetierung der Hypothesentests

Im ursprünglichen Simulator wurde im Fall der Durchführung eines Hypothesentests die geforderte Funktionalität in der Klasse *SimulationHandler* des Pakets *simulation* aufgerufen. Anstatt die Testalgorithmen ebenfalls in dieser Klasse zu implementierten, wurde eine neue Klasse *HypothesisTesting* angelegt und sämtliche Funktionalitäten der neu hinzugefügten und des vorhandenen Tests in diese Klasse verschoben. Wenn der Nutzer nun einen Hypothesentests ausführt, wird eine Instanz von *HypothesisTesting* erstellt und die Methode *performTesting* ausgerufen, welcher die ID des durch *setAlgorithm* festgelegten Hypothesentestalgorithmus übergeben wird. Je nachdem, ob die für den Tests benötigte Simulation mit einer konkreten Anzahl an Durchläufen durchgeführt werden soll oder nicht (einstellbar über den *set fixedruns*-Befehl), wird die entsprechende Variante des Hypothesentests ausgeführt.
Wie genau die Tests arbeiten wurde bereits in 4.4 detailliert erklärt und wird an dieser Stelle nicht ein weiteres Mal beschrieben.

Abbildung 5.1: Abbildung der implementierten Konsolenbefehle

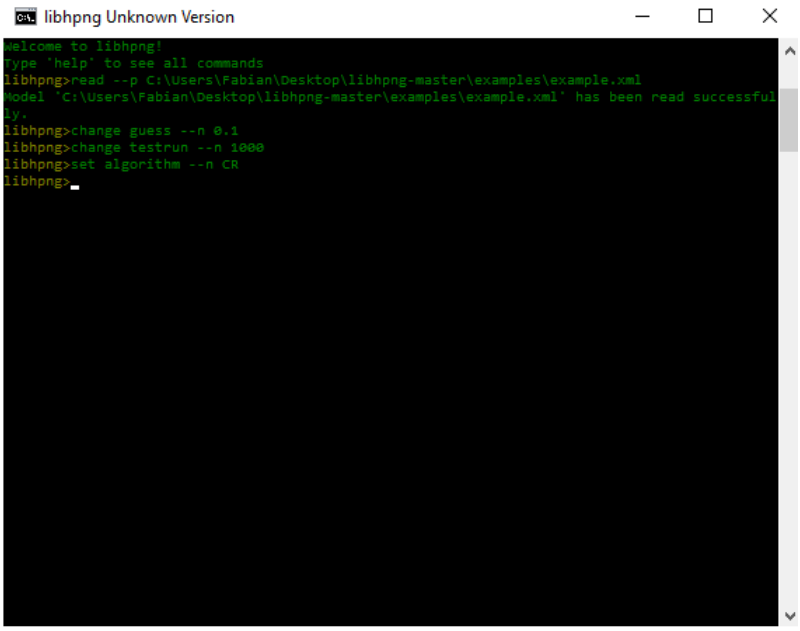
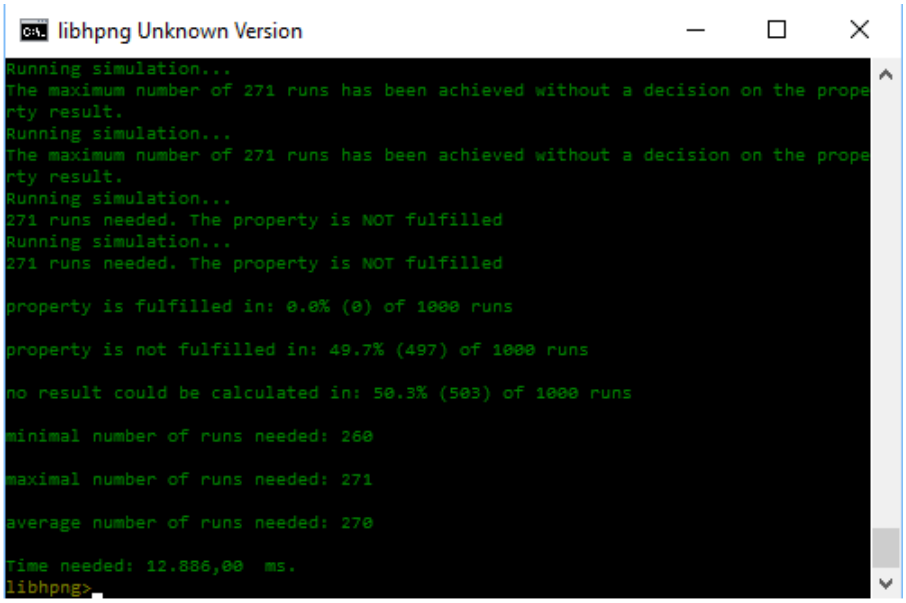


Abbildung 5.2: Abbildung der Ausgabe nach Abschluss des in 5.1 angezeigten Testdurchlaufs



6 Testdurchläufe und Performanceanalyse

In diesem Kapitel werden die implementierten Algorithmen anhand von Modellierungen hybrider Petri-Netze auf verschiedene Aspekte hin getestet und miteinander verglichen. Diese Aspekte umfassen: die Laufzeiten der Testdurchläufe, die Anzahl der Stichproben, die für die Bestimmung eines Ergebnisses benötigt wurden, die Varianz dieser Größe und die Korrektheit der berechneten Resultate, beziehungsweise wie oft ein Durchlauf endet, ohne dass ein Resultat berechnet werden konnte. In 6.1 werden zunächst die Erwartungen an Funktionalitäten und Performance der Algorithmen aufgelistet. Im Anschluss werden in 6.2 die Algorithmen anhand eines einfachen Beispiels und mehrerer Testfälle getestet. Letztlich werden in 6.3 über einer Fallstudie weitere Tests zu einer komplexeren Modellierung durchgeführt.

6.1 Erwartungen an die Algorithmen

In 4.2 und 4.4 wurden bereits die Eigenschaften der Hypothesentestalgorithmen genannt. Anhand dieser Charakteristika werden nun Erwartungen an die Performance dieser Hypothesentests gestellt, welche für die Auswertung der Testdurchläufe wieder aufgegriffen werden.

6.1.1 Gauss-Confidence-Intervall

Da die Anzahl der benötigten Stichproben beim *Gauss-CI*-Test allein von den aktuellen Modell- und Eingabeparameter abhängig ist (namentlich *guess* und p_0), gilt es herauszufinden, inwieweit sich diese Größe bei einer Variation der Parameter verändert. Genauer stellt sich die Frage, wie sehr diese Parameter zu Gunsten einer höheren Effizienz umgestellt werden können, ohne dabei die Ergebnisqualität maßgeblich einzuschränken.

6.1.2 Chow-Robbins

In Bezug auf Effizienz und Qualität der Ergebnisse ist der *Chow-Robbins*-Test ist bis zu einem gewissen Grad mit dem *Gauss-CI*-Test vergleichbar. Dennoch bestehen einige für die Analyse wichtige Unterschiede. So arbeitet der Algorithmus nicht ausschließlich mit einer einer festen Stichprobengröße, sondern auch sequentiell, weswegen mit einer Varianz dieser Größe zu rechnen ist. Es ist daher wichtig zu analysieren, wie stark diese Varianz ausfällt.
Der Algorithmus sollte zudem schneller als der *Gauss-CI*-Tests zu einem Ergebnis kommen, falls p weiter von $0,5$ entfernt ist als p_0 und braucht andernfalls länger (siehe [10]). Dies hat folgende Begründung: die Effizienz beider Algorithmen wird maßgeblich über

den Wert von *guess* bestimmt. Zusätzlich kann der *Chow-Robbins*-Test schneller ein Ergebnis erzielen, je näher p an 0 oder 1 heranreicht (siehe 4.4.2). Analog dazu braucht der *Gauss-CI*-Test weniger Stichproben, je näher p_0 an 0 oder 1 heranreicht (siehe 4.4.1). Liegt p_0 nun näher an 0,5 als p, sollte der *Chow-Robbins*-Algorithmus die bessere Effizienz aufweisen.
Es gilt also zu überprüfen, wie sehr sich diese Relation der beiden Hypothesentests in der Praxis widerspiegelt und unter welchen exakten Umständen der eine Test dem anderen vorzuziehen ist.

6.1.3 Azuma

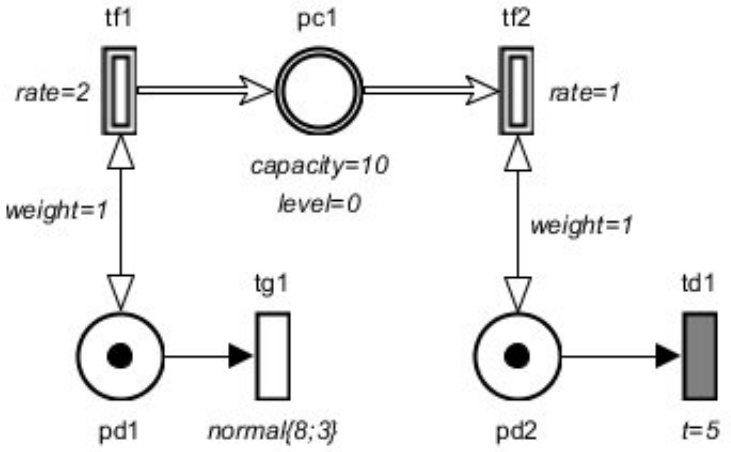
Im Falle des *Azuma*-Hypothesentest liegt der Fokus darauf, wie sich die Anzahl der benötigten Stichproben für verschieden große $|p - p_0|$ im Vergleich zu den anderen implementierten Algorithmen verhält und wie die Varianz dieser Größe ausfällt. Zudem soll getestet werden, ob der Tests tatsächlich in der Lage ist, selbst bei sehr kleinem $|p - p_0|$ ein korrektes Ergebnis erzielen zu können, was laut seiner Konstruktion der Fall sein müsste. Hierbei sei noch einmal angemerkt, dass "stets"in diesem Zusammenhang keine absolute Garantie bedeutet, sondern eine Garantie im Rahmen eines entsprechenden Konfidenzniveaus. Einen Hypothesentest zu konstruieren, dessen Garantie für die Korrektheit genau 100% beträgt (das heißt auch über das gegebene Konfidenzniveau hinaus) ist statistisch nicht möglich aufgrund der Gefahr des Fehlers erster und zweiter Art.
Ebenfalls interessant ist die Frage, in welchen Fällen sich die Anwendung des *Azuma* Tests trotz der Gefahr einer hohen Stichprobengröße lohnt, beziehungsweise in welchen Fällen der Kompromiss zwischen Effizienz und Qualität am besten ausfällt.

6.2 Erste Testdurchläufe

Zunächst werden anhand eines einfachen Beispiels einige Tests durchgeführt. Das Ziel dabei ist es, zu überprüfen, ob die Ergebnisse der Erwartungshaltung gerecht werden und wie erdacht funktionieren. Außerdem wird so ein erster Einblick in die allgemeine Testperformance gewonnen.

Für die Beispielmodellierung 6.1 werden Hypothesentests für die Eigenschaft **10.0:P>X(fluidlevel('pc1')>=9.0)** durch geführt, wobei in diesem Fall $\mathbf{P} \approx 0,6313$ gilt, mit einer ungefähren Abweichung von $+/-0,005$, eine maximale Stichprobengröße von $N = 100000$ vorliegt, für jeden $\mathbf{X}$ Wert 100 Tests durchgeführt wurden und die Begrenzungswerte α und β der Fehler erster und zweiter Art auf $\alpha = \beta = 0,05$ gesetzt wurden. Für diese Tests werden verschiedene Werte für $\mathbf{X}$ und dem Parameter *guess* genutzt.
Die für den Parameter *guess* verwendeten Werte beschränken sich dabei auf 0,1 und 0,01. Dies hat mehrere Gründe. Zum einen ist 0,1 der zu empfehlende Maximalwert für *guess*, da größere Werte zu einem starken Einbruch der Ergebnisqualität führt, ohne dabei einen signifikanten Effizienzgewinn zu erzielen. Analoges gilt für 0,01 als Minimalwert. Hier

Abbildung 6.1: Beispiel hybrides Petri-Netz. Quelle: [3], S.39



führen kleinere Werte zu einem Einbruch der Effizienz, ohne dass sich die Ergebnisqualität maßgeblich verbessert. Auf Werte zwischen diesen Grenzen wurde verzichtet, da die Unterschiede, welche gezeigt werden sollen, mit 0,1 und 0,01 am deutlichsten ausfallen.

6.2.1 Sequentiell-Probability-Ratio-Test Testdurchläufe

Zu Beginn werden hier die Ergebnisse des SPRT -Algorithmus vorgestellt, welcher nicht Teil dieser Arbeit ist, sondern bereits im Tool implementiert wurde. Diese Ergebnisse werden aufgeführt, um sie mit denen der neu hinzugefügten Algorithmen zu vergleichen zu können. Die Funktionsweise des Tests wird hier nicht näher erläutert, kann aber in [8] (S.52-54) eingesehen werden.

Der Test schafft es in den ersten beiden Testfällen mit einer relativ niedrigen durchschnittlichen Stichprobenanzahl und in kurzer Zeit überwiegend das korrekte Ergebnis zu erzielen. Für die letzten beiden Fälle, in denen $|p - p_0|$ sehr klein ausfällt, beginnt die Ergebnisqualität allerdings abzufallen und es geschieht häufiger, dass ein falsches Ergebnis berechnet wird.
Es sei angemerkt, dass in den Fällen in denen sich die ergebnisbezogenen Prozentualwerte nicht zu 100% aufaddieren, der Rest den Anteil der Tests darstellt, in denen innerhalb von 100000 Simulationsdurchläufen kein Ergebnis berechnet werden konnte und der Durchlauf also ergebnislos endete.

6.2.2 Gauss-Confidence-Intervall Testdurchläufe

Für $guess = 0{,}1$ erzielt der Test bei kleiner Stichprobengröße und kurzer Laufzeit zu Beginn meist das korrekte Ergebnis. Allerdings beginnt diese Ergebnisqualität recht schnell zu sinken, sobald $|p-p_0|$ abnimmt und sich immer weiter von *guess* entfernt. In den Fällen

Tabelle 6.1: Ergebnisse des *SPRT* Algorithmus

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	Durchschnittliche Laufzeit	korrekte Tests (in %)	inkorrekte Tests (in %)
$\mathbf{X} = 0{,}73$	∅N = 599	$N_{min} = 373$, $N_{max} = 885$	19 ms	99%	1%
$\mathbf{X} = 0{,}67$	∅N = 1107	$N_{min} = 757$, $N_{max} = 1160$	20 ms	96%	4%
$\mathbf{X} = 0{,}64$	∅N = 1127	$N_{min} = 1088$, $N_{max} = 1158$	34 ms	46%	20%
$\mathbf{X} = 0{,}638$	∅N = 1129	$N_{min} = 1099$, $N_{max} = 1166$	36 ms	32%	42%

$\mathbf{X} = 0{,}64$ und $\mathbf{X} = 0{,}638$, das heißt für $|p - p_0| \approx 0.009$, beziehungsweise $|p - p_0| \approx 0.007$, kann somit nur noch sehr selten ein Ergebnis berechnet werden und zusätzlich steigt dabei die Gefahr, dass ein falsches Ergebnis berechnet wird (dies ist daran zu erkennen, dass die ergebnisbezogenen Prozentwerte zusammen nicht 100% ergeben), wobei dieser Anstieg sehr gering ausfällt.
Wird der *guess*Parameter hingegen entsprechend angepasst, so ist direkt erkennbar, dass die Ergebnisqualität drastisch angestiegen ist. So konnte in allen vier Testfällen mit nur geringfügigen Abweichungen das korrekte Ergebnis berechnet werden und zudem wurden keine falschen Aussagen getroffen. Allerdings zeichnet sich auch ein drastischer Verlust an Effizienz ab.
Es sei auch hier noch einmal angemerkt, dass in Fällen, wo sich die ergebnisbezogenen Prozentualwerte nicht zu 100% aufaddieren, der Rest den Anteil der Tests darstellt, in denen ein inkorrektes Ergebnis berechnet wurde.

Tabelle 6.2: Ergebnisse des *Gauss-CI* Algorithmus für $guess = 0{,}1$

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	Durchschnittliche Laufzeit	korrekte Tests (in %)	ergebnislose Tests (in %)
$\mathbf{X} = 0{,}73$	∅N = 181	$N_{min} = N_{max} = 181$	3 ms	96%	4%
$\mathbf{X} = 0{,}67$	∅N = 214	$N_{min} = N_{max} = 214$	2 ms	76%	24%
$\mathbf{X} = 0{,}64$	∅N = 1127	$N_{min} = N_{max} = 1127$	4 ms	9%	86%
$\mathbf{X} = 0{,}638$	∅N = 1129	$N_{min} = N_{max} = 1129$	7 ms	8%	84%

Tabelle 6.3: Ergebnisse des *Gauss-CI* Algorithmus für $guess = 0{,}01$

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	Durchschnittliche Laufzeit	korrekte Tests (in %)	ergebnislose Tests (in %)
$\mathbf{X} = 0{,}73$	∅N = 21075	$N_{min} = N_{max} = 21075$	210 ms	100%	0%
$\mathbf{X} = 0{,}67$	∅N = 23738	$N_{min} = N_{max} = 23738$	219 ms	100%	0%
$\mathbf{X} = 0{,}64$	∅N = 24777	$N_{min} = N_{max} = 24777$	235 ms	98%	2%
$\mathbf{X} = 0{,}638$	∅N = 24839	$N_{min} = N_{max} = 24839$	241 ms	89%	11%

6.2.3 Chow-Robbins Testdurchläufe

Die Resultate des *Chow-Robbins*-Tests weisen eine starke Ähnlichkeit zu denen des *Gauss-CI*-Tests auf. Auch hier ist zu beobachten, dass die Ergebnisqualität für *guess*=

0,1 schon im zweiten Testfall deutlich sinkt, sogar ein wenig schneller als beim *Gauss-CI*-Algorithmus.
Für *guess*= 0,01 steigt die Ergebnisqualität sichtlich an, wobei auch hier deutlich mehr Stichproben benötigt werden. Die Varianz dieser Größe befindet sich jedoch in einem annehmbaren Bereich. Es fällt zusätzlich auf, dass keine inkorrekte Test vorliegen.

Tabelle 6.4: Ergebnisse des *Chow-Robbins* Algorithmus für *guess* = 0,1

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	Durchschnittliche Laufzeit	korrekte Tests (in %)	ergebnislose Tests (in %)
X = 0,73	∅N = 257	N_{min} =210, N_{max} = 266	2 ms	96%	4%
X = 0,67	∅N = 256	N_{min} = 256, N_{max} = 269	2 ms	65%	35%
X = 0,64	∅N = 248	N_{min} = 240, N_{max} = 270	2 ms	9%	81%
X = 0,638	∅N = 261	N_{min} = 231, N_{max} = 267	3 ms	13%	87%

Tabelle 6.5: Ergebnisse des *Chow-Robbins* Algorithmus für *guess* = 0,01

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	Durchschnittliche Laufzeit	korrekte Tests (in %)	ergebnislose Tests (in %)
X = 0,73	∅N = 25196	N_{min} =25035, N_{max} = 25341	134 ms	100%	0%
X = 0,67	∅N = 25223	N_{min} = 25048, N_{max} = 25405	209 ms	100%	0%
X = 0,64	∅N = 25212	N_{min} = 25089, N_{max} = 25378	199 ms	97%	3%
X = 0,638	∅N = 25227	N_{min} = 25143, N_{max} = 25324	211 ms	86%	14%

6.2.4 Azuma Testdurchläufe

Die Stichprobengröße des *Azuma*-Hypothesentests steigt schon für *guess* = 0,1 recht stark an, was zum einen bei einem Test der dritten Klasse zu erwarten ist (siehe 4.2), zum anderen aber auch zeigt, welchen starken Einfluss der Wert von *guess* auf die Effizienz der Algorithmen hat. In den letzten beiden Fällen wird dies besonders deutlich. Da *guess* eine ungefähre Vermutung für $|p - p_0|$ ist und in diesem Beispiel mit 0,1 weit vom eigentlich Wert (≈ 0.009, beziehungsweise ≈ 0.007) entfernt liegt, braucht der rein sequentiell arbeitende *Azuma*-Algorithmus eine erheblich höhere Stichprobengröße, welche das gesetzte Maximum von 100000 überschreitet.
Wird der Parameter indes auf 0,01 gesetzt, so gibt der Test in allen vier Fällen das korrekte Resultat aus und vermeidet dabei jegliche Abweichungen. Jedoch kann man auch hier einen erheblichen Anstieg der benötigten Stichprobengrößen für abnehmende $|p - p_0|$ erkennen.
Sowohl für *guess* = 0,1 als auch für 0,01 konnte beobachtet werden, dass die Varianz für kleiner werdende $|p - p_0|$ enorme Ausmaße annimmt.

Tabelle 6.6: Ergebnisse des *Azuma* Algorithmus für *guess* = 0,1

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	Durchschnittliche Laufzeit	korrekte Tests (in %)	ergebnislose Tests (in %)
X = 0,73	∅N = 376	N_{min} =100, N_{max} = 1018	3 ms	100%	0%
X = 0,67	∅N = 3792	N_{min} = 449, N_{max} = 9918	21 ms	100%	0%
X = 0,64	∅N = /	N_{min} = /, N_{max} = /	29 ms	0%	100%
X = 0,638	∅N = /	N_{min} = /, N_{max} = /	/	0%	100%

Tabelle 6.7: Ergebnisse des *Azuma* Algorithmus für *guess* = 0,01

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	Durchschnittliche Laufzeit	korrekte Tests (in %)	ergebnislose Tests (in %)
X = 0,73	∅N = 1882	N_{min} =1496, N_{max} = 2309	3 ms	100%	0%
X = 0,67	∅N = 5392	N_{min} = 3559, N_{max} = 8810	21 ms	100%	0%
X = 0,64	∅N = 4183	N_{min} = 11506, N_{max} = 85757	76 ms	100%	0%
X = 0,638	∅N = 54502	N_{min} = 25874, N_{max} = 97383	278 ms	100%	0%

6.2.5 Effizienzunterschied zwischen *Chow-Robbins* und *Gauss-CI*

An dieser Stelle soll der in 6.1.2 genannte, unter bestimmten Bedingungen geltende Effizienzunterschied zwischen *Chow-Robbins*- und *Gauss-CI*-Test näher betrachtet werden. Wie in 6.1.2 beschrieben, sollte der *Chow-Robbins*-Algorithmus für ein Ergebnis weniger Stichproben als der *Gauss-CI*-Algorithmus, falls p_0 näher an 0,5 heranreicht als p und andernfalls mehr.
Hierfür werden anhand von 6.1 fünf Testfälle ausgewertet:

- 1) **8.0:P>0.975(fluidlevel('pc1')>=4.0)** mit **P**≈ 0.875
- 2) **8.0:P>0.775(fluidlevel('pc1')>=4.0)**
- 3) **8.0:P>0.59(tokens('pd1')=0)** mit **P**≈ 0.5
- 4) **7.3:P>0.5(tokens('pd1')=0)** mit **P**≈ 0.40

In den Testfällen 1) und 2) liegt p mit ≈ 0.0875 sehr nahe bei 1. Es wird einmal für $p < p_0$ und einmal für $p > p_0$ getestet. Zu erwarten ist, dass der *Gauss-CI*-Algorithmus im Fall 1), das heißt für $p > p_0$ eine bessere Effizienz, als der *Chow-Robbins*-Test aufweist, bei gleicher Ergebnisqualität. Im Testfall 2) sollte hingegen das Gegenteil der Fall sein. In den Testfällen 3) und 4) sollen Test für suboptimale Bedingungen für die Testalgorithmen durchgeführt werden, um zu analysieren welcher Algorithmus unter solchen Bedingungen eventuell besser performt. Mit $p = 0,5$ in 3) gelten suboptimale Bedingungen für den *Chow-Robbins*-Test und mit $p_0 = 0,5$ in 4) gelten suboptimale Bedingungen für den *Gauss-CI*-Test.

Bei diesen Testfällen liegen folgende Parameter vor: *guess*= 0,01, $\alpha = \beta$= 0,05, maximale Stichprobengröße = 100000, Anzahl der Durchläufe pro Test = 100.

Tabelle 6.8: Ergebnisse des *Chow-Robbins* Tests

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	korrekte Tests (in %)	ergebnislose Tests (in %)
Fall 1)	⌀N = 11590	N_{min} =11085, N_{max} = 11963	100%	0%
Fall 2)	⌀N = 11494	N_{min} = 11008, N_{max} = 11879	100%	0%
Fall 3)	⌀N = 27054	N_{min} = 27047, N_{max} = 27056	100%	0%
Fall 4)	⌀N = 26154	N_{min} = 25980, N_{max} = 26258	100%	0%

Tabelle 6.9: Ergebnisse des *Gauss-CI* Tests

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	korrekte Tests (in %)	ergebnislose Tests (in %)
Fall 1)	⌀N = 2086	$N_{min} = N_{max} = 2086$	100%	0%
Fall 2)	⌀N = 18566	$N_{min} = N_{max} = 18566$	100%	0%
Fall 3)	⌀N = 26075	$N_{min} = N_{max} = 26075$	100%	0%
Fall 4)	⌀N = 27050	$N_{min} = N_{max} = 27050$	100%	0%

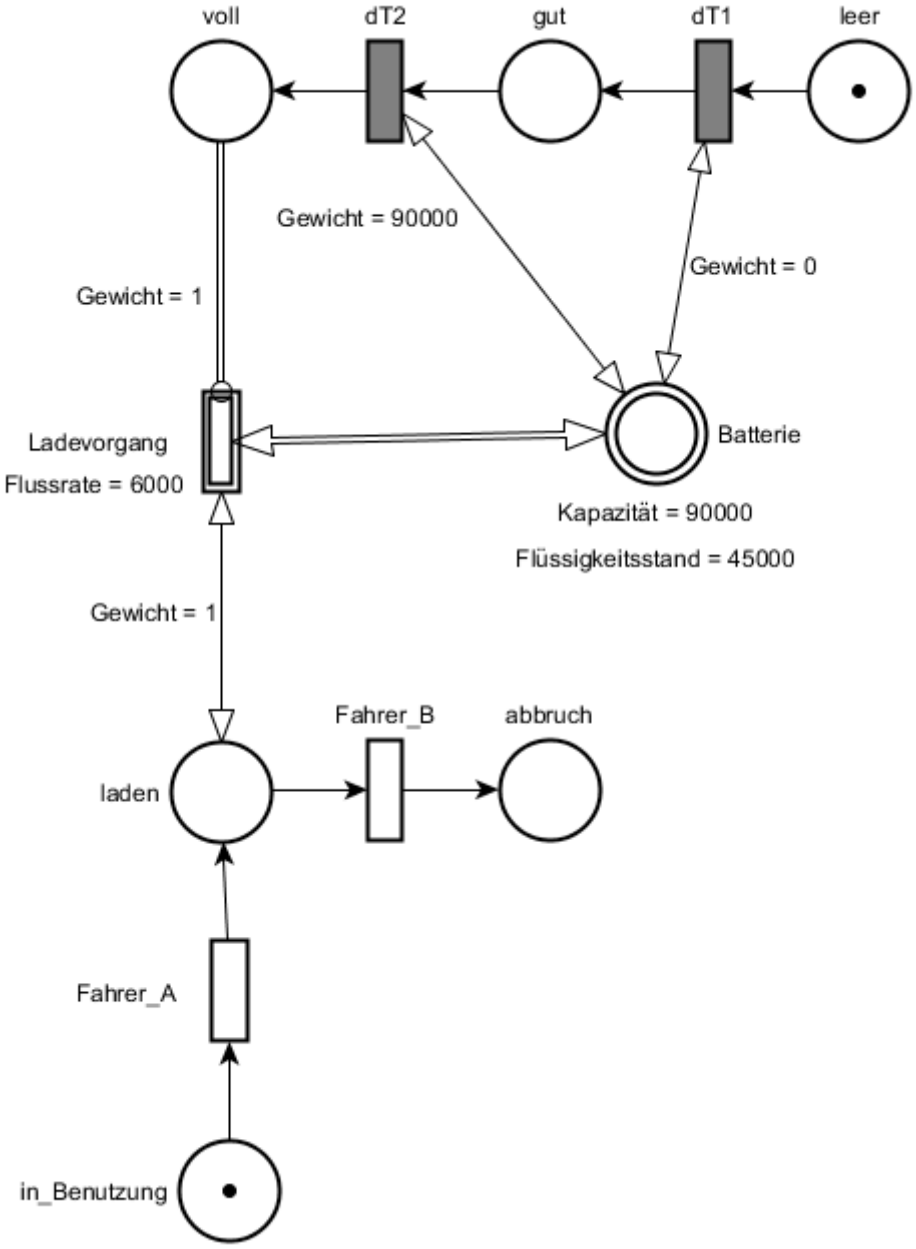
Wie zu erwarten profitiert der *Gauss-CI*-Test im Fall 1) von dem großen Wert für p_0 und schafft es dadurch deutlich schneller als der *Chow-Robbins*-Algorithmus zu einem (korrekten) Ergebnis zu kommen. Im Fall 2) hingegen erzielt der *Chow-Robbins*-Algorithmus die bessere Effizienz, auch hier fällt der Unterschied sehr deutlich aus. In den Fällen 3) und 4) liegen die gemessenen Effizienzwert wiederum sehr nahe beieinander in dem Sinne, dass die Anzahl der benötigten Stichproben des *Chow-Robbins*-Algorithmus aus 3) nahe bei der Anzahl der Stichproben des *Gauss-CI*-Tests aus 4) liegen. Analog sind die Effizienzwerte des *Chow-Robbins*-Tests aus 4) mit denen des *Gauss-CI*-Algorithmus aus 3) vergleichbar. Die beiden Hypothesentests performen also unter den jeweils eigenen suboptimalen Gegebenheiten sehr ähnlich.

6.3 Fallstudie

Nachdem nun eine Übersicht über die Leistungsfähigkeit der implementierten Hypothesentests gegeben wurde soll jetzt anhand eines praktisch orientierten Beispiels weitere Tests durchgeführt werden. In [8], Kapitel 5 wird eine Modellierung für den Aufladungsprozess eines Elektroautos dargestellt, auf dessen Basis eine eigene Modellierung erstellt

wurde. Diese Modellierung wird im Folgenden erläutert, mit anschließender Auflistung und Durchführung verschiedener Testfälle.

Abbildung 6.2: Petri-Netz zur Modellierung des Ladevorgangs. Quelle: Eigene Darstellung basierend auf [8], S.57



Modellierung

Das Modell soll den Ladeprozess eines Elektroautos unter einem bestimmten Nutzerverhalten darstellen. Neben den grafischen Aspekten besteht der wesentliche Unterschied zur Modellierung in [8] darin, dass der Fokus vom eigentlichen Aufladen des Autos genommen und auf die Verteilung der Benutzungszeiten (in denen natürlich nicht aufgeladen werden kann) gelegt wurde. Genauer sollen zwei Nutzer modelliert werden, welche nacheinander mit dem Auto fahren und die Batterie nur zwischen diesen Nutzungszeiten aufgeladen wird. 6.2 zeigt das verwendete Petri-Netz, welches an dieser Stelle kurz beschrieben wird.
Im oberen Teil befinden sich drei diskrete Zustände *leer, gut* und *voll*, welche den Ladestand des Wagens symbolisieren. Die zugehörigen deterministischen Transitionen sind dabei über Testkanten mit dem stetigen Zustand *Batterie* verbunden, welcher den konkreten Ladestand des Autos angibt, mit einem maximalen Stand von 90000(kWh). Wie der Name suggeriert, dient der stetige Zustand *Ladevorgang* dazu, das Aufladen der Batterie zu modellieren. Diese Transition ist jedoch nur dann aktiviert, wenn der Zustand *voll* kein Token hält (zu sehen an der Hemmkante) und der diskrete Zustand *laden* genau ein Token hält (zu sehen an der Testkante). Ob *laden* über ein Token verfügt, hängt dabei von der stochastischen Transition $Fahrer\_A$ ab. Sobald diese schaltet, wird von dem diskreten Zustand $in\_Benutzung$ ein Token genommen und ein Token auf *laden* gelegt, das heißt der Wagen ist nun nicht mehr in Benutzung und kann aufgeladen werden. Jedoch nur so lange, bis die zweite stochastische Transition $Fahrer\_B$ schaltet, worauf der Ladevorgang wieder abgebrochen wird, dargestellt durch ein Token auf dem diskreten Zustand *abbruch*.

Im ersten Testfall soll geprüft werden, ob die Wahrscheinlichkeit unter 80% liegt, dass der Batteriestand mindestens die 90% Marke (das heißt mindestens 81000kWh) erreicht, bevor der zweite Fahrer den Ladevorgang abbricht und bis zum Ende des Tages nicht mehr reaktiviert. Dabei gelten 6 Zeiteinheiten als eine Stunde, das heißt eine Tag besteht aus 144 Zeiteinheiten. Zudem sei angegeben, dass der erste Fahrer den Wagen bei einem Ladestand von genau 45000kWh abgibt.
Die zu testende Formulierung lautet:

144.0:P<0.8(fluidlevel('Batterie')>=81000.0)

align Anmerkung: für **T** wurde 144.0 (das Ende eines Tages) gewählt, da der Zeitpunkt, zu dem der zweite Fahrer den Wagen in Anspruch nimmt, durch eine Verteilungsfunktion berechnet und nicht exakt bestimmt werden kann. 144.0 genügt der Modellierung, da sich der Stand der Batterie nach der Schaltung von $Fahrer\_B$ nicht mehr verändert.
Die Werte der stochastischen Transitionen seien wie folgt: die Verteilung von $Fahrer\_A$ ist gegeben durch eine Normalverteilung mit dem Erwartungswert $\mu = 40$ und der Varianz $\sigma = 6$. Die Verteilung von $Fahrer\_B$ ist gegeben durch eine Normalverteilung mit dem Erwartungswert $\mu = 14$ und der Varianz $\sigma = 6$. Dabei ist zu beachten, dass $Fahrer\_B$ erst aktiviert werden kann, wenn $Fahrer\_A$ geschaltet hat, das heißt ein Token auf *laden* liegt. Der Erwartungswert $\mu = 14$ ist also als Zeiteinheiten nach dieser Schaltung zu verstehen.

Eine erste Betrachtung dieser Werte lässt vermuten, dass die Wahrscheinlichkeit **P** aus 6.3 recht hoch liegt (aufgrund der Tatsache, dass die Erwartungswerte bei (in diesem Kontext) niedriger Varianz recht weit auseinander liegen). Man darf also annehmen, dass $|p - p_0|$ nicht allzu klein ausfällt. Diese einfache Beobachtung kann für die Auswahl eines geeigneten Tests sowie für die optimale Bestimmung des Parameters *guess* genutzt werden. Da zu erwarten ist, dass $p > p_0$ gilt, dürfte der *Chow-Robbins*-Test die besten Effizienzergebnisse erzielen. Da p_0 mit 0,8 recht nahe bei 1 liegt, ist zudem anzunehmen, dass auch der *Gauss-CI*-Algorithmus gute Ergebnisse erzielt, während der *Azuma*-Test die höchste durchschnittliche Stichprobenanzahl benötigen dürfte. Der vermutetermaßen relativ große Wert für $|p - p_0|$ erlaubt zudem die Wahl eines größeren Wertes für den Parameter *guess*, was die Effizienz aller Tests positiv beeinflusst, ohne dabei die Ergebnisqualität maßgeblich zu beeinträchtigen.
Für die Testdurchläufe gelten folgende Parameter: 100 Testdurchläufe, $\alpha = \beta = 0{,}05$, $guess = 0{,}1$ und ein maximales N = 100000.

Tabelle 6.10: Ergebnisse der Testdurchläufe für $guess = 0{,}1$

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	korrekte Tests (in %)	inkorrekte Tests (in %)	ergebnislose Tests (in %)
Chow-Robbins	∅N = 102	$N_{min} = 104$,$N_{max} = 133$	99%	0%	1%
Gauss-CI	∅N = 132	$N_{min} = N_{max} = 132$	98%	0%	2%
Azuma	∅N = 320	$N_{min} = 189$,$N_{max} = 586$	100%	0%	0%

Als Referenz werden hier die Ergebnisse für $guess = 0{,}01$ angegeben.

Tabelle 6.11: Ergebnisse der Testdurchläufe für $guess = 0{,}01$

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	korrekte Tests (in %)	inkorrekte Tests (in %)	Tests mit N > 100000 (in %)
Chow-Robbins	∅N = 8935	$N_{min} = 8353$,$N_{max} = 9598$	100%	0%	0%
Gauss-CI	∅N = 16983	$N_{min} = N_{max} = 16983$	100%	0%	0%
Azuma	∅N = 1749	$N_{min} = 1529$, $N_{max} = 1973$	100%	0%	0%

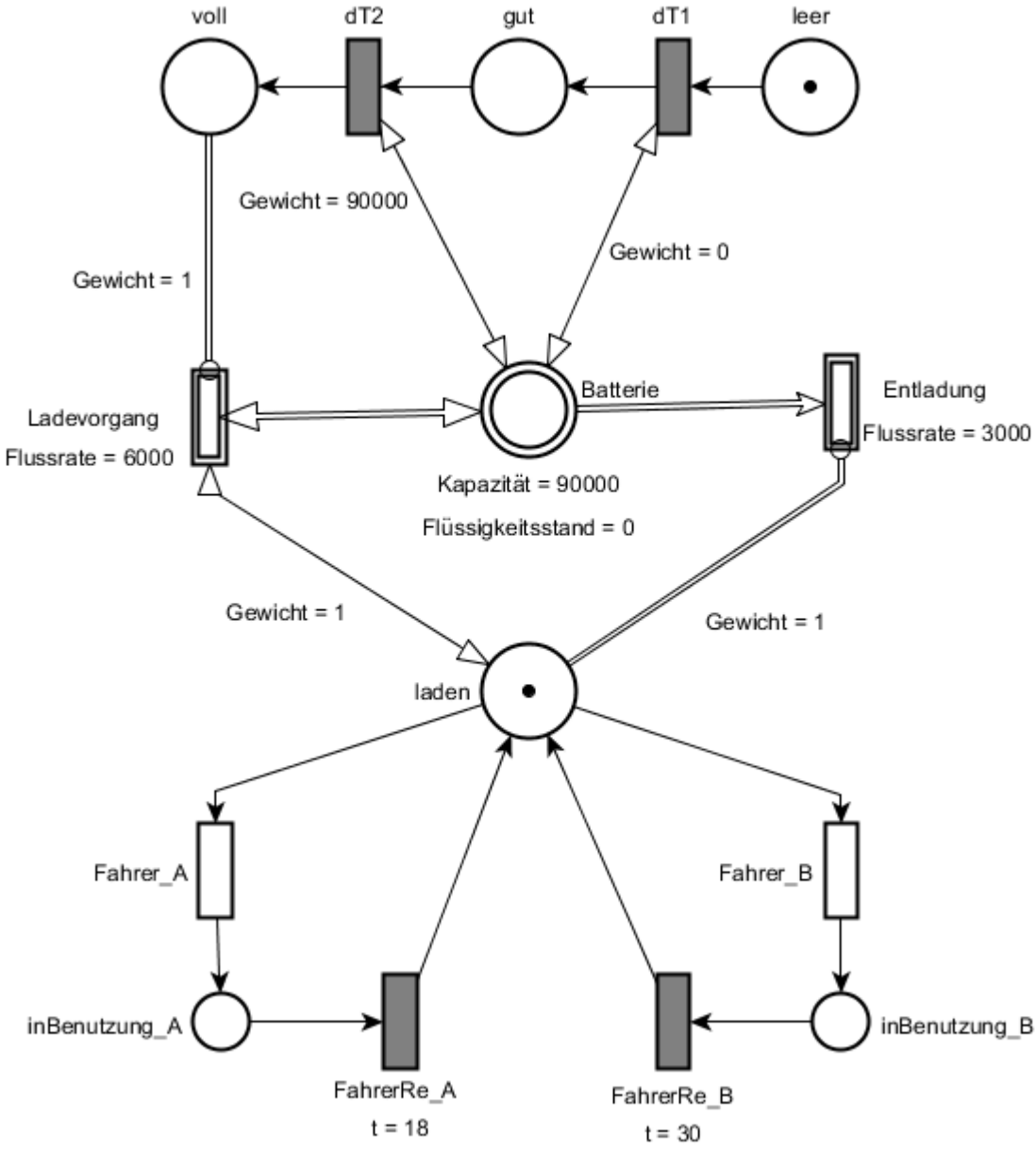
Wie in der ersten Tabelle deutlich zu sehen ist, konnten mit $guess = 0{,}1$ allgemein sehr gute Testergebnisse in Bezug auf Effizienz und Qualität erzielt werden, wobei der *Chow-Robbins* -Algorithmus wie erwartet das beste Gesamtresultat erzielt hat. Ein kurzer Vergleich mit der zweiten Tabelle zeigt, dass das Verhältnis zwischen Effizienz und Qualität für $guess = 0{,}01$ ein deutlich schlechteres ist.

Erweiterte Modellierung

Nun soll die Modellierung leicht verändert werden, um den Komplexitätsgrad zu erhöhen. Statt eine strikte Reihenfolge der Fahrer bei der Benutzung des Wagens vorzugeben,

haben diese nun gleichermaßen Zugriff auf das Auto. Dabei werden die Nutzungszeiten der Fahrer durch verschiedene Verteilungsfunktionen angegeben und jeder Fahrer beansprucht das Auto für einen konkreten Zeitraum. Zusätzlich kann in dieser Modellierung jeder Fahrer mehrfach auf den Wagen zugreifen, das heißt die entsprechenden Transitionen können mehrfach schalten. 6.3 zeigt den dieser Modellierung zugrundeliegenden Graphen. Für die stochastischen Transitionen wurden folgende Werte gewählt: die Verteilungsfunktion von$Fahrer_A$ ist eine Gleichverteilung mit a = 0,0 und b= 15,0. Die Verteilungsfunktion von$Fahrer_B$ ist eine Normalverteilung mit μ = 10 und σ= 5,0.

Abbildung 6.3: Petri-Netze der erweiterten Modellierung, Quelle: Eigene Darstellung basierend auf [8], S.57



Es soll nun getestet werden, ob die Wahrscheinlichkeit über 10%, beziehungsweise über 6% liegt, dass im Verlauf eines halben Tages (t = 0 bis t= 72) der Fall eintritt, dass die Batterie leer ist und der Wagen dennoch zur gleichen Zeit beansprucht wird. Die

zugehörige Formulierung lautet wie folgt:

0.0:P>X(U[0.0,72.0](tokens('leer')=1,tokens('laden')=0)),
wobei $\mathbf{X} = 0{,}1$ oder $\mathbf{X} = 0{,}06$.

Aufgrund der erhöhten Komplexität ist das Aufstellen von Prognosen durch eine kurze Betrachtung der Parameter nicht ohne weiteres möglich. Es kann lediglich vermutet werden, dass die Wahrscheinlichkeit, dass der beschriebene Testfall eintritt, recht gering ausfallen dürfte. Ähnlich wie im vorherigen Modell, liegen sowohl p_0 als auch p (vermutetermaßen) weit von 0,5 entfernt, weshalb gute Effizienzwerte für die *Chow-Robbins*- und *Gauss-CI*-Tests erwartet werden. Allerdings kann dies gerade für den Fall $\mathbf{X} = 0{,}06$ zu einer geringeren Ergebnisqualität führen, da $|p - p_0|$ wahrscheinlich recht klein ausfällt. Hierbei dürften sich die Ergebnisse des *Azuma*-Algorithmus deutlich von denen der anderen Tests abheben, allerdings ist hier auch eine größere Anzahl an benötigten Stichproben zu erwarten.

Tabelle 6.12: Ergebnisse der Testdurchläufe für $\mathbf{X} = 0{,}1$

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	korrekte Tests (in %)	inkorrekte Tests (in %)	ergebnislose Tests (in %)
Chow-Robbins	∅N = 100	$N_{min} = 100$,$N_{max} = 101$	64%	0%	36%
Gauss-CI	∅N = 132	$N_{min} =$,$N_{max} = 132$	68%	0%	32%
Azuma	∅N = 2010	$N_{min} = 687$,$N_{max} = 3609$	100%	0%	0%

Tabelle 6.13: Ergebnisse der Testdurchläufe für $\mathbf{X} = 0{,}1$ mit $guess = 0{,}01$

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	korrekte Tests (in %)	inkorrekte Tests (in %)	ergebnislose Tests (in %)
Chow-Robbins	∅N = 4970	$N_{min} = 4286$, $N_{max} = 5660$	100%	0%	0%
Gauss-CI	∅N = 10162	$N_{min} = N_{max} = 10162$	100%	0%	0%
Azuma	∅N = 3949	$N_{min} = 3371$, $N_{max} = 4797$	100%	0%	0%

An der ersten Tabelle ist direkt ersichtlich, dass $guess = 0{,}1$ in diesem Fall unzureichend präzise ist (in Bezug auf $|p - p_0|$), da die Ergebnisqualität nicht zufriedenstellend ist. Lediglich der *Azuma*-Test kann sein gewohntes Niveau aufrecht erhalten. Stellt man dagegen den Parameter auf 0,01 um, zu sehen in der zweiten Tabelle, so lassen sich deutlich bessere Ergebnisse erzielen, mit dem zu erwartenden Anstieg der benötigten Stichprobenanzahl.

Tabelle 6.14: Ergebnisse der Testdurchläufe für $\mathbf{X} = 0{,}06$

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	korrekte Tests (in %)	inkorrekte Tests (in %)	ergebnislose Tests (in %)
Chow-Robbins	⌀N = 108	$N_{min} = 104$,$N_{max} = 112$	3%	0%	97%
Gauss-CI	⌀N = 100	$N_{min} = N_{max} = 100$	11%	0%	89%
Azuma	⌀N = /	$N_{min} = /$,$N_{max} = /$	/	/	/

Tabelle 6.15: Ergebnisse der Testdurchläufe für $\mathbf{X} = 0{,}06$ für *guess* = 0,01

	Durchschnittliche Stichprobengröße	Min/Max Stichprobengröße	korrekte Tests (in %)	inkorrekte Tests (in %)	ergebnislose Tests (in %)
Chow-Robbins	⌀N = 5122	$N_{min} = 4670$, $N_{max} = 5515$	100%	0%	0%
Gauss-CI	⌀N = 6566	$N_{min} = N_{max} = 6566$	100%	0%	89%
Azuma	⌀N = 31014	$N_{min} = 23278$, $N_{max} = 35313$	100%	0%	0%

Für den Fall $\mathbf{X} = 0{,}06$ mit *guess* = 0,1, fallen die Ergebnisse äußerst schlecht aus. So endet der überwiegende Teil der *Chow-Robbins* und *Gauss-CI* Testdurchläufe ergebnislos, während der *Azuma*-Test gar kein Ergebnis innerhalb von 100000 gezogenen Stichproben berechnen konnte. Auch hier sollte der Parameter *guess* auf einen niedrigeren und damit präziseren Wert gesetzt werden, um zufriedenstellende Testqualitäten zu erzielen, was an der zweiten Tabelle zu sehen ist, da jeder der Algorithmen für *guess* = 0,01 ausschließlich korrekte Ergebnisse präsentiert.

6.4 Allgemeine Analyse der Testdurchläufe

Die durchgeführten Tests zeigen, dass sich die Erwartungen aus 6.1 bestätigt haben. Zum Schluss sollen die Ergebnisse der einzelnen Test aufbereitet und ein allgemeines Fazit gezogen werden. Es konnte gezeigt werden, dass der *Chow-Robbins*-Test bessere Effizienzwerte aufweist, je weiter p von 0,5 entfernt liegt. Zudem lag die Varianz der Stichprobengröße bei allen durchgeführten Tests in einem überaus akzeptablen Bereich. Allerdings fällt die Ergebnisqualität für kleine $|p - p_0|$ relativ deutlich ab. Es lässt sich also sagen, dass man bei der Wahl dieses Hypothesentests deutlich von Prognosen bezüglich der Lage von p profitieren kann. Wenn bekannt ist, dass p relativ nahe bei 0 oder 1 liegt, kann man wie bereits erwähnt eine geringe Stichprobengröße erwarten. Wenn auch bekannt ist, dass $|p - p_0|$ angemessen groß ausfällt, kann man zudem von einer hohen Korrektheit ausgehen und durch einen größeren Wert für *guess* die Effizienz noch weiter verbessern, ohne diesen Grad an Korrektheit ausschlaggebend zu mindern.

Dadurch dass der *Gauss-CI*-Test mit einer festen Stichprobengröße arbeitet, hat diese Größe zwar keine Varianz, jedoch verbessert sich dessen Effizienz auch nicht durch eindeutig ausfallende Simulationsdaten (wie es zum Beispiel bei rein sequentiellen Tests wie

dem *SPRT*-Algorithmus zu erwarten wäre). Allerdings hat man hier den größten Einfluss auf auf die Anzahl der benötigten Stichproben, da diese nicht nur von dem Parameter *guess* sondern auch maßgeblich vom, ebenfalls vom Nutzer einzugebenden Parameter p_0 abhängt. Da der Tests nicht durch den unbekannten Wert von p beeinflusst wird, hat man zwar durch eventuell vorhandene Prognosen zur Testmodelleruierung nur einen geringeren Vorteil als beim *Chow-Robbins*-Test, allerdings auch nur einen geringen Nachteil, wenn das Aufstellen von Prognosen nur bedingt oder praktisch gar nicht möglich ist. Der Test eignet sich dementsprechend vor allem dann sehr gut, wenn p_0 sehr groß oder sehr klein gewählt ist.

Für beide Algorithmen gilt, dass sich mit ihnen sehr zuverlässige Hypothesentests durchführen lassen. Dies lässt sich damit begründet, dass die Gefahr ein falsches Ergebnis zu berechnen, bei beiden Test sehr minimal ausfällt und dass die Wahl des Parameters *guess*, mit dem sowohl Effizienz, als auch Ergebnisqualität direkt und stark beeinflusst werden können, vom Nutzer justierbar ist. Wenn also ein Ergebnis berechnet werden kann, so ist dies mit großer Wahrscheinlichkeit auch korrekt. Sollte ein Testdurchlauf hingegen ergebnislos enden, so kann über *guess* die Präzision des Tests deutlich aufgebessert und wahrscheinlich auch ein (korrektes) Ergebnis berechnet werden.
Der *Azuma*-Algorithmus kann durch seine hohe Ergebnisqualität punkten, allerdings weist er auch eine zum Teil extreme Varianz der Stichprobengröße auf. In den Testfällen konnte sich der Test leider nur selten wirklich hervorheben, da sich die Ergebnisqualität meist zusammen mit den anderen Test verschlechterte (in dem Sinne, dass keine Ergebnisse errechnet werden konnten). Dies macht den *Azuma*-Algorithmus zumindest im direkten Vergleich zu einem eher unzuverlässigen Test, da es sich in der Praxis eher lohnt, die Risiken der anderen Tests in Erwartung höherer Effizienz in Kauf zu nehmen.

7 Schlussbemerkungen

In dieser Arbeit wurden drei Hypothesentestalgorithmen für ein Tool zur Simulation von hybriden Petri-Netzen und der Durchführung von Model Checks implementiert. Im Rahmen der Testdurchläufe wurde gezeigt, inwieweit die zuvor dargelegten Vor- und Nachteile der einzelnen Algorithmen für verschiedene Modellbedingungen genutzt werden können und unter welchen Bedingungen welcher Testalgorithmus vorzuziehen ist. Zudem wurde konnte gezeigt werden, welchen Einfluss der variable Parameter *guess* auf die Testperformance hat und wie dieses signifikante Maß an zusätzlicher Kontrolle bei der Durchführen der Hypothesentests für zufriedenstellende Resultate genutzt werden kann.

Auch die Änderungen am Programmcode, welche für die Implementierungen der Testalgorithmen vorgenommen wurden, weisen einige Vorteile im Vergleich zu vorher auf. So erleichtert die für die Einbettung der Algorithmen implementierte Codestruktur das Hinzufügen weiterer Tests. Außerdem wurde das Tool um eine Reihe zusätzlicher Funktion im Bereich der Hypothesentestdurchführung ergänzt, welche dem Nutzer das Anpassen bestimmter Testparameter erlaubt und im Allgemeinen eine komfortable Bedienstruktur schafft. Zudem wurde die Ausgabe der Ergebnisse der Hypothesentestdurchläufe erweitert, wodurch eine detailliertere Analyse dieser Resultate in Bezug auf verschiedene Aspekte möglich ist.

In zukünftigen Arbeiten wäre die Erweiterung des Tools durch weitere Hypothesentestalgorithmen mit einer entsprechenden Performance-Evaluation und Vergleich zu den in dieser Arbeit implementierten Algorithmen sicherlich von Interesse. Da der Kontext der Modellierung mit Petri-Netzen, des (statistischen) Model Checkings und der damit verbundenen Durchführung von Hypothesentests die Systemanalyse ist, könnten zudem weitere Analysefunktionen in das Tool integriert werden. So könnte zum Beispiel eine grafische Repräsentation der in 4.1 beschriebenen vier Bereiche (U, L, I und NC) und des Wachstums der Teststatistik (Z_N) einen detaillierteren Einblick in den Ablauf der Hypothesentest sowie in die Ergebnisse der Simulation, auf der der Hypothesentest basiert, gewähren.

Literatur

[1] Joaquin Ezpeleta, Jose Manuel Colom und Javier Martinez. „A Petri net based deadlock prevention policy for flexible manufacturing systems“. In: *IEEE transactions on robotics and automation* 11.2 (1995), S. 173–184.

[2] Hamed Ghasemieh, Anne Remke und Boudewijn R Haverkort. „Survivability evaluation of fluid critical infrastructures using hybrid Petri nets“. In: *Dependable Computing (PRDC), 2013 IEEE 19th Pacific Rim International Symposium on*. IEEE. 2013, S. 152–161.

[3] Marco Gribaudo und Anne Remke. „Hybrid Petri nets with general one-shot transitions“. In: *Performance Evaluation* 105 (2016), S. 22–50.

[4] Boudewijn R. Haverkort. *Performance of computer communication systems: a model-based approach*. Wiley, 1998, S. 409–418.

[5] Axel Legay, Benoît Delahaye und Saddek Bensalem. „Statistical model checking: An overview“. In: *International Conference on Runtime Verification*. Springer. 2010, S. 122–135.

[6] Stephan Merz. „Model checking: A tutorial overview“. In: *Modeling and verification of parallel processes*. Springer, 2001, S. 3–38.

[7] Tadao Murata. „Petri nets: Properties, analysis and applications“. In: *Proceedings of the IEEE* 77.4 (1989), S. 541–542.

[8] Carina Pilch. „Development of an event-based simulator for model checking hybrid Petri nets with random variables“. Magisterarb. Computer Science Department, University of Münster, 2016. In Vorbereitung.

[9] Daniël Reijsbergen, Werner Scheinhardt und Pieter-Tjerk de Boer. „A sequential hypothesis test based on a generalized azuma inequality“. In: *Statistics & probability letters* 97 (2015), S. 192–196.

[10] Daniël Reijsbergen u. a. „On hypothesis testing for statistical model checking“. In: *International journal on software tools for technology transfer* 17.4 (2015), S. 377–395.

[11] Wil MP Van Der Aalst. „Workflow verification: Finding control-flow errors using petri-net-based techniques“. In: *Business Process Management*. Springer, 2000, S. 161–183.

Plagiatserklärung der / des Studierenden

Hiermit versichere ich, dass die vorliegende Arbeit über _____ _____ selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommenen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

_____
(Datum, Unterschrift)

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

_____
(Datum, Unterschrift)