Westfälische
Wilhelms-Universität
Münster

CSL Model Checking von QBDs in PRISM

Bachelorarbeit
zur Erlangung des akademischen Grades
Bachelor of Science

Westfälische Wilhelms-Universität Münster
Fachbereich Mathematik und Informatik
Institut für Informatik

Betreuerin/Erstgutachterin:
Prof. Dr. Anne Remke

Zweitgutachter:
Prof. Dr. Markus Müller-Olm

Eingereicht von:
Stefanie Eva Drerup

Münster, Mai 2016

Zusammenfassung

In dieser Arbeit werden zwei verschiedene Lösungsstrategien für das Model Checking von sogenannten Quasi-Birth-Death-Processes *(Abkürzung: QBD)* vorgestellt. QBDs bilden eine spezielle Klasse der unendlichen kontinuierlichen Markov-Ketten, weshalb wir zu prüfende Eigenschaften mithilfe einer nur leicht modifizierten Version der Continuous Stochastic Logic formulieren können. Für die atomaren Eigenschaften werden wir ein neues Prinzip der Levelunabhängigkeit fordern, um die QBDs für das Model Checking greifbarer zu machen.
Es wird zum einen eine neue Methode für das Berechnen von transienten Zustandswahrscheinlichkeiten, die Uniformisierung mit Repräsentanten für QBDs, nach Remke et al. [Rem+07], vorgestellt und zum anderen eine eigene Approximation des QBD geschildert. Während bei der vorgestellten Methode ein unendlicher QBD betrachtet wird und nur die Matrixdarstellung endlich ist, so schneidet man bei der Approximation tatsächlich den QBD ab einem gewissen Level ab und macht ihn dadurch endlich.
Wir werden die Effizienz und Anwendbarkeit beider Verfahren diskutieren. Spezielle Beachtung findet dabei ein dynamisches Abbruchkriterium, welches durch die Uniformisierung mit Repräsentanten möglich gemacht wird. Anhand des wahrscheinlichkeitstheoretischen Model Checkers PRISM werden wir Probleme, die sich durch die Approximation des Abschneidens ergeben, ermitteln und einen Lösungsansatz entwickeln. Dieser wird dann in Hinblick auf Laufzeit und Iterationsanzahl mit den Ergebnissen nach Remke et al. [Rem+07] verglichen.

Inhaltsverzeichnis

1. Einleitung

In der heutigen Zeit ist Software in unserem Alltag allgegenwärtig. Eingebettete Systeme sind wichtig für die Steuerung oder Überwachung gewisser Abläufe wie zum Beispiel in medizinischen Geräten, in Flugzeugen oder in Mobilfunknetzgeräten. Darüber hinaus finden auch Kommunikationsprotokolle und Transportsysteme viele Anwendungen. Die dafür entwickelten Softwaresysteme werden jedoch immer größer, komplexer und vor allem fehleranfälliger. Gerade bei sicherheitskritischen Systemen sind aber auch kleine Fehler mit schweren Konsequenzen verbunden. Deshalb ist es nötig, diese Systeme zu testen und zu verifizieren, was oftmals mit hohen Kosten verbunden ist.
Mit Quantitativem Model Checking existiert jedoch ein sehr etabliertes Verfahren zur automatischen Verifikation von abstrakten Modellen. Dabei wird die Leistung und Zuverlässigkeit eines Systems nicht nur qualitativ, sondern vor allem auch quantitativ bewertet. Zur Modellierung der Systeme finden bereits endliche diskrete oder kontinuierliche Markov-Ketten ihre Anwendung.
In vielen Fällen ist es aber notwendig, Systeme auch als unendliche Modelle darstellen zu können. Als Beispiele seien hier Modelle mit unendlich großen Speichern, mit Variablen oder einfach extrem großskalige Systeme genannt. Die bisherigen Model Checking-Algorithmen sind jedoch auf endliche Modelle ausgelegt. Daher müssen entweder neue Approximationen oder neue Model-Checking-Algorithmen für unendliche Modelle entwickelt werden.
Wir möchten dabei nicht beliebige unendliche Modelle betrachten, sondern konkret Quasi-Birth-Death-Processes untersuchen, welche eine spezielle Klasse der unendlichen kontinuierlichen Markov-Ketten darstellen. Ihre besondere Struktur ermöglicht es, etablierte Methoden wie die Uniformisierung wiederzuverwenden, um neue Model-Checking-Algorithmen zu entwickeln. Zur Formalisierung von quantitativen Eigenschaften in QBDs werden wir die Continuous Stochastic Logic verwenden. Für das konkrete Model Checking nutzen wir das Software-Tool PRISM. Dieses unterstützt nur endliche Modelle, sodass wir in dieser Arbeit eine endliche Approximation eines QBDs finden müssen, um trotzdem Model Checking von unendlichen Modellen simulieren zu können.
Die Arbeit ist wie folgt aufgebaut: Wir werden zunächst einige Grundlagen einführen und erläutern, was kontinuierliche Markov-Ketten sind und insbesondere, welche Unterschiede zwischen den allgemeinen unendlichen kontinuierlichen Markov-Ketten und QBDs liegen. Danach adaptieren wir die Continuous Stochastic Logic für unsere QBDs und erläutern die Levelunabhängigkeit, was eine grundlegende Voraussetzung für unser weiteres Vorgehen ist.
Im folgenden Kapitel werden wir dann schildern, wie die bereits etablierte Uni-

formisierung funktioniert und warum sie in ihrer Originalform nicht auf QBDs anwendbar ist. Dennoch können wir ihre Grundidee verwenden und werden, nach der Erarbeitung von einer endlichen Darstellung des unendlichen QBDs, eine speziell auf QBDs zugeschnittene Uniformisierung mit Repräsentanten vorstellen. Diese Methode bietet eine effiziente Möglichkeit zur Berechnung von transienten Zustandswahrscheinlichkeiten und führt zur Entwicklung eines dynamischen Abbruchkriterium, welches zuverlässige Aussagen über die Gültigkeit einer CSL-Formel möglich macht.
Anhand eines Anwendungsbeispiels werden wir dann eine eigene simplere Approximation eines QBDs, bei der der QBD auf ein endliches Modell reduziert wird, in PRISM vorstellen. Wir analysieren diese in Bezug auf Wahrscheinlichkeitsgenauigkeiten und verschiedene Zeitpunkte und arbeiten als Probleme heraus, dass für unterschiedlich lange Approximationen eines QBD verschiedene Aussagen über die Gültigkeit einer CSL-Formel getroffen werden können. Als Lösungsansatz stellen wir ein Konvergenzkriterium vor und implementieren dieses in Python. Als letzter Teil der Analyse erfolgt ein Testen des Konvergenzkriterium-Skripts und wir vergleichen unsere Approximation mit der Uniformisierung durch Repräsentanten nach Remke et al. [Rem+07] in Bezug auf Laufzeit und Iterationsanzahl. Abgeschlossen wird diese Arbeit von einem Fazit, in dem wir unsere erarbeiteten Erkenntnisse kurz zusammenfassen und diskutieren. Zudem geben wir einen Ausblick, welche Lösungsstrategien es außer den in dieser Arbeit vorgestellten Methoden gibt.

2. Theoretische Grundlagen

In dieser Arbeit werden Modelle behandelt, die die Struktur eines **Q**uasi-**B**irth-**D**eath-Process (Abkürzung: QBD) besitzen. Ein QBD ist eine unendliche kontinuierliche Markov-Kette (Abkürzung: CTMC für **C**ontinous-**T**ime **M**arkov **C**hain) mit einer speziellen Struktur. Um QBDs zu definieren, führen wir zunächst CTMCs ein und gehen danach auf die Besonderheiten von QBDs ein.
Außerdem untersuchen wir später QBDs auf bestimmte Eigenschaften und müssen dafür Eigenschaften formalisieren. Das geschieht mithilfe der **C**ontinuous **S**tochastic **L**ogic (Abkürzung: CSL), dessen Operatoren im zweiten Abschnitt vorgestellt werden.
Zuletzt führen wir das Prinzip der Levelunabhängigkeit ein und erläutern dessen Zusammenhang mit CSL-Formeln.
Insgesamt beziehen wir uns in diesem Kapitel auf die Definitionen nach Baier et al. [Bai+03], Kwiatkowska et al. [Kwi+07] und insbesondere auf Remke et al. [Rem+07].

2.1. Continuous-Time Markov Chain

Eine CTMC besteht im Allgemeinen aus einer Menge von Zuständen S und Transitionen zwischen diesen Zuständen, welche mit positiven Raten versehen sind. Diese werden in einer Ratenmatrix $\mathbf{R}$ festgehalten, wobei die Rate für die Transition von s zu s' im Eintrag $\mathbf{R}_{s,s'}$ der Matrix gespeichert ist. Um den Zuständen zusätzlich Eigenschaften zuweisen zu können, gibt es eine Menge AP von atomaren Eigenschaften *(atomic properties)*. Die Zuordnung dieser Eigenschaften zu den Zuständen geschieht durch eine Labelling-Funktion. Formell ist eine CTMC damit wie folgt definiert:

Definition 1 (Continuous-Time Markov Chain)**.** *Eine Continuous-Time Markov Chain $\mathcal{C}$ ist ein Tupel (S,$\boldsymbol{R}$,L), wobei*

- *S eine endliche Menge von Zuständen,*
- $\boldsymbol{R} : S \times S \to \mathbb{R}_{\geq 0}$ *eine Ratenmatrix,*
- $\mathcal{L} : S \to 2^{AP}$ *eine Labelling-Funktion, die jedem Zustand $s \in S$ seine gültigen Eigenschaften aus AP zuweist,*

ist.

Eine grafische Darstellung einer Beispiel-CTMC folgt am Ende dieses Abschnitts in Abbildung 2.1. Nach der formellen Definition einer CTMC ist es

zulässig, dass eine Rate den Wert 0 hat. Dies ist genau dann der Fall, wenn keine Transition zwischen den Zuständen s und s' existiert. Entsprechend bedeutet $\mathbf{R}(s, s') > 0$, dass eine Transition von s zu s' vorhanden ist.
Die Wahrscheinlichkeit, dass diese Transition innerhalb der nächsten t Zeiteinheiten gewählt wird, beträgt $1 - e^{-\mathbf{R}(s,s')\cdot t}$. Dies ist darauf zurückzuführen, dass jeder Zustand $s \in S$ eine zu einem Parameter $q_s \in \mathbb{R}^+$ exponentialverteilte Zufallsvariable besitzt. Der Parameter q_s lässt sich durch die durchschnittliche Verweilzeit *(mean residence time)* eines Zustands definieren, da diese $\frac{1}{q_s}$ ist. Abhängig von q_s und der Wahrscheinlichkeit eine Transition von s zu s' zu wählen, entsteht die Rate $\mathbf{R}(s, s')$ einer CTMC.
In den meisten Fällen gilt jedoch $\mathbf{R}(s, s') > 0$ für mehr als einen Zustand s', sodass ein Wettbewerb zwischen den Transitionen entsteht. Wir definieren deshalb den Begriff der Ausgangsrate *(exit rate)*.

Definition 2 (Ausgangsrate)**.** *Sei $s \in S$. Die Ausgangsrate von $s \in S$ ist definiert als*

$$E(s) = \sum_{s' \in S} \boldsymbol{R}(s, s').$$

Die Wahrscheinlichkeit im Zustand s als nächstes innerhalb von t Zeiteinheiten die Transition zu s' zu wählen, beträgt dann

$$\mathcal{V}(s, s', t) = \frac{\mathbf{R}(s, s')}{E(s)} \cdot (1 - e^{-E(s)\cdot t}).$$

Außerdem entspricht E(s) in diesem Zusammenhang genau dem Parameter q_s der exponentialverteilten Zufallsvariable des Zustands s, das heißt die durchschnittliche Verweilzeit beträgt $\frac{1}{E(s)}$. Falls $E(s) = 0$ gilt, so wird der Zustand s *absorbierend* genannt und die durchschnittliche Verweilzeit beträgt per Definition ∞.
Wir möchten an dieser Stelle nicht weiter ins Detail gehen, welche Wahrscheinlichkeiten außerdem berechenbar sind, sondern werden dies insbesondere nach der Definition von QBDs in Abschnitt 2.2 dieser Arbeit fortführen.
In Hinblick auf die Analyse einer CTMC benötigen wir jedoch noch eine weitere Definition. Deshalb führen wir den Begriff der Generatormatrix $\mathbf{Q}$ ein.

Definition 3 (Generatormatrix)**.** *Sei $\mathcal{C}$ eine CTMC und $\boldsymbol{R}$ die zugehörige Ratenmatrix. Die Generatormatrix $\boldsymbol{Q} : S \times S \to \mathbb{R}$ ist definiert als:*

$$\boldsymbol{Q}(s, s') = \begin{cases} \boldsymbol{R}(s, s'), & \text{wenn } s \neq s' \\ -E(s), & \text{sonst} \end{cases}$$

Weshalb diese Generatormatrix $\mathbf{Q}$ nötig ist, soll hier aufgrund des Umfangs der Arbeit nicht detailliert aufgeführt werden. Es sei nur erwähnt, dass sich $\mathbf{Q}$ aufgrund ihrer im Gegensatz zu $\mathbf{R}$ etwas anderen Form besser für die Anwendung von Model Checking-Algorithmen in Kapitel 3 eignet. Deshalb arbeiten wir im Folgenden auch oftmals mit $\mathbf{Q}$ anstelle von $\mathbf{R}$.

Zum besseren Verständnis möchten wir nun abschließend ein Beispiel einer CTMC mit deren Ausgangsraten und der Generatormatrix $\mathbf{Q}$ betrachten. Eine grafische Darstellung unseres Beispiels findet sich in Abbildung 2.1.

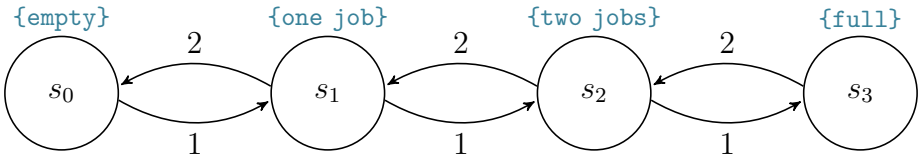


Abbildung 2.1.: Warteschlange für Jobs als CTMC modelliert

Wir betrachten ein Warteschlangen-Modell, welches sogenannte Jobs speichert. In der Anwendung könnten diese zum Beispiel die Aufträge in einer Prozessor-Warteschlange sein. Unsere Zustandsmenge ist $S = \{s_0, s_1, s_2, s_3\}$. Außerdem ist die Kapazität der Warteschlange endlich und umfasst maximal drei Jobs. Wir unterscheiden deshalb die Anzahl der Jobs in der Warteschlange durch entsprechende atomare Eigenschaften, konkret $AP = \{$empty, one job, two jobs, full$\}$.
Die Ratenmatrix $\mathbf{R}$ können wir an der grafischen Darstellung ablesen. Wir haben außerdem die Ausgangsrate für jeden Zustand $s \in S$ berechnet und daraus resultierend die Generatormatrix $\mathbf{Q}$ aufgestellt:

$$\mathbf{R} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 2 & 0 \end{pmatrix} \Rightarrow \begin{array}{c} \mathrm{E}(s_0) = 1 \\ \mathrm{E}(s_1) = 2 + 1 = 3 \\ \mathrm{E}(s_2) = 2 + 1 = 3 \\ \mathrm{E}(s_3) = 2 \end{array} \Rightarrow \mathbf{Q} = \begin{pmatrix} -1 & 1 & 0 & 0 \\ 2 & -3 & 1 & 0 \\ 0 & 2 & -3 & 1 \\ 0 & 0 & 2 & -2 \end{pmatrix}$$

Angenommen man befindet sich aktuell im Zustand s_1. Wenn man nun die Wahrscheinlichkeit berechnen möchte zum Zustand s_2 innerhalb der nächsten drei Zeiteinheiten zu wechseln, so erhält man

$$\mathcal{V}(s_1, s_2, 3) = \frac{\mathbf{R}(s_1, s_2)}{E(s_1)} \cdot (1 - e^{-E(s_1)\cdot 3}) = \frac{1}{3} \cdot (1 - e^{-3\cdot 3}) \approx 0.33.$$

Damit haben wir einen ersten Eindruck von CTMCs gewonnen. Im nächsten Abschnitt dieses Kapitels setzen wir uns mit QBDs auseinander.

2.2. Quasi-Birth-Death-Process

Ein QBD $\mathcal{Q}$ ist ein Spezialfall einer CTMC. Bisher haben wir angenommen, dass CTMCs endlich sind. Ein QBD ist jedoch unendlich, aber folgt trotzdem einer bestimmten Struktur. Insofern bilden QBDs nur eine Teilmenge der unendlichen CTMCs. Wir definieren konkret einen QBD der Ordnung (N_0, N).

Er besitzt die Form eines Bands von unendlicher Länge, welches dennoch eine endliche Breite besitzt. Dabei werden die Zustände des QBD in sogenannten *Leveln* zusammengefasst. Eine Übersicht darüber gibt Abbildung 2.2.

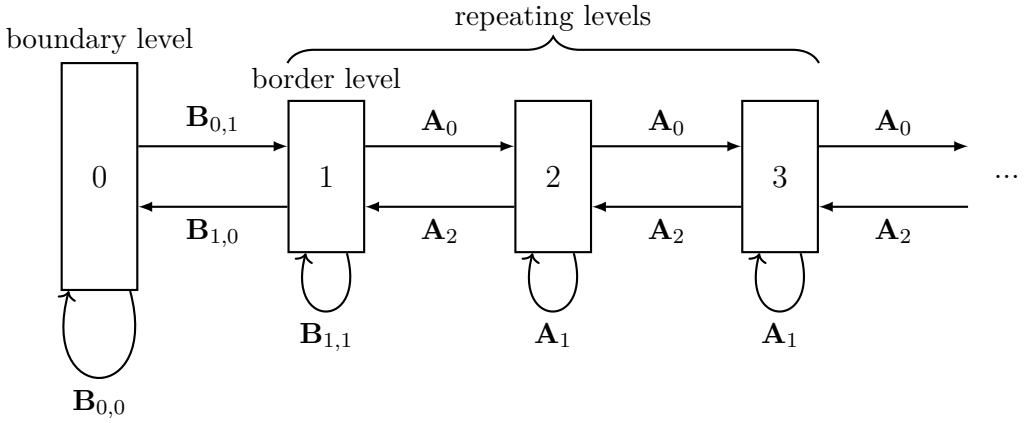


Abbildung 2.2.: Struktur eines QBD nach Remke et al. [Rem+07]

Den Anfang des QBD bildet das *Grundlevel (boundary level)*, dessen Breite durch N_0 angegeben ist. Es folgen unendlich viele *Wiederholungslevel (repeating levels)*, die alle dieselbe Zustandsstruktur der Breite N besitzen. Die Zustandsmenge S hat also die Form

$$S = \{(i,j) \mid \underbrace{(i \in \{0,...,N_0-1\} \wedge j = 0)}_{Grundlevel} \vee \underbrace{(i \in \{0,...,N-1\} \wedge j \in \mathbb{N}^+)}_{Wiederholungslevel}\}.$$

Offensichtlich ist sie unendlich groß. Das erste Wiederholungslevel stellt dabei einen Sonderfall dar und wird auch als *Grenzlevel (border level)* bezeichnet, da zwischen ihm und dem Grundlevel andere Transitionen vorhanden sein können als zwischen zwei Wiederholungsleveln.
Die Transitionen von $\mathcal{Q}$ werden als Matrizen in einer Block-Generatormatrix $\mathbf{Q}$ dargestellt. Jede Blockmatrix stellt dabei entweder Transitionen innerhalb eines Levels oder Transitionen zwischen genau zwei Leveln dar. Durch die endliche Breite eines QBD ergibt sich, dass die Blockmatrizen ebenfalls endlich sind. Aufgrund der Eigenschaft von $\mathcal{Q}$, dass nur Transitionen zwischen benachbarten Leveln vorhanden sind, ist diese Matrix $\mathbf{Q}$ tridiagonal in Bezug auf seine Blockmatrizen, siehe Abbildung 2.3.
Die Transitionen der jeweiligen Blockmatrizen sind im Folgenden kurz aufgeführt:

$\mathbf{B}_{0,0} \in \mathbb{R}^{N_0 \mathrm{x} N_0}$	Transitionen innerhalb des Grundlevels
$\mathbf{B}_{0,1} \in \mathbb{R}^{N_0 \mathrm{x} N}$	Transitionen vom Grund- zum Grenzlevel
$\mathbf{B}_{1,0} \in \mathbb{R}^{N \mathrm{x} N_0}$	Transitionen vom Grenz- zum Grundlevel
$\mathbf{B}_{1,1} \in \mathbb{R}^{N \mathrm{x} N}$	Transitionen innerhalb des Grenzlevels
$\mathbf{A}_0 \in \mathbb{R}^{N \mathrm{x} N}$	Transitionen von einem Wiederholungslevel zum

$$\mathbf{Q} = \begin{pmatrix} \mathbf{B}_{0,0} & \mathbf{B}_{0,1} & & & \\ \mathbf{B}_{1,0} & \mathbf{B}_{1,1} & \mathbf{A}_0 & & \\ & \mathbf{A}_2 & \mathbf{A}_1 & \mathbf{A}_0 & \\ & & \mathbf{A}_2 & \mathbf{A}_1 & \mathbf{A}_0 \\ & & \ddots & \ddots & \ddots \end{pmatrix}$$

Abbildung 2.3.: Generatormatrix $\mathbf{Q}$ nach Remke et al. [Rem+07]

	nächsthöheren Wiederholungslevel
$\mathbf{A}_1 \in \mathbb{R}^{N \text{x} N}$	Transitionen innerhalb eines Wiederholungslevel
$\mathbf{A}_2 \in \mathbb{R}^{N \text{x} N}$	Transitionen von einem Wiederholungslevel zum nächstniedrigeren Wiederholungslevel

Um für die Zustände des QBD Eigenschaften zu formulieren, wird genauso wie bereits für allgemeine CTMCs eine Labelling-Funktion $\mathcal{L}$ definiert:

$$\mathcal{L} : S \to 2^{AP}$$

Wir benötigen außerdem erst die Definition eines Pfads, um auch über diesen Eigenschaften formulieren zu können.

Definition 4 (Pfad)**.** *Sei $i \in \mathbb{N}$, seien $s_i \in S$ Zustände und $t_i \in \mathbb{R}^+$ Transitionen mit $\boldsymbol{Q}(s_i, s_{i+1}) > 0 \ \forall \ i$. Ein unendlicher Pfad σ ist eine Sequenz $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} \ldots$.*
Ein endlicher Pfad σ der Länge l+1 ist eine Sequenz $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \ldots s_{l-1} \xrightarrow{t_{l-1}} s_l$ mit $\boldsymbol{Q}(s_i, s_{i+1}) > 0 \ \forall \ i < l$ und $Q(s_l, s') = 0 \ \forall \ s' \in S$ (s_l ist absorbierend).
Dabei bezeichnet $\sigma[i] = s_i$ den (i+1)ten Zustand des Pfads σ. Für endliche Pfade gilt $\sigma[l] = \infty \ \forall \ i \geq l$.

Notation: $Path^{\mathcal{Q}}$ ist die Menge aller Pfade von $\mathcal{Q}$. $Path^{\mathcal{Q}}(s)$ ist die Menge aller Pfade von $\mathcal{Q}$, die im Zustand s beginnen.

Uns interessiert insbesondere der Zusammenhang zwischen Pfaden und Zeitpunkten, daher definieren wir, was wir unter der Verweilzeit verstehen und in welchem Zustand sich ein Pfad zu einem Zeitpunkt $t \in \mathbb{R}$ befindet.

Definition 5 (Verweilzeit)**.** *Sei $i \in \mathbb{N}$ und $\sigma \in Path^{\mathcal{Q}}$ ein Pfad von $\mathcal{Q}$. Die Verweilzeit im Zustand s_i ist definiert durch $\delta(\sigma, i) = t_i$. Falls σ endlich ist, so gilt $\delta(\sigma, l) = \infty \ \forall \ i \geq l$.*

Definition 6 (Zustand zum Zeitpunkt t)**.** *Sei $t \in \mathbb{R}, i \in \mathbb{N}$ der minimale Index mit $t \leq \sum_{j=0}^{i} t_j$, dann definieren wir als $\sigma@t = \sigma[i]$ den Zustand, in dem sich der Pfad σ zum Zeitpunkt t befindet.*
Für endliche Pfade ist $\delta@t = s_l \; \forall \; t > \sum_{j=0}^{l-1} t_j$.

In einem QBD benötigen wir auch ein Wahrscheinlichkeitsmaß. Wie in allgemeinen CTMCs besitzt jeder Zustand eine exponentialverteilte Zufallsvariable mit einem Parameter, der etwas über die durchschnittliche Verweilzeit des Zustands aussagt. Wir möchten an dieser Stelle aber nicht konkret die Wahrscheinlichkeiten von einzelnen Transitionen betrachten, wie wir es beispielhaft für CTMCs getan haben. Deshalb definieren wir nun die *transiente Zustandswahrscheinlichkeit* und die *stationäre Wahrscheinlichkeit*. Bei ersterer möchten wir eine Aussage über den QBD zu einem bestimmten Zeitpunkt treffen, bei letzterer über das langfristige Verhalten des QBD. Selbstverständlich sind diese Wahrscheinlichkeiten keine Besonderheit eines QBD, sondern können auch für allgemeine CTMCs definiert werden.

Definition 7 (transiente Zustandswahrscheinlichkeit)**.** *Die Wahrscheinlichkeit, sich im Zustand s' zum Zeitpunkt t zu befinden, wenn wir im Zustand s gestartet sind, bezeichnen wir als*

$$\mathcal{V}^{\mathcal{Q}}(s, s', t) = Pr\{\sigma \in Path^{\mathcal{Q}} \mid \sigma@0 = s \wedge \sigma@t = s'\}.$$

Definition 8 (stationäre Wahrscheinlichkeit)**.** *Die Wahrscheinlichkeit, sich langfristig im Zustand s' zu befinden, wenn wir im Zustand s gestartet sind, bezeichnen wir als*

$$\pi^{\mathcal{Q}}(s, s') = \lim_{t \to \infty} \mathcal{V}^{\mathcal{Q}}(s, s', t).$$

Inwiefern wir diese Wahrscheinlichkeiten für einen QBD berechnen können, wird in einem späteren Kapitel näher erläutert.
Wenn wir uns an das Beispiel aus 2.1 erinnern, so war dort die Kapazität der Warteschlange endlich. Wenn wir jedoch die Kapazität nicht beschränken, so erhalten wir eine unendliche Warteschlange. In diesem Fall betrachten wir dann den Spezialfall eines QBD. Dieser ist grafisch in Abbildung 2.4 dargestellt.

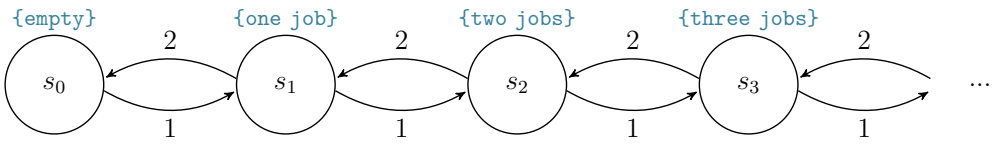


Abbildung 2.4.: Warteschlange für Jobs als QBD modelliert

Entsprechend ist die Zustandsmenge nun $S = \{s_0, s_1, s_2, s_3, ...\}$. Außerdem hat die Generatormatrix $\mathbf{Q}$ unendliche Größe, wobei die Blockmatrizen in diesem Fall nur aus einem Eintrag bestehen, im Detail:

$$\mathbf{Q} = \begin{pmatrix} -1 & 1 & & & \\ 2 & -3 & 1 & & \\ & 2 & -3 & 1 & \\ & & \ddots & \ddots & \ddots \end{pmatrix} \quad \mathbf{B}_{0,0} = \begin{pmatrix} -1 \end{pmatrix} \quad \mathbf{B}_{0,1} = \begin{pmatrix} 1 \end{pmatrix} \quad \mathbf{B}_{1,0} = \begin{pmatrix} 2 \end{pmatrix} \quad \mathbf{B}_{1,1} = \begin{pmatrix} -3 \end{pmatrix} \quad \mathbf{A}_0 = \begin{pmatrix} 1 \end{pmatrix} \quad \mathbf{A}_1 = \begin{pmatrix} -3 \end{pmatrix} \quad \mathbf{A}_2 = \begin{pmatrix} 2 \end{pmatrix}$$

Offensichtlich wiederholen sich die Matrizen $\mathbf{A}_0, \mathbf{A}_1$ und $\mathbf{A}_2$ ab dem Grenzlevel fortlaufend.
Nachdem wir nun damit auch ein Beispiel für einen QBD gesehen haben, möchten wir im Folgenden auch Eigenschaften über unseren QBD formulieren können. Deshalb führen wir im nächsten Kapitelteil die *Continuous Stochastic Logic* für QBDs ein.

2.3. Continuous Stochastic Logic

Die Notation der CSL Operatoren für QBDs ist dieselbe wie die für allgemeine CTMCs, vgl. Baier et al. [Bai+03]. Wenn wir jedoch prüfen, ob eine CSL-Formel erfüllt ist, so müssen wir dabei die spezielle Struktur des QBD berücksichtigen.

Definition 9 (CSL Zustands- und Pfadformeln)**.** *Sei $p \in [0,1]$ eine Wahrscheinlichkeit, $\bowtie \in \{\leq, <, >, \geq\}$ ein Vergleichsoperator, seien $t_1, t_2 \in \mathbb{R}^+$ reelle Zeitpunkte und $ap \in AP$ eine atomare Eigenschaft. Eine CSL-Zustandsformel wird definiert durch*

$$\Phi ::= tt \mid ap \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathcal{S}_{\bowtie p}(\Phi) \mid \mathcal{P}_{\bowtie p}(\phi),$$

wobei ϕ eine Pfadformel der Form

$$\phi ::= \mathcal{X}^{[t_1,t_2]}\Phi \mid \Phi\, \mathcal{U}^{[t_1,t_2]}\, \Phi$$

ist.

Die verschiedenen CSL-Operatoren werden an späterer Stelle genauer betrachtet und erläutert.
Wir haben nun Möglichkeiten, Eigenschaften über die QBDs zu formulieren, dessen Gültigkeit wir für verschiedene Zustände des QBD prüfen möchten. Dafür müssen wir jedoch definieren, wann ein Zustand eine CSL-Zustandsformel erfüllt. Wir werden dies formell und, zum besseren Verständnis, auch umgangssprachlich aufführen.

Notation: Die Menge aller Zustände, die eine CSL-Zustandsformel Φ erfüllen, bezeichnen wir als $\mathrm{Sat}(\Phi)$.

Die stationäre Wahrscheinlichkeit des Zustands s, eine Formel Φ zu erfüllen, ist die Summe der stationären Wahrscheinlichkeiten von s und den Zuständen, die Φ erfüllen:

$$\pi^{\mathcal{Q}}(s, Sat(\Phi)) = \sum_{s' \in Sat(\Phi)} \pi^{\mathcal{Q}}(s, s').$$

Sei nun $s \in S$ und $\models$ die Relation zwischen s und Φ, wenn s eine Formel Φ erfüllt.

$s \models tt$		$\forall\, s \in S$	*Für alle Zustände gilt die Eigenschaft true.*
$s \models ap$	$\Leftrightarrow$	$ap \in \mathcal{L}(s)$	*Zustand s besitzt die atomare Eigenschaft ap.*
$s \models \neg\Phi$	$\Leftrightarrow$	$s \not\models \Phi$	*Zustand s erfüllt Φ nicht.*
$s \models \Phi \wedge \Psi$	$\Leftrightarrow$	$s \models \Phi$ und $s \models \Psi$	*Zustand s erfüllt Φ und Ψ.*
$s \models S_{\bowtie p}(\Phi)$	$\Leftrightarrow$	$\pi^{\mathcal{Q}}(s, Sat(\Phi)) \bowtie p$	*Die stationäre Wahrscheinlichkeit von s, sich in einem Zustand zu befinden, der Φ erfüllt, ist $\bowtie p$.*
$s \models P_{\bowtie p}(\phi)$	$\Leftrightarrow$	$Prob^{\mathcal{Q}}(s, \phi) \bowtie p$	*Die Wahrscheinlichkeit von s, die Pfadformel ϕ zu erfüllen, ist $\bowtie p$.*

Die Wahrscheinlichkeit, dass ein Zustand s eine Pfadformel ϕ erfüllt, ist die Wahrscheinlichkeit, dass seine Pfade die Pfadformel erfüllen:

$$Prob^{\mathcal{Q}}(s, \phi) = Pr\{\sigma \in Path^{\mathcal{Q}}(s) \mid \sigma \models \phi\}.$$

Sei daher σ ein Pfad und $\models$ die Relation zwischen σ und ϕ, wenn σ die Pfadformel ϕ erfüllt.

$\sigma \models \mathcal{X}^{[t_1,t_2]}\Phi \quad \Leftrightarrow \quad \sigma[1] \neq \infty$ und $\sigma[1] \models \Phi$ und $t_1 \leq \delta(\sigma, 0) \leq t_2$
Der nächste Zustand auf dem Pfad σ erfüllt Φ und wird zwischen den Zeitpunkten t_1 und t_2 erreicht.

$\sigma \models \Phi\, \mathcal{U}^{[t_1,t_2]}\, \Psi \Leftrightarrow \quad \exists t(t_1 \leq t \leq t_2)(\sigma@t \models \Psi \wedge (\forall t' \in [0, t)(\sigma@t' \models \Phi)))$
Ein Zustand des Pfads σ erfüllt Ψ und wird zwischen den Zeitpunkten t_1 und t_2 erreicht. Alle zuvor besuchten Zustände erfüllen Φ.

Damit wissen wir, wie wir eine Eigenschaft formulieren können und wann sie gültig ist. Betrachten wir dies nun konkret anhand einiger Beispiele. Dabei beziehen wir uns auf unsere als QBD modellierte Warteschlange aus Abbildung 2.4.

„Die Wahrscheinlichkeit, dass innerhalb von 10 Zeiteinheiten bereits 3 Aufträge in der Warteschlange gespeichert sind, ist größer als 0.1“

$$\Rightarrow P_{>0.1}(tt\, \mathcal{U}^{[0,10]} \text{ three jobs})$$

„Die Wahrscheinlichkeit, dass die Warteschlange dauerhaft leer ist, ist kleiner als 0.2“

$$\Rightarrow S_{<0.2}(\text{empty})$$

Im nächsten Abschnitt nutzen wir die spezielle Struktur des QBD aus, um Levelunabhängigkeit zu definieren und levelunabhängige Eigenschaften zu erläutern.

2.4. Levelunabhängigkeit

Für das spätere Model Checking beschränken wir uns auf stark zusammenhängende QBDs mit *levelunabhängigen* atomaren Eigenschaften.
Wenn wir die spezielle Struktur eines QBD ausnutzen möchten, so ist es notwendig zu beobachten, welche Zustände in jedem Level vorhanden sind und die gleiche Position in ihrem jeweiligen Level besitzen. Wir definieren dafür den Begriff der korrespondierenden Zustände.

Definition 10 (Korrespondierende Zustände)**.** *Zwei Zustände $(i_1, j_1) \in S$ und $(i_2, j_2) \in S$ korrespondieren genau dann, wenn für sie gilt: $i_1 = i_2 \wedge j_1, j_2 > 0$.*

Intuitiv korrespondieren zwei Zustände, wenn sie (unterschiedlichen) Wiederholungsleveln zugeordnet sind, aber die gleiche Position im Level einnehmen.
Für die Levelunabhängigkeit einer atomaren Eigenschaft möchten wir nun zusätzlich fordern, dass die korrespondierenden Zustände entweder beide die atomare Eigenschaft besitzen oder sie beide nicht besitzen.

Definition 11 (Levelunabhängigkeit von atomaren Eigenschaften)**.** *Sei $i \in \{0, ..., N-1\}$ eine beliebige Position in einem Wiederholungslevel und $ap \in AP$ eine atomare Eigenschaft. Dann ist ap levelunabhängig genau dann, wenn $\forall\, l, k \geq 1 : ap \in \mathcal{L}(i, k) \Leftrightarrow ap \in \mathcal{L}(i, l)$.*

Wenn wir jedoch nicht nur atomare Eigenschaften bei QBDs prüfen wollen, sondern auch deren Kombinationen durch die Verwendung von CSL-Formeln, so müssen wir zunächst den Zusammenhang von CSL-Formeln und Levelunabhängigkeit herstellen.

Das Grundlevel und das Grenzlevel besitzen andere Strukturen als die Wiederholungslevel, die immer gleich aufgebaut sind. Aufgrund dieser Tatsache lässt sich bereits vermuten, dass es keine allgemeine Levelunabhängigkeit für CSL-Formeln gibt, sondern, dass Levelunabhängigkeit für jede CSL-Formel eine andere Interpretation besitzt. Insbesondere ist Levelunabhängigkeit von CSL-Formeln *für alle Level* kaum realistisch.

Ein simples Beispiel dafür kann wieder mit dem Warteschlangenmodell aus Abbildung 2.4 gegeben werden. Wir möchten die CSL-Formel

$$\Phi = P_{>0}(\mathcal{X}^{[0,t]} \text{ empty}), \quad t \in \mathbb{R}^+$$

prüfen. Es ist direkt klar, dass diese Formel für die Zustände s_0 und s_1 erfüllt ist, da empty $\in \mathcal{L}(s_0)$ und s_1 den Zustand s_0 innerhalb von einer Transition erreichen kann. Insbesondere ist diese Formel dadurch für Level 0 und Level 1 erfüllt. Ab Level 2 jedoch, welches aus dem Zustand s_2 besteht, kann diese Formel nicht mehr erfüllt werden, weil kein Zustand mit der Eigenschaft empty noch erreichbar ist. Deshalb ist die CSL-Formel nicht levelunabhängig. Durchaus denkbar ist jedoch eine Levelunabhängigkeit ab einem bestimmten Index, das heißt für alle Levelindizes größer gleich eines bestimmten Levelindex. In diesem Fall ist die oben aufgeführte Formel zum Beispiel levelunabhängig ab Level 2, da ab dort keines der Level mehr die Formel erfüllen kann.
Wir verallgemeinern dies in einer Levelunabhängigkeit-Definition für CSL-Formeln, die teilweise von der Definition zur Levelunabhängigkeit von atomaren Eigenschaften adaptiert wird.

Definition 12 (Levelunabhängigkeit von CSL-Formeln)**.** *Sei $\mathcal{Q}$ ein QBD der Ordnung (N_0, N). Eine CSL-Zustandsformel Φ ist levelunabhängig für ein Level $k \geq 1$ genau dann, wenn*

$$\forall\, i \in \{0, ..., N-1\} \wedge \forall\, l \geq k : (i, l) \models \Phi \Leftrightarrow (i, k) \models \Phi.$$

Zu beachten ist, dass wir bei dieser Definition die Levelunabhängigkeit für alle Positionen $i \in \{0, ..., N-1\}$ innerhalb der Level fordern und nicht mehr nur für eine beliebige Position. Dadurch haben wir für unsere CSL-Formel dieselbe Struktur in allen Leveln ab dem Levelindex l.

Es wurde bereits von Remke et al. [Rem+07] induktiv bewiesen, dass für jeden stark zusammenhängenden QBD $\mathcal{Q}$ ein Level $k \geq 1$ existiert, für welches die CSL-Formel Φ levelunabhängig ist. Der folgende Satz hält diese Aussage fest:

Satz 1 (Existenz)**.** *Sei $\mathcal{Q}$ ein QBD, dessen atomare Eigenschaften levelunabhängig sind und sei Φ eine CSL-Zustandsformel, aber $\Phi \neq \mathcal{P}_{\bowtie p}(\Phi\ \mathcal{U}^I\ \Psi)$, wobei I ein Intervall ist.*
Dann existiert $k \in \mathbb{N}$ derart, dass Φ levelunabhängig für das Level k in $\mathcal{Q}$ ist.
Sei $\Phi = \mathcal{P}_{\bowtie p}(\Phi\ \mathcal{U}^I\ \Psi)$, dann fordern wir zusätzlich, dass $\forall\, s \in S$:
$Prob(s, \Phi\ \mathcal{U}^I\ \Psi) \neq p$ gilt. Dann existiert ebenfalls ein $k \in \mathbb{N}$ derart, dass Φ levelunabhängig für das Level k in $\mathcal{Q}$ ist.

Wir müssen uns dafür nicht darauf beschränken, dass jede atomare Eigenschaft levelunabhängig ist, sondern es genügt, dass sie levelunabhängig für ein Level $k \geq 1$ ist, da wir dann die Level 0 bis $k-1$ zu einem neuen Grundlevel zusammenfassen. Zu beachten ist allerdings für die Until-Formel, dass ihre exakte Wahrscheinlichkeit nicht genau die Wahrscheinlichkeitsgrenze p sein darf. In

diesem Fall kann es sein, dass man wirklich erst im unendlichen Fall die Grenze p erreichen wird und somit kein Level findet, ab welchem die Formel levelunabhängig ist. Der Fall tritt in der Anwendung allerdings sehr selten auf.
Damit haben wir nun den QBD und seine besonderen Eigenschaften hinreichend definiert, um uns im nächsten Kapitel dem Model Checking eines QBD zu widmen.

3. Model Checking

Im Folgenden werden wir uns mit der Fragestellung beschäftigen, wie wir die CSL-Formeln aus Kapitel 2, Abschnitt 2.3, prüfen können. Dabei nutzen wir aus, dass wir QBDs betrachten, welche eine Teilmenge der allgemeinen CTMCs bilden.
Wir werden daher zunächst die allgemeine Methode der Uniformisierung für CTMCs vorstellen. Anschließend erläutern wir, wie wir, trotz der Unendlichkeit des QBDs, eine endliche Matrixdarstellung erreichen können. Es folgt eine Beschreibung der Uniformisierung mit Repräsentanten, die die in Abschnitt 2.4 vorgestellte Levelunabhängigkeit verwendet, um die Bestimmung von transienten Zustandswahrscheinlichkeiten eines QBDs zu ermöglichen.
Zuletzt führen wir ein dynamisches Abbruchkriterium ein, mit welchem wir effizienter eine Aussage über die Gültigkeit einer CSL-Formel treffen können, als wenn wir eine *a priori*-Fehlerschranke verwenden würden.
Wir beziehen uns in diesem Kapitel auf die Erläuterungen von Kwiatkowska et al. [Kwi+07] und Remke et al. [Rem+07].

3.1. Uniformisierung

Bereits aus der Analyse von endlichen CTMCs in der Literatur wissen wir, dass die exakte Berechnung von transienten Wahrscheinlichkeiten oftmals schwierig ist. Im allgemeinen Fall muss dafür ein lineares Gleichungssystem von Differentialgleichungen, bekannt als Chapman-Kolmogorow-Gleichungen, gelöst werden. Da dies in der Umsetzung kaum durchführbar ist, wird häufig auf die Methode der *Uniformisierung (Uniformization)* zurückgegriffen, vgl. Baier et al. [Bai+03] und Kwiatkowska et al. [Kwi+07].
Dabei berechnen wir die transienten Wahrscheinlichkeiten der CTMC, indem wir diese auf ihre *eingebettete DTMC (Discrete-Time Markov Chain)* reduzieren. Wir müssen beachten, dass in einer CTMC Übergänge von unterschiedlicher Zeit modelliert werden, aber in der DTMC jeder Übergang in demselben Zeitschritt genommen wird. Daher haben wir in der DTMC nur noch die Wahrscheinlichkeit, exakt eine Transition zu wählen und können dafür genau die bereits gegangenen Schritte zählen. Eine CTMC besitzt eine exponentialverteilte Zufallsvariable pro Zustand, die die Aufenthaltsdauer im Zustand modelliert. Die Raten ergeben sich dann dadurch, dass man die Wahrscheinlichkeit, eine Transition zu wählen, mit dem Wert dieser Zufallsvariable multipliziert. Diese Informationen müssen wir in der DTMC aber auch berücksichtigen, ob-

wohl wir dort nicht unterschiedliche Zeitabschnitte den Transitionen zuordnen können. Deshalb interpretieren wir die größte Ausgangsrate unseres QBD $\mathcal{Q}$, im Folgenden mit λ bezeichnet, als Parameter eines Poisson-Prozesses. Das heißt, wir erwarten pro Zeiteinheit genau λ Ereignisse. Die Wahrscheinlichkeit, eine Transition der eingebetteten DTMC zu wählen, ist dann annähernd das Verhältnis der entsprechenden Ausgangsrate in der CTMC zur maximalen Ausgangsrate.
Das Verfahren ergibt eine ausreichende Approximation, da wir dabei ausnutzen können, dass die Zeiten zwischen den Ereignissen des Poisson-Prozesses exponentialverteilt sind, also dieselbe Verteilung besitzen wie die Zufallsvariablen der CTMC-Zustände.
Wir möchten im Folgenden nicht noch näher darauf eingehen, wie genau diese Approximation durch die Uniformisierung ist. Es gibt bereits zahlreiche wissenschaftliche Arbeiten, die dies genauer untersuchen und wir konzentrieren uns daher nur auf dieser Verwendung der Methode. Dabei beziehen wir uns sowohl auf Remke et al. [Rem+07] als auch auf Kwiatkowska et al. [Kwi+07].

Sei $\mathcal{Q}$ unser QBD, dessen Generatormatrix $\mathbf{Q}$ eine Struktur wie in Kapitel 2.2 besitzt, und λ die größte Ausgangsrate von $\mathcal{Q}$, das heißt

$$\lambda := \max_{i \in \mathbb{N}_0} \{|\mathbf{Q}_{i,i}|\}.$$

Da unser QBD aus unendlich vielen Wiederholungsleveln und somit aus unendlich vielen Zuständen besteht, ist zwar auch die Anzahl der Ausgangsraten unendlich, doch unterscheiden sich die Ausgangsraten ab dem zweiten Wiederholungslevel nicht mehr. Konkret verändern sich die Diagonaleinträge unserer Generatormatrix $\mathbf{Q}$ ab dem zweiten Wiederholungslevel nicht und es genügt den betragsmäßig größten Diagonaleintrag der Matrizen $\mathbf{B}_{0,0}$, $\mathbf{B}_{1,1}$ und $\mathbf{A}_1$ zu finden. Wir durchsuchen also maximal N_0+2N Diagonaleinträge, um die größte Ausgangsrate λ zu bestimmen.
Die Wahrscheinlichkeitsmatrix $\mathbf{P}$ der eingebetteten DTMC wird damit wie folgt bestimmt:

$$\mathbf{P} := \mathbf{I} + \frac{\mathbf{Q}}{\lambda} \tag{3.1}$$

Die Addition mit der Einheitsmatrix $\mathbf{I}$ ist unter anderem nötig, da nicht notwendigerweise alle Ausgangsraten gleich sind und deshalb sogenannte *Self-Loops* hinzugefügt werden, um fehlende Wahrscheinlichkeiten auszugleichen. Ein Zustand s besitzt einen Self-Loop genau dann, wenn $\mathbf{P}(s,s) > 0$. $\mathbf{P}$ ist nun eine stochastische Matrix, da ihre Zeilensummen jeweils genau 1 und ihre Einträge größer als 0 und kleiner als 1 sind.
Da unser QBD unendlich ist und damit auch die Generatormatrix $\mathbf{Q}$, wird unsere Wahrscheinlichkeitsmatrix $\mathbf{P}$ ebenfalls unendlich groß. Sie behält jedoch die tridiagonale Struktur von $\mathbf{Q}$ bei, da wir $\mathbf{Q}$ nur mit dem Faktor $\frac{1}{\lambda}$ multiplizieren. Deshalb können wir speziell die neue Struktur der Blockmatrizen $\hat{\mathbf{B}}_{0,0}, \hat{\mathbf{B}}_{0,1}, \hat{\mathbf{B}}_{1,0}, \hat{\mathbf{A}}_0, \hat{\mathbf{A}}_1, \hat{\mathbf{A}}_2$ durch Anpassung der ursprünglichen Blockmatrizen

bestimmen:

$$\hat{\mathbf{B}}_{i,j} = \begin{cases} I + \frac{\mathbf{B}_{i,j}}{\lambda}, & \text{wenn i = j} \\ \frac{\mathbf{B}_{i,j}}{\lambda}, & \text{wenn i} \neq \text{j} \end{cases} \quad \text{und } \hat{\mathbf{A}}_i = \begin{cases} I + \frac{\mathbf{A}_i}{\lambda}, & \text{wenn i = 1} \\ \frac{\mathbf{A}_i}{\lambda}, & \text{wenn i} \neq 1 \end{cases} \tag{3.2}$$

Damit haben wir unsere Wahrscheinlichkeitsmatrix der eingebetteten DTMC bestimmt. Wir müssen jedoch noch den Zusammenhang zwischen ihr und den transienten Zustandswahrscheinlichkeiten der CTMC herstellen.
Sei nun $\mathbf{U}^{(k)}$ die Zustandswahrscheinlichkeitsmatrix nach k Schritten in der DTMC mit der Wahrscheinlichkeitsmatrix $\mathbf{P}$. Der Eintrag $\mathbf{U}^{(k)}_{i,j}$ bezeichnet dann die Wahrscheinlichkeit, den Zustand j vom Zustand i aus in k Schritten zu erreichen. $\mathbf{U}^{(k)}$ kann rekursiv berechnet werden durch:

$$\mathbf{U}^{(0)} = I, \qquad \mathbf{U}^{(k)} = \mathbf{U}^{(k-1)}\mathbf{P}, \quad k \in \mathbb{N}^+ \tag{3.3}$$

Wenn wir nun noch die Wahrscheinlichkeit hinzunehmen, dass in dem Zeitintervall $[0, t)$ k Ereignisse in einem Poisson-Prozess mit dem Parameter λ eintreffen, notiert als $\psi(\lambda t; k)$, so können wir die Matrix $\mathbf{V}$(t) der transienten Zustandswahrscheinlichkeiten der CTMC zum Zeitpunkt t wie folgt berechnen:

$$\mathbf{V}(t) = \sum_{k=0}^{\infty} \psi(\lambda t; k)\mathbf{P}^k = \sum_{k=0}^{\infty} \psi(\lambda t; k)\mathbf{U}^{(k)} \tag{3.4}$$

Im Prinzip multiplizieren wir lediglich die Wahrscheinlichkeit, dass k Ereignisse in $[0, t)$ eintreffen mit den Zustandswahrscheinlichkeiten der DTMC nach genau k Schritten. Dann summieren wir über alle möglichen Schrittanzahlen k, das heißt, wir beachten jede mögliche Schrittanzahl k für das Intervall $[0, t)$, um das Verhalten der CTMC im Intervall $[0, t)$ zu simulieren.
Da wir nun eine unendliche Summe haben, die wir nicht exakt ausrechnen können, approximieren wir die Summe bis zu n Schritten, also:

$$\mathbf{V}^{(n)}(t) = \sum_{k=0}^{n} \psi(\lambda t; k)\mathbf{U}^{(k)} \tag{3.5}$$

Falls wir eine höhere Genauigkeit fordern, kann die transiente Zustandswahrscheinlichkeit nach n+1 Schritten für ein beliebiges $n \in \mathbb{N}_0$ iterativ berechnet werden:

$$\begin{aligned} \mathbf{V}^{(n+1)}(t) &= \sum_{k=0}^{n+1} \psi(\lambda t; k)\mathbf{U}^{(k)} \\ &= \sum_{k=0}^{n} \psi(\lambda t; k)\mathbf{U}^{(k)} + \psi(\lambda t; n+1)\mathbf{U}^{(n+1)} \\ &= \mathbf{V}^{(n)}(t) + \psi(\lambda t; n+1)\mathbf{U}^{(n+1)} \end{aligned} \tag{3.6}$$

Durch das Abschneiden der Summe nach n Schritten erhalten wir einen gewissen Fehler, welchen wir in Satz 2 abschätzen. Die dort verwendete Norm ist die Maximumsnorm:

$$\|\mathbf{A}\|_\infty := \max_{\substack{i=1,\dots,m \\ j=1,\dots,n}} |\mathbf{A}_{i,j}| \quad \text{für eine Matrix } \mathbf{A} \in \mathbb{R}^{m \times n}.$$

Satz 2 (maximaler Fehler der Uniformisierung)**.** *Sei $\boldsymbol{U}^{(k)}$ die Zustandswahrscheinlichkeitsmatrix nach k Schritten, λ die größte Ausgangsrate, t der Zeitschritt und n die Anzahl der Schritte. Dann ist der maximale Fehler $\varepsilon_{t,\lambda}^{(n)}$ der Uniformisierung*

$$\varepsilon_{t,\lambda}^{(n)} = \left\| \sum_{k=n+1}^{\infty} \psi(\lambda t; k)\, \boldsymbol{U}^{(k)} \right\| \leq 1 - \sum_{k=0}^{n} e^{-\lambda t} \frac{(\lambda t)^k}{k!}.$$

Beweis: Dies können wir abschätzen, da $\sum_{k=0}^{\infty} \left\| \psi(\lambda t; k) \mathbf{U}^{(k)} \right\| < \infty$ und

$$\begin{aligned}
\varepsilon_{t,\lambda}^{(n)} &= \left\| \sum_{k=n+1}^{\infty} \psi(\lambda t; k) \mathbf{U}^{(k)} \right\| \\
&\overset{\text{Subadditivität}}{\leq} \sum_{k=n+1}^{\infty} \left\| \psi(\lambda t; k) \mathbf{U}^{(k)} \right\| \\
&\overset{\text{Homogenität}}{\leq} \sum_{k=n+1}^{\infty} |\psi(\lambda t; k)| \left\| \mathbf{U}^{(k)} \right\| \\
&\overset{\|\mathbf{U}^{(k)}\| \leq 1}{\leq} \sum_{k=n+1}^{\infty} |\psi(\lambda t; k)| \\
&\overset{\psi(\lambda t; k) \geq 0}{\leq} \sum_{k=n+1}^{\infty} \psi(\lambda t; k) \\
&= \sum_{k=0}^{\infty} \psi(\lambda t; k) - \sum_{k=0}^{n} \psi(\lambda t; k) \\
&= 1 - \sum_{k=0}^{n} \psi(\lambda t; k) \\
&= 1 - \sum_{k=0}^{n} e^{-\lambda t} \frac{(\lambda t)^k}{k!} \qquad \square
\end{aligned}$$

Wenn $\varepsilon_{t,\lambda}^{(n)}$, λ und t gegeben sind, kann so die nötige Anzahl an Iterationen n schon *a priori* berechnet werden. Wenn wiederum eine feste Anzahl von Iterationen n vorgegeben wird, steigt $\varepsilon_{t,\lambda}^{(n)}$ an, je größer λt ist.

3.2. Endliche Matrixdarstellung

Bisher haben wir das allgemeine Prinzip der Uniformisierung geschildert. Zu beachten ist dabei jedoch, dass unsere Zustandswahrscheinlichkeitsverteilungsmatrix $\mathbf{U}^{(k)}$ und unsere Wahrscheinlichkeitsmatrix $\mathbf{V}^{(n)}(t)$ unendlich groß sind. Abhängig von der Iterationsanzahl n bei der Uniformisierung und von der speziellen Struktur des QBD können wir jedoch im Folgenden auch eine endliche Darstellung herleiten.
Von jedem Startzustand des QBDs ist mit n Schritten nur eine endliche Anzahl

von Zuständen erreichbar. Die transiente Zustandswahrscheinlichkeit, einen nicht-erreichbaren Zustand zu erreichen, beträgt 0. Deshalb genügt es, wenn wir bei der Berechnung der gesamten transienten Zustandswahrscheinlichkeiten nur die erreichbaren Zustände betrachten. Diese Stellen für genau einen Startzustand eine endliche Menge durch die Endlichkeit der Schrittanzahl dar. Würde man nun für alle möglichen Startzustände, welche unendlich viele sind, die endlichen Mengen von erreichbaren Zuständen betrachten, so wären dies immer noch unendlich viele Zustände. Allerdings existiert ein Wiederholungslevel mit dem Index l, von dem wir mit maximal n Schritten nicht mehr das Grundlevel erreichen können. Deshalb besitzen alle von diesem Level l aus erreichbaren Level dieselbe Struktur, da sie alle Wiederholungslevel sind.
Betrachten wir nun alle korrespondierenden Zustände von Leveln größer l als Startzustände, so haben diese alle dieselbe transiente Wahrscheinlichkeit. Diesen Umstand nutzen wir aus, indem wir die Zustände des Levels l als *Repräsentanten* für alle korrespondierenden Zustände der höheren Level ansehen. Wir erreichen dadurch, dass die Berechnung für beliebige Startzustände auf endlich viele Startzustände eingegrenzt werden kann und trotzdem eine aussagekräftige transiente Analyse für alle Startzustände möglich ist.
Für eine endliche Darstellung von $\mathbf{U}^{(k)}$ und $\mathbf{V}^{(n)}(t)$ ist es nun ausreichend, wenn die positiven Einträge der Startzustände bis einschließlich Level l gespeichert werden. Welches Level l genügt, ist abhängig von der Iterationsanzahl n und dementsprechend von der Zeit t, der Uniformisierungsrate λ und der geforderten Genauigkeit von $\varepsilon_{t,\lambda}^{(n)}$.
Außerdem muss beachtet werden, mit wie vielen Schritten ein Level durchquert werden kann. Wir möchten dies anhand des Leveldurchmessers angeben. Um diesen formell zu definieren, benötigen wir jedoch noch einige Aussagen über die Position eines Zustands in seinem Level.

Definition 13 (Randzustände)**.** *Sei $i \in \mathbb{N}^+$ der Index eines beliebigen Wiederholungslevels. $S_{in}^{i,\uparrow}$ ist die Menge aller Zustände, die vom nächstniedrigeren Level $i-1$ in einem Schritt erreicht werden können.*
Analog ist $S_{in}^{i,\downarrow}$ die Menge aller Zustände, die vom nächsthöheren Level $i+1$ in einem Schritt erreicht werden können.

Definition 14 (Kürzester Pfad)**.** *Seien $s_1, s_2 \in S$ beliebige Zustände. Der Pfad mit der minimalen Schrittanzahl, um von s_1 zu s_2 zu gelangen, wird als kürzester Pfad bezeichnet. Die nötige Schrittanzahl beträgt*

$$g(s_1, s_2) = |\textit{kürzester Pfad}(s_1, s_2)|.$$

Damit können wir nun auch Aussagen über die Beziehungen zwischen den Zuständen treffen und den Leveldurchmesser definieren.

Definition 15 (Leveldurchmesser)**.** *Sei $i \in \mathbb{N}^+$ der Index eines beliebigen Wiederholungslevels. Wir definieren als Aufwärts-Leveldurchmesser die minimale*

Anzahl von nötigen Schritten, um von Level i aus das nächsthöhere Level $i+1$ zu erreichen und bezeichnen dies mit

$$d^{\uparrow} = min\{g(s_1, s_2) \mid s_1 \in S_{in}^{i,\uparrow}, s_2 \in S_{in}^{i+1,\uparrow}\}.$$

Analog ist der Abwärts-Leveldurchmesser defniert als

$$d^{\downarrow} = min\{g(s_1, s_2) \mid s_1 \in S_{in}^{i,\downarrow}, s_2 \in S_{in}^{i-1,\downarrow}\}.$$

Der symmetrische Leveldurchmesser ist dann $d = min\{d^{\uparrow}, d^{\downarrow}\}$.

Falls wir im Folgenden nur von einem Leveldurchmesser sprechen, so meinen wir damit implizit den symmetrischen Leveldurchmesser.
Mit dem Leveldurchmesser haben wir die Möglichkeit anzugeben, mit wie vielen Schritten ein Level durchquert werden kann. Wir werden deshalb die Größe von $\mathbf{U}^{(n)}$ abhängig von den Schritten n und dem Leveldurchmesser d bestimmen.

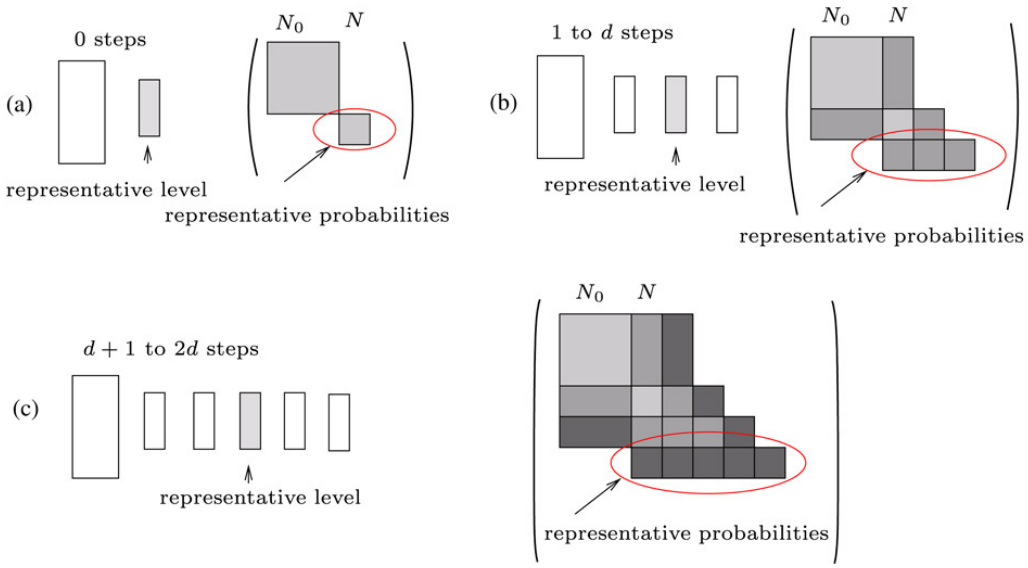


Abbildung 3.1.: Repräsentatives Level abhängig von n, Quelle: Remke et al. [Rem+07]

Sei $n = 0$. Dann kann kein Schritt gegangen werden und somit auch kein Levelwechsel stattfinden. Wir können bereits das Grenzlevel als repräsentativ für alle weiteren Level ansehen. Es genügt das Grund- und das Grenzlevel in $\mathbf{U}^{(0)}$ zu speichern, also besitzt $\mathbf{U}^{(0)}$ die Dimension $(N_0 + N) \times (N_0 + N)$, siehe Abbildung 3.1(a).

Sei $n = 1$. Dann können wir maximal das nächsthöhere oder das nächstniedrigere Level erreichen. Das Grenzlevel genügt nicht als repräsentatives Level, das zweite Wiederholungslevel jedoch schon. Da wir nun in den ersten drei Leveln starten können, sind die ersten vier Level insgesamt erreichbar und

dementsprechend müssen die Transitionen von den ersten drei Leveln bis zu den ersten vier Leveln gespeichert werden, graphisch in Abbildung 3.1(b) beschrieben. Transitionen vom vierten zum dritten Level müssen nicht gespeichert werden, da wir nur in den ersten drei Leveln starten. Es ergibt sich eine Dimension von $(N_0 + 2N) \times (N_0 + 3N)$ für $U^{(1)}$.

Sei $n \in \{1, ..., d\}$. In diesem Fall ist $\mathbf{U}^{(n)} = \mathbf{U}^{(1)}$, da mindestens d+1 Schritte gegangen werden müssen, um mehr als einen Levelwechsel durchzuführen. Maximal ein Levelwechsel ist genau der Fall wie für $n = 1$.

Sei $n = d + 1, ..., 2d$. Nun sind maximal zwei Levelwechsel möglich. Von jedem Level sind sowohl die zwei nächsthöheren als auch die zwei nächstniedrigeren Level erreichbar. Daher genügt das zweite Wiederholungslevel nicht mehr als repräsentatives Level, sondern erst das dritte Wiederholungslevel. Dadurch vergrößert sich $U^{(n)}$ um eine Zeile für das neue Startlevel und um jeweils eine Spalte pro vorheriger Zeile, da nun für jedes vorherige Startlevel ein weiteres erreichbares Level hinzugekommen ist. Die Dimension der endgültigen Matrix ist dann $(N_0 + 3N) \times (N_0 + 5N)$, siehe Abbildung 3.1(c). Wenn wir nun verallgemeinern, wie viele Wiederholungslevel l maximal von einem repräsentativen Level in eine Richtung erreichbar sind, erhalten wir, abhängig von den Schritten $n \geq 1$ und dem Durchmesser d die folgende Anzahl:

$$l = \left\lfloor \frac{n-1}{d} \right\rfloor + 1 \tag{3.7}$$

Intuitiv teilen wir unsere möglichen Schritte auf mögliche Levelwechsel durch das Dividieren mit d auf, berücksichtigen aber, dass wir bei einem Schritt mindestens ein Level unabhängig vom Leveldurchmesser wechseln können.

Bei der Dimension von $\mathbf{U}^{(n)}$ wiederum sind $l + 1$ Wiederholungslevel als Startlevel anzusehen, da erst das $(l + 1)$te Level ein repräsentatives Level ist. Dementsprechend sind 2l Wiederholungslevel vom repräsentativen Level erreichbar, da in jede Richtung l Wiederholungslevel erreichbar sind, das heißt inklusive dem repräsentativen Level sind $2l + 1$ mögliche Zielwiederholungslevel erreichbar. Die Wiederholungslevel besitzen eine Breite von N und das Grundlevel eine Breite von N_0, womit sich dann die folgende Dimension von $\mathbf{U}^{(n)}$ für ein beliebiges n bzw. l ableiten lässt:

$$(N_0 + (l + 1)N) \times (N_0 + (2l + 1)N) \tag{3.8}$$

Insgesamt haben wir nun eine endliche Darstellung von $\mathbf{U}^{(n)}$. Da sich $\mathbf{V}^{(n)}$ aus $\mathbf{U}^{(n)}$ ableitet (3.4), hat $\mathbf{V}^{(n)}$ ebenfalls eine endliche Darstellung.

3.3. Uniformisierung mit Repräsentanten

In den letzten Abschnitten haben wir bereits das allgemeine Prinzip der Uniformisierung geschildert und eine endliche Darstellung für die Matrizen $\mathbf{U}^{(n)}$ und $\mathbf{V}^{(n)}$ aufgeführt. Dieses Wissen möchten wir nun kombinieren, um die Uniformisierung mit Repräsentanten nach Remke et al. [Rem+07] einzuführen.

Wir nutzen dafür aus, dass wir $\mathbf{V}^{(n)}$ nach Gleichung (3.6) iterativ aus $\mathbf{U}^{(n)}$ und $\mathbf{V}^{(n-1)}$ berechnen können. Im Gegensatz zur allgemeinen Uniformisierung können wir jetzt $\mathbf{V}^{(n)}$ endlich darstellen, indem wir ab einem repräsentativen Level keine weiteren Level in der Matrix darstellen müssen. Deshalb wird die neue Methode als *Uniformisierung mit Repräsentanten* bezeichnet.
Es ist notwendig, $\mathbf{U}^{(n)}$ entsprechend anzupassen, um die endlichen Matrizen $\mathbf{U}^{(n)}$ und $\mathbf{P}$ miteinander multiplizieren zu können. $\mathbf{U}^{(n)}$ und $\mathbf{V}^{(n)}$ besitzen eine Blockstruktur, dessen Blöcke Leveln zugeordnet werden können. Deshalb werden wir uns bei den folgenden Berechnungen auch genau auf diese Blöcke beziehen und bezeichnen die Blockmatrizen, die die Wahrscheinlichkeiten, von Zuständen in Level i zu Zuständen in Level j zu gelangen, repräsentieren, als $\mathbf{U}_{i,j}^{(n)}$ und $\mathbf{V}_{i,j}^{(n)}(t)$.

Sei $n = 0$. Dann enthält die endliche Darstellung von $\mathbf{U}^{(0)}$ die Blockmatrizen des Grund- und Grenzlevels, vgl. Abbildung 3.1(a). Wir möchten jedoch unsere Berechnungen um ein Wiederholungslevel erweitern und müssen dafür eine Blockmatrix vom vorherigen Wiederholungslevel, in diesem Fall des Grenzlevels, der endlichen Darstellung von $\mathbf{U}^{(0)}$ hinzufügen.
Die Matrix $\mathbf{P}$ wiederum muss nun die erforderlichen Informationen für drei mögliche Startwiederholungslevel und vier erreichbare Zielwiederholungslevel besitzen, das heißt aus drei Zeilen und vier Spalten Blockmatrizen bestehen. Die Multiplikation der endlichen Darstellungen von $\mathbf{U}^{(0)}$ und $\mathbf{P}$ ist in Abbildung 3.2 beispielhaft dargestellt. Wir haben bereits in den Gleichungen (3.2) eine Berechnung der einzelnen Blockmatrizen angegeben, für die nicht die gesamte Berechnung der Matrix $\mathbf{P}$ nötig ist.

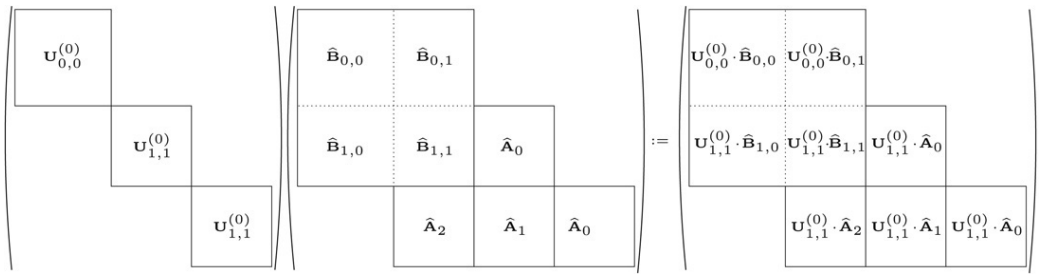


Abbildung 3.2.: Berechnung von $\mathbf{U}^{(1)} = \mathbf{U}^{(0)} \cdot \mathbf{P}$, Quelle: Remke et al. [Rem+07]

Für beliebiges $n \geq 1$ verallgemeinern wir nun die Berechnung von $U_{i,j}^{(n)}$. Dabei unterscheiden wir die Berechnungen von drei verschiedenen Arten von Blockmatrizen, je nachdem worüber die Blockmatrix etwas aussagt:

- Wechsel vom einem beliebigen Level zum Grundlevel
  $\Rightarrow$ Entweder Verweilen im Grundlevel oder vom Grenz- zum Grundlevel wechseln.

- Wechsel von einem beliebigen Level zum Grenzlevel

⇒ Wechsel vom Grund- zum Grenzlevel, Verweilen im Grenzlevel oder Wechsel vom höheren Wiederholungslevel zum Grenzlevel.

- Wechsel von einem beliebigen Level zu einem beliebigen Wiederholungslevel
  ⇒ Wechsel vom nächstniedrigeren Wiederholungslevel zum Wiederholungslevel, Verweilen im Wiederholungslevel oder Wechsel vom nächsthöheren Wiederholungslevel zum Wiederholungslevel.

Wenn wir dies formalisieren, lassen sich folgende Berechnungen von $\mathbf{U}_{i,j}^{(n)}$ für $n \geq 1$ festhalten:

$$\begin{aligned}
\mathbf{U}_{i,0}^{(n)} &= \mathbf{U}_{i,0}^{(n-1)} \cdot \hat{\mathbf{B}}_{0,0} + \mathbf{U}_{i,1}^{(n-1)} \cdot \hat{\mathbf{B}}_{1,0}, && \text{für i=0, ..., l} \quad (3.9)\\
\mathbf{U}_{i,1}^{(n)} &= \mathbf{U}_{i,0}^{(n-1)} \cdot \hat{\mathbf{B}}_{0,1} + \mathbf{U}_{i,1}^{(n-1)} \cdot \hat{\mathbf{B}}_{1,1} + \mathbf{U}_{i,2}^{(n-1)} \cdot \hat{\mathbf{A}}_2, && \text{für i=0, ..., l+1}\\
\mathbf{U}_{i,j}^{(n)} &= \mathbf{U}_{i,j-1}^{(n-1)} \cdot \hat{\mathbf{A}}_0 + \mathbf{U}_{i,j}^{(n-1)} \cdot \hat{\mathbf{A}}_1 + \mathbf{U}_{i,j+1}^{(n-1)} \cdot \hat{\mathbf{A}}_2, && \text{für i=0, ..., l+1}\\
& && \text{j=2, ..., i+l}
\end{aligned}$$

Dabei sei l die Anzahl der erreichbaren Wiederholungslevel nach der Gleichung (3.7). Wenn wir nun die Berechnungen von $\mathbf{U}_{i,j}^{(n)}$ durch Blockmatrizen auf $\mathbf{V}_{i,j}^{(n)}$ übertragen, erhalten wir:

$$\begin{aligned}
\mathbf{V}_{i,j}^{(n)}(t) = \mathbf{V}_{i,j}^{(n-1)}(t) + \psi(\lambda t; n)\mathbf{U}_{i,j}^{(n)} \qquad & \text{für i=0, ..., l+1} \quad (3.10)\\
& \text{j=max\{0, i-l\}, ..., i+l}
\end{aligned}$$

Nach d Schritten muss die Größe von $\mathbf{V}^{(n)}(t)$ angepasst werden, da immer dann ein neues Level erreicht werden kann. Dies ist aber vertretbar, da wir entweder Null-Blockmatrizen oder Kopien von bereits berechneten Blockmatrizen hinzufügen.
Zusammengefasst lässt sich Folgendes sagen: Es ist möglich, die transienten Zustandswahrscheinlichkeiten durch die Uniformisierung mit Repräsentanten zu berechnen ohne dabei unendliche Matrizen verwenden zu müssen. Die unendliche Summe können wir dabei approximieren und den maximalen Fehler angeben.
Uns fehlt zum jetzigen Zeitpunkt noch der Zusammenhang zwischen der Uniformisierung mit Repräsentanten und dem Model Checking von CSL-Formeln. Dies wollen wir im nächsten Abschnitt behandeln.

3.4. Dynamisches Abbruchkriterium

Das Prinzip der Uniformisierung mit Repräsentanten wird verwendet, wenn transiente Zustandswahrscheinlichkeiten einer CTMC berechnet werden müssen. Dies ist zum Beispiel der Fall, wenn wir eine CSL-Formel der Form $P_{\bowtie p}(\Phi\, \mathcal{U}^{[0,t]}\, \Psi)$ untersuchen möchten.
Wir müssen dafür nicht die exakte Wahrscheinlichkeit $Prob^{\mathcal{Q}}(s, \Phi\ \mathcal{U}^{[0,t]}\ \Psi)$

berechnen, da wir nur eine Aussage darüber treffen möchten, ob sie $\bowtie p$ ist. Geben wir nun einen maximalen Fehler *a priori* an, zum Beispiel unter der Nutzung von Satz 2, so könnten wir eine feste Anzahl n von Iterationen bestimmen. Es kann aber sein, dass diese n Iterationen gar nicht nötig sind, sondern man bereits mit weniger Iterationen eine Aussage über die Gültigkeit der CSL-Formel treffen kann.
Andererseits reicht möglicherweise die Genauigkeit der Wahrscheinlichkeit nicht aus, um eine Aussage treffen zu können, sodass man im Nachhinein die Iterationsanzahl n erhöhen muss. Als Resultat dieser Nachteile einer statischen Iterationsanzahl n wurde von Remke et al. [Rem+07] ein *dynamisches Abbruchkriterium* entwickelt.

Für das Model Checking von $P_{\bowtie p}(\Phi\, \mathcal{U}^{[0,t]}\, \Psi)$ wird der QBD $\mathcal{Q}$ speziell transformiert. Da beim Erreichen eines Ψ-Zustands innerhalb von [0,t] die nachfolgenden Zustände nicht mehr relevant für die Gültigkeit der CSL-Formel sind und gleichzeitig die Formel auf jeden Fall ungültig ist, sobald ein $\neg\Psi \wedge \neg\Psi$-Zustand erreicht ist, machen wir diese Zustände absorbierend. Den resultierenden QBD bezeichnen wir als $\mathcal{Q}[\neg\Psi \vee \Psi]$.
Die Wahrscheinlichkeit $\gamma_s(t)$, von einem Startzustand s in der transformierten CTMC einen Ψ-Zustand zu erreichen, sodass die Formel erfüllt wird, beträgt:

$$\gamma_s(t) = \sum_{i=0}^{\infty} \sum_{s' \in Sat^i(\Psi)} \mathcal{V}^{[\neg\Phi \vee \Psi]}(s, s', t) \tag{3.11}$$

Wir addieren dabei über alle Level i und über alle Ψ-Zustände eines Levels i die transiente Wahrscheinlichkeit des Startzustands s zu dem Ψ-Zustand s' bis zum Zeitpunkt t zu gelangen.
Wenn wir nun diese Wahrscheinlichkeit approximieren, indem wir nur eine endliche Anzahl n von Iterationen durchführen, erhalten wir den folgenden Wahrscheinlichkeitsvektor:

$$\gamma^{(n)}(t) = V^{(n)}(t) \cdot \gamma(0) \qquad \text{mit } \gamma_s(0) = \begin{cases} 1, & s \models \Psi, \\ 0, & \text{sonst} \end{cases} \tag{3.12}$$

Anzumerken ist hierbei, dass der Vektor von unendlicher Größe ist, da unendlich viele Startzustände betrachten werden müssen.
Wie wir aber bereits aus den letzten Kapiteln wissen, können wir durch die Verwendung eines repräsentativen Levels die Anzahl der Startzustände auf endlich viele begrenzen. Daher schneiden wir den Vektor $\gamma^{(n)}(t)$ nach dem repräsentativen Level ab.
Falls wir nun die Iterationsanzahl n erhöhen, so steigt die Wahrscheinlichkeit mindestens monoton an. Aufgrund dieser Tatsache können wir das Entscheidungsproblem der Gültigkeit der CSL-Formel möglicherweise bereits für kleinere n lösen. Falls die Iterationsanzahl n nicht ausreicht, können wir diese jedoch auch erhöhen, um dann eine Aussage zu treffen.
Wenn wir nun unseren maximalen Fehler nach Satz 2 hinzunehmen, so wissen

wir, dass $\gamma_s(t) \leq \gamma_s^{(n)}(t) + \varepsilon_{t,\lambda}^{(n)}$ für ein Zeitintervall $I = [0, t]$ gilt. Wir können ebenfalls feststellen, dass der maximale Fehler abnimmt, wenn die Iterationsanzahl n zunimmt. Dadurch lässt sich folgendes dynamisches Abbruchkriterium herleiten:

$$(a) \qquad \gamma_s^{(n)}(t) \geq p \qquad \Rightarrow \qquad \gamma_s(t) \geq p, \tag{3.13}$$

$$(b) \qquad \gamma_s^{(n)}(t) \leq p - \varepsilon_{t,\lambda}^{(n)} \qquad \Rightarrow \qquad \gamma_s(t) \leq p. \tag{3.14}$$

Solange nicht exakt $\gamma_s(t) = p$ gilt, können wir solange iterieren, bis entweder die Ungleichung (a) oder (b) erfüllt ist. Sollte $\gamma_s(t) = p$ gelten, ist es nicht möglich, durch das dynamische Abbruchkriterium eine Aussage zu treffen. Dies tritt jedoch nur sehr selten in der Praxis auf, insbesondere da man bereits bei der Berechnung mithilfe von Computern selten eine exakte Wahrscheinlichkeit errechnen kann. Daher nehmen wir an, dass für die folgenden Analysen $\gamma_s(t) \neq p$ gilt.
Zusammengefasst lässt sich sagen, dass wir in diesem Kapitel die Levelunabhängigkeit eingeführt haben und mithilfe von dieser eine neue Art der Uniformisierung zeigen konnten. Diese Uniformisierung mit Repräsentanten ist eine akzeptable Methode zur Bestimmung von transienten Zustandswahrscheinlichkeiten, obwohl der zu analysierende QBD unendlich groß ist. Zuletzt haben wir ein dynamisches Abbruchkriterium bestimmt, mit dem wir effizient eine Aussage über die Gültigkeit einer CSL-Formel treffen können.
Dieses theoretische Wissen möchten wir nun in den folgenden Kapiteln anhand eines Anwendungsbeispiels genauer nachvollziehen.

4. Anwendungsbeispiel

In den vorherigen Kapiteln haben wir definiert, was QBDs sind und wie durch das Prinzip der Levelunabhängigkeit und die Uniformisierung mit Repräsentanten die CSL-Operatoren effizient geprüft werden können.
Wir möchten nun eine vergleichbare, wenn auch deutlich simplere Approximation eines QBDs durchführen, indem wir die Anzahl der Wiederholungslevel begrenzen, das heißt den QBD endlich machen.
Als Anwendungsbeispiel betrachten wir das **T**ransmission **C**ontrol **P**rotocol *(Abkürzung: TCP)*, für dessen Beschreibung wir uns an den Ausführungen von Remke et al. [Rem+07] orientieren.

4.1. Beschreibung

Das TCP ist ein verbindungsorientiertes Netzwerkprotokoll. Es muss also zunächst eine Verbindung zwischen zwei Rechnern aufgebaut werden, bevor Daten als Pakete versendet werden. Da ein dauerhaftes Aufrechterhalten der Verbindung nicht kostengünstig ist, muss diese zwischendurch getrennt und wiederhergestellt werden. Die Wiederherstellung der Verbindung benötigt aufgrund technischer Details eine gewisse Minimalzeit, sodass es zu Verspätungen beim Versenden der Daten kommt.
Unser Model des TCP besteht aus drei Komponenten: *Packet Generator, Queue* und *Connection Management*. Wir möchten im Wesentlichen das Verhalten des Connection Managements analysieren, welches als *on-demand connection with delayed release* (Abkürzung: OCDR) bekannt ist.

Der Packet Generator kann die Zustände *burst* und *off* einnehmen. Wenn er im burst ist, so produziert er in kurzer Zeit sehr viele Pakete. Im Zustand off hingegen produziert er keine Pakete, sondern wartet im Ruhemodus.
Die Queue speichert die Pakete, die vom Packet Generator produziert werden. Im theoretischen Modell ist ihre Kapazität unendlich, das heißt sie nimmt beliebig viele Pakete auf. Wir werden jedoch in unserer Analyse ihre Länge beschränken und diese mit dem Parameter m bezeichnen.
Das Connection Management sorgt für den Aufbau der Verbindung zum Daten verschicken. Da es ein hoher Kostenaufwand wäre, die Verbindung kontinuierlich aufrecht zu erhalten, wechselt das Connection Management zwischen den Zuständen *active* und *released*. Im Zustand active versendet das Connection Management die Pakete aus der Queue und löst die Verbindung erst wieder, wenn die Queue geleert und nach einem gewissen Zeitraum *(Time-Out)* immer

noch keine neuen Pakete produziert wurden.
Da nach einem Wechsel des Packet Generators in einen burst-Zustand jedoch auch zum Verschicken der Pakete erst wieder die Verbindung vom Connection Management aufgebaut werden muss, ist eine Verspätung unvermeidbar. Das Connection Management versucht daher, einen Kompromiss zwischen minimalen Kosten und minimaler Verspätung zu finden. Bildlich werden diese Zusammenhänge in Abbildung 4.1 dargestellt. Für weitere Erläuterungen zum OCDR verweisen wir auf Heijenk und Haverkort [Hei+96].

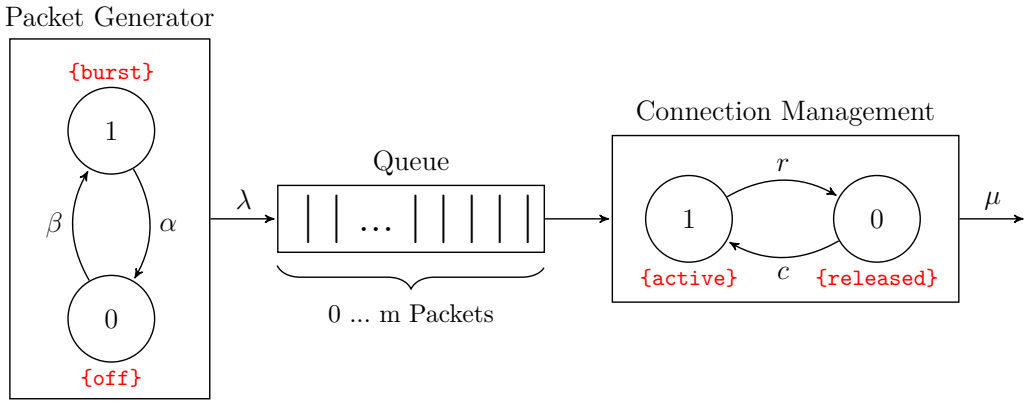


Abbildung 4.1.: Abstraktes Model des TCP nach Remke et al. [Rem+07]

Wir gehen nun davon aus, dass die Verspätung durch den nötigen Verbindungsaufbau exponentialverteilt mit Rate c ist. Ebenfalls sei das Time-Out, nach welchem die Verbindung gelöst wird, exponentialverteilt mit durchschnittlich 1/r, was zur Rate r für den Abbruch der Verbindung führt. Pakete benötigen eine exponentialverteilte Übertragungszeit, sodass wir die Rate für das Senden als μ bezeichnen. Die Produktion von Paketen wird als Poisson-Prozess mit dem Parameter λ modelliert. Der Packet Generator selbst wechselt von off zu burst mit Rate β und entsprechend von burst zu off mit Rate α.
Durch diese Modellierung erhalten wir den QBD, wie er in Abbildung 4.2 zu sehen ist.

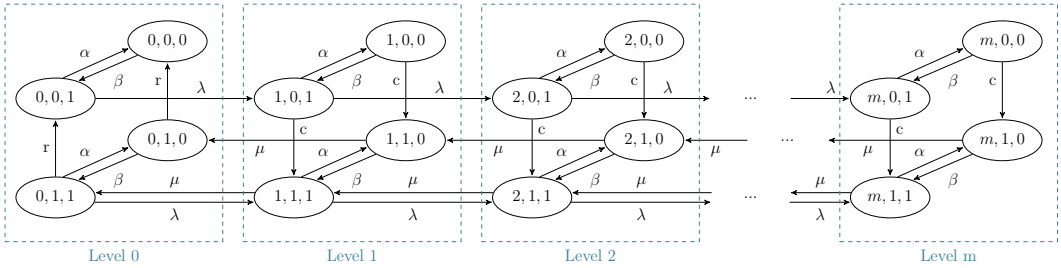


Abbildung 4.2.: QBD des Transmission Control Protocol (endliches Modell)

Der Zustandraum besitzt nun die Form

$S = \{(i, j, k) \mid (i \in \mathbb{N} \wedge i \leq m), j, k \in \{0, 1\}\}$

i: Anzahl der Pakete in der Queue

j: Connection Management, $1 \triangleq$ active, $0 \triangleq$ released

k: Packet Generator, $1 \triangleq$ burst, $0 \triangleq$ off

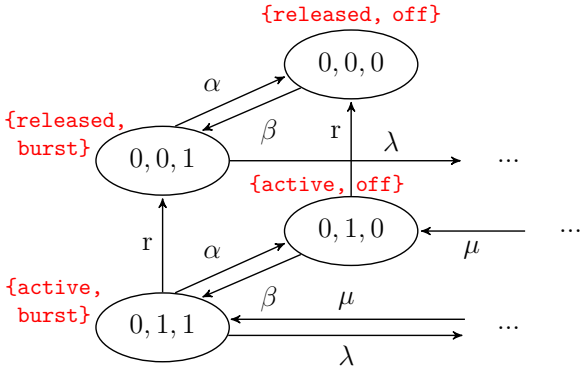
Die Variable i sorgt gleichzeitig für die Levelzuordnung der Zustände. Jedes Level besteht aus vier Zuständen:

$S^i = \{(i, 0, 0), (i, 0, 1), (i, 1, 0), (i, 1, 1)\}$

In Abbildung 4.2 sind die jeweiligen Level in blau eingerahmt.
Um den Zuständen Eigenschaften zuzuweisen, definieren wir die Labelling-Funktion $\mathcal{L} : S \to 2^{AP}$. Zur besseren Übersicht sei dabei die Bildmenge von $\mathcal{L}$ als $L : AP \to 2$ mit $2 = \{1, 0\}$ dargestellt und wie folgt definiert:

$$s \mapsto AP_s, \text{ wobei } AP_s \subseteq AP \text{ mit } \begin{cases} ap \in AP_s, & \text{wenn } L(\text{ap}) = 1 \\ ap \notin AP_s, & \text{wenn } L(\text{ap}) = 0 \end{cases}$$

Die 1 codiert, dass der Zustand die Eigenschaft besitzt, bei 0 besitzt er sie nicht. Durch diese Schreibweise erhalten wir die Labelling-Zuweisung, wie sie textuell in Abbildung 4.3 aufgeführt ist. Zusätzlich haben wir zur besseren Darstellung das Grundlevel auch graphisch gelabelt.



$\forall\, i \in \mathbb{N}_0, i \leq m :$

$(i, 0, 0) \mapsto \{\text{released, off}\}$
$(i, 0, 1) \mapsto \{\text{released, burst}\}$
$(i, 1, 0) \mapsto \{\text{active, off}\}$
$(i, 1, 1) \mapsto \{\text{active, burst}\}$

Abbildung 4.3.: Labelling des TCP-QBD

Damit haben wir unser Modell des TCP formell definiert und können nun konkrete Werte für unsere Raten angeben, die in Tabelle 4.1 aufgeführt sind. Wir übernehmen dabei die Parameter von Remke et al. [Rem+07], um eine bessere Vergleichbarkeit zu erhalten. Zudem spiegelt die Größenordnung der Raten gut das Verhalten des TCP wider.
Im zweiten Abschnitt dieses Kapitels werden wir diskutieren, welche CSL-Formeln wir im Kapitel 5 analysieren möchten.

4.2. CSL-Formeln

Für die Analyse des TCP formulieren wir für uns interessante Eigenschaften des QBD als CSL-Formeln. Es ist klar, dass es nicht sinnvoll, ist alle möglichen

λ	μ	α	β	c	r
100	125	1	0.04	10	10

Tabelle 4.1.: Parameter der Case Study nach Remke et al. [Rem+07]

CSL-Formeln zu prüfen, sondern nur diejenigen, die auch inhaltlich etwas über das Verhalten des TCP aussagen. Daher berücksichtigen wir bei der Auswahl der CSL-Formeln sowohl die spezielle Struktur des TCP-QBD als auch die festgelegten Raten aus Tabelle 4.1. Wir werden zunächst herausstellen, welche CSL-Formeln besonders interessant zu prüfen sind und danach unter diesen nochmals eingrenzen, welche Formeln wir dann im Detail analysieren werden. Wir starten im Zustand (0,0,1), das heißt die Verbindung existiert noch nicht, aber wir befinden uns im burst. Solange das Connection Management noch im Zustand released ist, werden nur Pakete produziert, also pro genommener Transition in das nächsthöhere Level gewechselt. Erst wenn das Connection Management zu active umschaltet, werden Pakete versendet. Solange die Queue noch nicht leer ist, kann das Connection Management die Verbindung nicht lösen. Daher wäre es interessant zu wissen, welche stationären Wahrscheinlichkeiten die Eigenschaften $\Phi_2 = \textit{released \& burst}$ und $\Phi_3 = \textit{active \& burst}$ besitzen.
Da die Rate $\mu = 125$ zum Pakete verschicken höher ist, als die Rate $\lambda = 100$ zum Produzieren von Paketen, wird irgendwann das Time-Out erreicht, indem die Queue leer ist und das Connection Management die Verbindung wieder lösen kann. Der Zustand $\Phi_1 = \textit{active \& no burst}$ ist jedoch sehr ungünstig, da wir Kosten für das Aufrechterhalten der Verbindung haben, obwohl keine Pakete versendet werden können. Daher bietet es sich an, auch hier die stationäre Wahrscheinlichkeit zu berechnen.
Der Grund, warum wir für den Zustand $\Phi_4 = \textit{released \& no burst}$ nicht die stationäre Wahrscheinlichkeit berechnen möchten, ist dessen Erreichbarkeit (*Reachability*). Während Φ_1, Φ_2 und Φ_3 Eigenschaften von Zuständen sind, die jeweils vom nächsthöheren oder nächstniedrigeren Level erreichbar sind, das heißt bei einem Levelwechsel erfüllt werden können, so ist Φ_4 nur bei einer Transition innerhalb eines Levels zu erreichen: Mit Rate α von (i,0,1) zu (i,0,0). Sobald einmal ein Wechsel zu (i,1,1) oder (i,1,0) erfolgt, kann der Zustand Φ_4 erst nach einer Rückkehr zu Level 0 wieder erreicht werden. Ähnlich verhält sich das zwar auch für Φ_2, jedoch starten wir in diesem Zustand und deshalb ist dort die stationäre Wahrscheinlichkeit interessanter als die Erreichbarkeitswahrscheinlichkeit. Für Φ_4 aber wäre es gut zu prüfen, wie wahrscheinlich es ist den Zustand zu erreichen, das heißt wie wahrscheinlich es ist, dass der Packet Generator früher in off umschaltet als das Connection Management eine Verbindung herstellt. In dem Fall befindet sich das TCP in einem *Ruhemodus*, in dem keinerlei Funktion vorhanden ist. Deshalb prüfen wir dessen

Erreichbarkeit bis zu einem gewissen Zeitpunkt $t \in \mathbb{R}^+$ in Form einer Bounded Until-Formel.
Um für diese Formel jedoch auch Vergleiche ziehen zu können, möchten wir eine weitere Bounded Until-Formel untersuchen. Es bietet sich dabei an, eine Formel zu wählen, die ein ähnliches Verhalten vermuten lässt. Daher werden wir diese Auswahl später während unserer Analyse treffen.

Im Überblick interessieren uns also folgende CSL-Formeln:

$$S_{>p}(\Phi_1) = S_{>p}(\text{active \& no burst}) \tag{4.1}$$
$$S_{>p}(\Phi_2) = S_{>p}(\text{released \& burst}) \tag{4.2}$$
$$S_{>p}(\Phi_3) = S_{>p}(\text{active \& burst}) \tag{4.3}$$
$$P_{>p}(tt\, \mathcal{U}^{[0,t]}\, \Phi_4) = P_{>p}(tt\, \mathcal{U}^{[0,t]}\, \text{released \& no burst}), \quad t \in \mathbb{R}^+ \tag{4.4}$$

Diese sind dieselben CSL-Formeln, die auch von Remke et al. [Rem+07] untersucht werden. Es ist im Umfang dieser Arbeit nicht möglich, alle Formeln detailliert zu analysieren. Für Φ_1, Φ_2 und Φ_3 haben Remke et al. [Rem+07] sowohl stationäre Wahrscheinlichkeiten berechnet sowie auch die Iterationsanzahl gemessen. Für Φ_4 hingegen existieren nur Ergebnisse in Bezug auf die Iterationsanzahl, aber nicht über die exakte Wahrscheinlichkeit.
Bei den Bounded-Until-Formeln können wir außerdem zumindest theoretisch die Anwendung des dynamischen Abbruchkriteriums diskutieren. Deshalb verzichten wir in dieser Arbeit auf die Analyse von Φ_1, Φ_2 und Φ_3. Stattdessen werden wir umfangreicher Φ_4 untersuchen und wie bereits angekündigt auch noch eine weitere Bounded-Until-Formel prüfen, um einen Vergleich ziehen zu können.
Eine Implementierung von Φ_4 im Model Checker PRISM erfolgt im nächsten Kapitel.

5. Analyse

Im vorherigen Kapitel haben wir unser Modell definiert und die zu prüfenden CSL-Formeln diskutiert. Damit kommen wir nun zur eigentlichen Analyse des TCP. Wir verwenden dafür das Tool PRISM und möchten die Genauigkeit des Model Checkings in Bezug auf die maximale Levelanzahl m untersuchen.
Als Vorbereitung auf die Analyse werden wir zunächst unser theoretisches Modell und die Bounded-Until-Formel aus Kapitel 4 in PRISM implementieren. Des Weiteren werden wir eine Problemstellung für die folgende Analyse herausarbeiten.
Im ersten Teil der Analyse setzen wir uns dann anhand der CSL-Formel mit dem Zusammenhang von Zeitpunkten t und der maximalen Levelanzahl m auseinander. Wir werden dabei einige Erkenntnisse gewinnen, und diese durch eine Analyse einer zweiten anderen Bounded-Until-Formel erweitern.
Im Anschluss entwickeln wir ein Konvergenzkriterium, welches erreichen soll, dass für beliebige Zeitpunkte t die Wahrscheinlichkeit abhängig von der maximalen Levelanzahl m aussagekräftig genug ist, um über die Gültigkeit einer CSL-Formel urteilen zu können.
Abschließend folgt der Vergleich zu den Ergebnissen von Remke et al. [Rem+07] in Hinblick auf Laufzeit und Iterationsanzahl.

5.1. PRISM-Modell

PRISM ist ein wahrscheinlichkeitstheoretischer Model Checker, ein Tool zur Modellierung und Analyse von randomisierten und wahrscheinlichkeitstheoretischen Modellen. Daher eignet es sich, um unser als QBD modelliertes TCP (vgl. Abbildung 4.2) zu analysieren. Wir werden aufgrund des Umfangs dieser Arbeit keine umfangreiche Einführung in PRISM geben und verweisen dafür auf weitere Literatur, wie zum Beispiel Kwiatkowska et al. [Kwi+11].
Das TCP-Modell besteht aus drei Komponenten: Packet Generator, Connection Management und der Queue. Wir haben diese Aufteilung in PRISM-Module umgesetzt und aus Konsistenzgründen die Kommentierung des Quellcodes auf Englisch vorgenommen.
Die Zustände des **Packet Generators** werden durch die Variable k dargestellt. Diese kann wie in dem QBD die Zustände 0 (off) und 1 (burst) einnehmen. Zu Beginn ist der Packet Generator im burst und produziert Pakete, wie im folgenden Listing:

```
// Packet Generator
// generates packets in burst and does nothing in off

//alpha: rate for burst to off, beta: rate for off to burst
const double alpha = 1;
const double beta = 0.04;

module packet_generator
  // State k: 0-off (idle), 1-burst (active)
  k: [0..1] init 1;

  //from off to burst
  [] k=0 -> beta: (k'=1);
  //from burst to off
  [] k=1 -> alpha: (k'=0);
endmodule
```

Listing 5.1: PRISM-TCP: Packet Generator

Das *Connection Management* ist sehr ähnlich wie der Packet Generator aufgebaut, indem es durch die Variable j repräsentiert wird und ebenfalls die Zustände 0 und 1 annehmen kann. Allerdings steht 0 dabei für released und 1 für active.
Die Variable i gibt die aktuelle Paketzahl in der *Queue* an. Sie ist beschränkt auf 0 bis m Pakete und kann nur neue Pakete aufnehmen, wenn sich der Packet Generator in burst befindet, also $k = 1$ gilt. Eine Besonderheit des Connection Managements ist es außerdem, dass der Aufbau der Verbindung nur möglich ist, wenn Pakete in der Queue vorhanden sind, das heißt die Variable i größer als 0 ist. Entsprechend darf die Verbindung auch erst abgebrochen werden, wenn keine Pakete mehr in der Queue vorhanden sind, also $i = 0$ gilt.

```
// Connection Management
// When it is active, there is a connection established and packets can be
    transmitted. Otherwise the connection is relased.

//r: rate for releasing the connection, c: rate for establishing the
    connection
const double r = 10;
const double c = 10;

module connection_management
  // State j: 0-released, 1-active
  j: [0..1] init 0;

  //establish the connection if packets are avaiable
  [] i>0 & j=0 -> c: (j'=1);
  //release the connection if there are no packets any more
  [] i=0 & j=1 -> r: (j'=0);
endmodule
```

Listing 5.2: PRISM-TCP: Connection Management

Falls bereits Pakete in der Queue vorhanden sind und das Connection Management eine Verbindung aufgebaut hat, das heißt $j = 1$ gilt, dann können Pakete verschickt und die Anzahl der Pakete in der Queue verringert werden. Den Quellcode für die Queue gibt das Listing 5.3 an:

```
1  // Transfer Queue
2  // Holds the packets waiting to be sent
3
4  // m = max_level: size of the queue (maximum level of the QBD), lambda: rate
       of generating a packet, mu: rate of transmitting a packet
5  const int m;
6  const double lambda = 100;
7  const double mu = 125;
8
9  module transfer_queue
10   // State i: number of waiting packets
11   i: [0..m] init 0;
12
13   //generate a new packet while the packet generator is active (burst). If
         possible, store it in the queue (otherwise the packet is lost)
14   [] i<m & k=1 -> lambda: (i'=i+1);
15
16   //send a packet while the connection management is active and the queue not
         empty
17   [] i>0 & j=1 -> mu: (i'=i-1);
18 endmodule
```

Listing 5.3: PRISM-TCP: Queue

Diese drei Module bilden kombiniert unser PRISM-TCP-Modell ab. Die Raten α, β, c, r, μ und λ an den Transitionen sind parametrisiert und können beliebig verändert werden. Im hier aufgeführten Quellcode sind bereits die Werte aus unserem theoretischen Modell eingesetzt (vgl. Tabelle 4.1). Die Länge der Queue kann beliebig variiert werden und ist hier noch nicht definiert. Erst bei der Analyse werden wir unterschiedliche Längen analysieren.
Außerdem müssen wir die Labelling-Funktion des theoretischen Modells übertragen. Dafür bietet PRISM speziell die Definition von Labels an, sodass der letzte Teil unseres Quellcodes wie im folgenden Listing 5.4 aussieht. Das Signalwort *label* leitet ein Label ein, danach folgt dessen Bezeichnung und nach dem ersten Gleichzeichen die Bedingung, die ein Zustand für die Eigenschaft erfüllen muss.

```
1  // Labels for defining properties of states
2  label "burst" = k=1;
3  label "off" = k=0;
4  label "active" = j=1;
5  label "released" = j=0;
```

Listing 5.4: PRISM-TCP: Labelling

Damit haben wir das Modell vollständig in PRISM implementiert. Der gesamte Quellcode ist aufgrund seiner Länge ausgelagert und im Anhang A.1 zu finden. Wenden wir uns nun der eigentlichen Analyse zu.

5.2. Problemstellung

In Abschnitt 3.3 dieser Arbeit haben wir die Uniformisierung mit Repräsentanten vorgestellt, welche das Model Checking von Formel Φ_{u1} effizienter gestalten soll. Da wir jedoch mit dem Softwaretool PRISM arbeiten und aufgrund des

Rahmens dieser Arbeit nicht PRISM um die Uniformisierung mit Repräsentanten erweitern können, verwenden wir eine eigene, simplere Approximation. Um die Unterschiede zwischen diesen Methoden zu verstehen, werden wir diese nun noch genauer erläutern.
Die Uniformisierung mit Repräsentanten nutzt aus, dass nicht die exakte transiente Wahrscheinlichkeit ausgerechnet werden muss, um zu entscheiden, ob diese größer als eine Schranke $p \in [0, 1]$ ist. Das dynamische Abbruchkriterium aus Abschnitt 3.4 iteriert deshalb nur so lange, bis eine eindeutige Aussage getroffen werden kann, vgl. Gleichungen 3.13 und 3.14.
In PRISM hingegen wird immer eine approximierte transiente Wahrscheinlichkeit berechnet, indem solange iteriert wird, bis zwischen zwei Iterationen eine geringere Abweichung als standardmäßig 10^{-6} erreicht wird, also ein Konvergenzkriterium erfüllt ist. Erst danach wird verglichen, ob diese approximierte Wahrscheinlichkeit größer als die Schranke p ist oder nicht. Wir wählen als Approximation des QBD das Abschneiden nach einem gewissen maximalen Level m. Da wir bei 0 beginnend die Level durchnummerieren, besitzt die Queue dann auch genau die Länge m. Es kann nun passieren, dass die approximierte Wahrscheinlichkeit π^m des endlichen QBDs kleiner als unsere Schranke p ist, aber die exakte Wahrscheinlichkeit des unendlichen QBDs π^∞ größer als die Schranke p. In dem Fall würde PRISM die CSL-Formel als nicht erfüllt deklarieren, da sie dies für die Approximation auch nicht ist. Unser theoretisches Modell jedoch, dem wir so nahe wie möglich kommen wollen, hätte eine gegenteilige Aussage. Analog ist es möglich, dass die Wahrscheinlichkeit π^m größer als p ist, aber π^∞ kleiner als p.
Dieses Problem können wir auch nicht durch den *a priori*-Fehler von PRISM beheben, da dieser nur eine Aussage über die Konvergenz des approximierten Modells trifft, aber keinerlei Zusammenhang zur exakten Wahrscheinlichkeit π^∞ des unendlichen QBD herstellt. Wir halten dies formell als unser *Starkes Max-Level-Problem* fest:

Definition 16 (Starkes Max-Level-Problem)**.** *Sei $P_{>p}(tt\, \mathcal{U}^{[0,t]}\, \Phi), t \in \mathbb{R}^+, \Phi \in AP$, die zu checkende CSL-Formel. Sei $m \in \mathbb{N}^+$ und π^m die approximierte PRISM-Wahrscheinlichkeit des QBD mit m Leveln. Sei π^∞ die exakte Wahrscheinlichkeit des unendlichen theoretischen QBD. Ein Starkes Max-Level-Problem tritt auf, falls: $\pi^m < p < \pi^\infty$ oder $\pi^\infty < p < \pi^m$.*

Es ist klar, dass dieses Problem nicht eindeutig identifizieren werden kann, da wir π_∞ nicht exakt berechnen können. Wir werden also keine exakte Aussage darüber treffen können, ob $\pi^\infty < p$ oder $p < \pi^\infty$ gilt. Deshalb schwächen wir die Definition wie folgt ab:

Definition 17 (Schwaches Max-Level-Problem)**.** *Sei $P_{>p}(tt\, \mathcal{U}^{[0,t]}\, \Phi), t \in \mathbb{R}^+$, $\Phi \in AP$ unsere zu checkende CSL-Formel. Sei $m \in \mathbb{N}^+$ und π^m die approximierte PRISM-Wahrscheinlichkeit des QBD mit m Leveln. Sei $n \in \mathbb{N}^+$ und π^{n+m} die approximierte PRISM-Wahrscheinlichkeit des QBD mit m+n Le-*

veln. Ein Schwaches Max-Level-Problem tritt auf, falls: $\pi^m < p < \pi^{m+n}$ *oder* $\pi^{m+n} < p < \pi^m$.

Inhaltlich bedeutet dies, dass wir schon eine Approximation des QBD mit m Leveln erhalten, aber eine Approximation mit $m + n$ Leveln eine höhere Genauigkeit in Bezug auf die exakte Wahrscheinlichkeit des unendlichen QBDs liefern würde, die eine andere Aussage über die Gültigkeit der CSL-Formel hätte. Die Häufigkeit des Vorkommens des Schwachen Max-Level-Problems werden wir nun beim eigentlichen Model Checking mit PRISM diskutieren.

5.3. Bounded-Until-Formeln

In diesem Abschnitt analysieren wir unsere Bounded-Until-Formeln, für welche wir transiente Zustandswahrscheinlichkeiten berechnen müssen. Wir möchten zuerst die Formel

$$P_{>p}(tt\, \mathcal{U}^{[0,t]} \text{ released \& no burst}), \quad t \in \mathbb{R}^+ \tag{Φ_{u1}}$$

betrachten. Dessen PRISM-Code sei nachfolgend aufgeführt. Dabei sehen wir bereits, dass die Zeitgrenze t parametrisiert ist und wir unterschiedliche Werte dafür untersuchen können.

```
1 //TCP-model: bounded-until-formula 1
2 const double t;
3
4 P=? [ F<=t ("released"&!"burst") ]
```

Listing 5.5: Implementierung von $P_{=?}(tt\, \mathcal{U}^{[0,t]}$ released & no burst) ; $t \in \mathbb{R}^+$

Um das Vorkommen des Schwachen Max-Level-Problems zu prüfen, wurden im TCP für verschiedene Zeitpunkte t die maximale Levelanzahl m schrittweise von 5 bis zu 200 erhöht und die berechneten Wahrscheinlichkeiten festgehalten. Es sei darauf hingewiesen, dass im Folgenden die Wahrscheinlichkeiten auf vier Nachkommastellen gerundet angegeben werden. Die PRISM-Ergebnisse besitzen jedoch eine Genauigkeit von ca. 16 Nachkommastellen. Da wir zum Beispiel Differenzen von zwei Wahrscheinlichkeiten betrachten möchten und diese sehr klein sein können, werden wir hier mit den 16 Nachkommastellen von PRISM rechnen und die Differenzen dann wiederum auf weniger Nachkommastellen gerundet angeben. Dadurch könnten gegebenenfalls die gerundeten Differenzen auf den ersten Blick nicht zu den gerundeten Wahrscheinlichkeiten passen. Die exakten Ergebnisse sind jedoch korrekt und alle exakten Wahrscheinlichkeiten sind im Anhang aufgeführt.

Wir beginnen mit den Ergebnissen, die in Abbildung 5.1 für $t = 0.5$ visualisiert sind. Bei einer niedrigen maximalen Levelanzahl von $m = 5$ erreichen wir die höchste Wahrscheinlichkeit von ungefähr 0.3312. Bis zu $m = 25$ nimmt diese Wahrscheinlichkeit relativ schnell ab bis zu einem Wert von 0.2967.
Betrachten wir nun den Fall $p = 0.3$, also die CSL-Formel

$$P_{>0.3}(tt\, \mathcal{U}^{[0,0.5]} \text{ released \& no burst}).$$

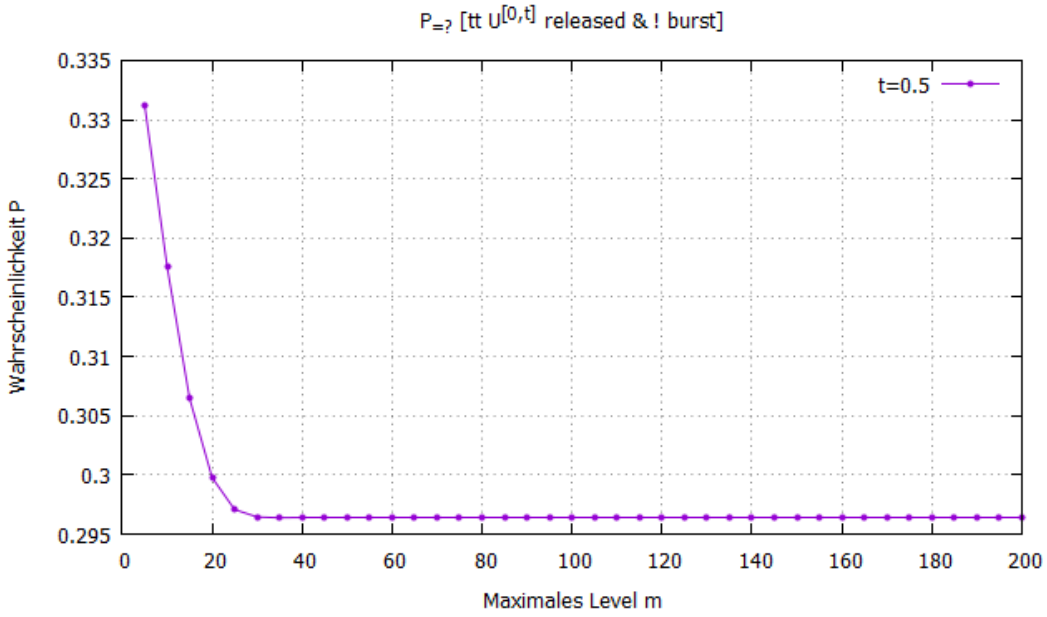


Abbildung 5.1.: $P_{=?}(tt\ \mathcal{U}^{[0,t]}$ released & no burst), $t = 0.5$

Dann erhalten wir offensichtlich ein wie oben definiertes Schwaches Max-Level-Problem, da für m = 5 die Formel erfüllt ist, für m = 25 jedoch nicht. Letzteres würde aber viel mehr der Aussage des unendlichen QBDs entsprechen, da wir ebenfalls in der Abbildung 5.1 sehen können, dass die Wahrscheinlichkeit monoton abnimmt und ab $m = 40$ in etwa konstant bleibt. Im Anhang A.4 sind die exakten Ergebnisse ausgeführt, bei denen man sogar ab $m = 60$ trotz einer Genauigkeit von 16 Nachkommastellen keinerlei Veränderungen mehr in den PRISM-Wahrscheinlichkeiten hat.
Inhaltlich ist diese Entwicklung nachvollziehbar, da der Zustand *released & no burst* nur eintreten kann, nachdem alle Pakete versendet und während des Time-Outs keine neuen Pakete produziert wurden. Je größer m ist, umso mehr Pakete können jedoch in der Queue gespeichert werden, also um so mehr Pakete müssen abgearbeitet werden, bis der Zustand möglich wird. Informell könnte man sagen, dass der Weg zum Zustand *released & no burst* länger wird, wenn der QBD länger wird.

Um dieses Verhalten zu verifizieren und auch den Einfluss des Zeitpunkts t zu berücksichtigen, untersuchen wir nun die Ergebnisse für $t = 1$ in Abbildung 5.2. Der Kurvenverlauf ist ähnlich zu dem für $t = 0.5$. Er beginnt für $m = 5$ mit der höchsten Wahrscheinlichkeit von 0.5924 und nimmt dann bis etwa $m = 40$ relativ stark ab. Im Vergleich zu $t = 0.5$ ist dieses Abnehmen aber gemäßigter und die Konvergenz tritt erst bei einem höheren maximalen Level ein. Konkret verändert sich die Wahrscheinlichkeit für $m = 95$ nicht mehr und behält dann konstant einen Wert von 0.5637, vgl. A.5.
Insbesondere sind aber für alle m bei $t = 1$ die Wahrscheinlichkeiten höher als

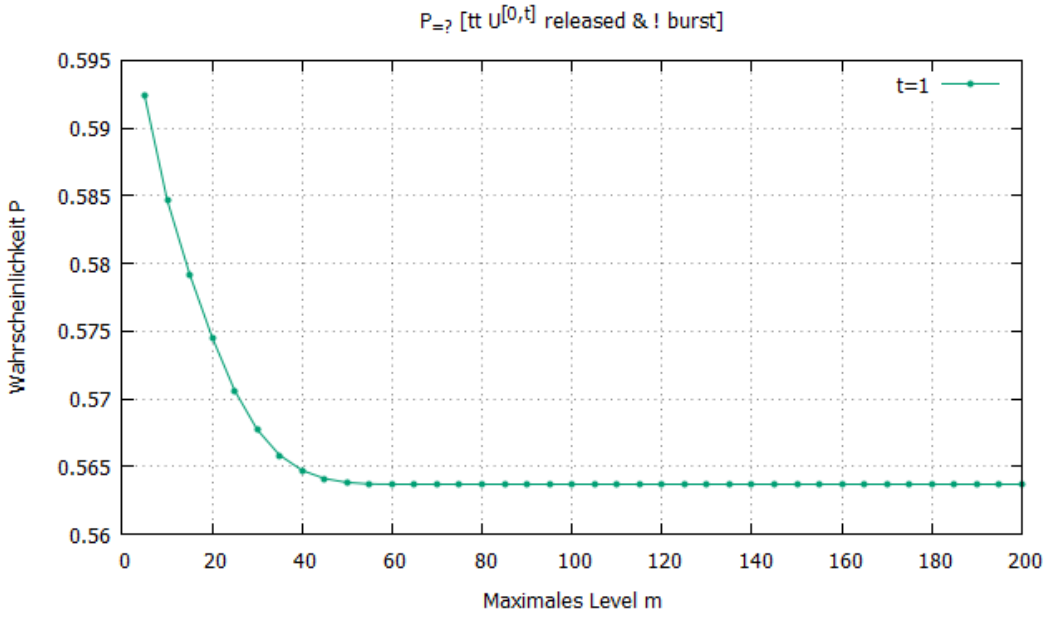


Abbildung 5.2.: $P_{=?}(tt\,\mathcal{U}^{[0,t]}$ released & no burst), $t = 1$

für $t = 0.5$. Dies ist insofern stimmig, da für ein höheres t mehr Transitionen gewählt werden können und somit längere Wege in dem QBD möglich sind. Da für den Zustand *released & no burst* diese Länge wesentlich im Zusammenhang zu den maximalen Leveln steht, wird die Wahrscheinlichkeit für höhere t insgesamt größer.
Die Ergebnisse für $t = 2$ in Abbildung 5.3 unterstützen diese Aussagen. Der Kurvenverlauf ist wieder monoton fallend, aber im Vergleich zu $t = 0.5$ und $t = 1$ gemäßigter. Mit 0.8494 für $m = 5$ wird die höchste Wahrscheinlichkeit erzielt, bis sie ab etwa $m = 65$ relativ konstant wird und sich ab $m = 160$ nicht mehr verändert und gegen den Wert 0.8385 konvergiert. Die Wahrscheinlichkeiten für alle m sind im Vergleich zu $t = 1$ wieder deutlich höher.

In der Analyse wurden auch die Ergebnisse für $t = 3$ und $t = 4$ untersucht, diese werden wir jedoch nicht explizit grafisch separiert analysieren, da sie unsere bisherigen Aussagen unterstützen und keine anderen Erkenntnisse liefern als unsere nachfolgende Analyse von $t = 5$. In Hinblick auf Vollständigkeit sind die Ergebnisse jedoch noch im Anhang unter A.7 und A.8 dieser Arbeit hinzugefügt.
Für $t = 5$ sehen wir, dass sich das abnehmende Verhalten der Kurve deutlich abgeschwächt hat. Die höchste Wahrscheinlichkeit in Abbildung 5.4 beträgt ungefähr 0.9924 und erst ab $m = 80$ wird die Wahrscheinlichkeit annähernd konstant. Dabei sagen wir betont *annähernd konstant*, da im Gegensatz zu $t = 0.5$ bis $t = 2$ bis zu einem maximalen Level von $m = 200$ immer noch Veränderungen in den PRISM-Wahrscheinlichkeiten auftreten.

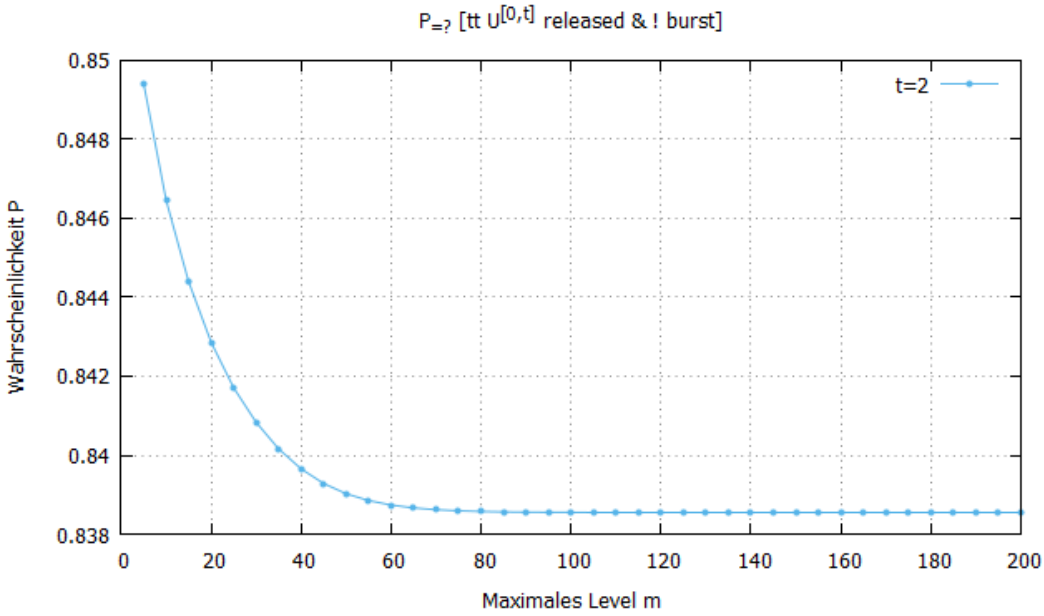


Abbildung 5.3.: $P_{=?}(tt\,\mathcal{U}^{[0,t]}$ released & no burst), $t = 2$

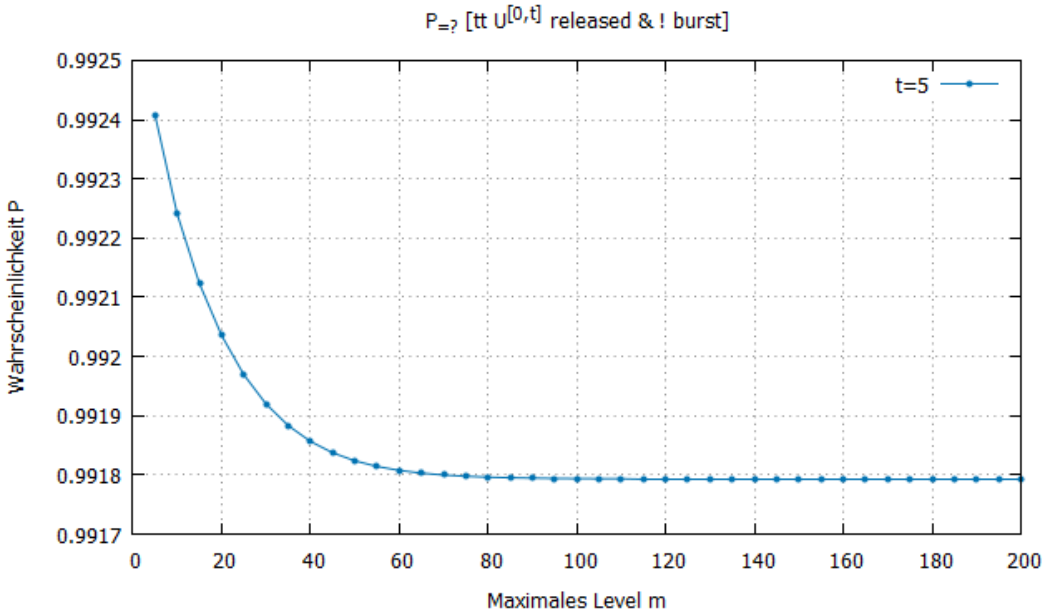


Abbildung 5.4.: $P_{=?}(tt\,\mathcal{U}^{[0,t]}$ released & no burst), $t = 5$

Die Veränderungen sind zwar vergleichsweise minimal, dennoch zeigen sie, dass für größere t auch erst für viel größere maximale Level m die Wahrscheinlichkeit konstant wird.
Es wurden zur Bestätigung der Minimalität dieser Abweichungen für $t = 5$ bis zu $m = 300$ Ergebnisse berechnet und die Differenz der Wahrscheinlichkeit für $m = 200$ und $m = 300$ überprüft. Diese ist in der Größenordnung kleiner als 10^{-10}, siehe A.9.
Der absolute Wert für $m = 300$ beträgt ungefähr 0.9918. Wenn wir nun in Betracht ziehen, dass wir für $m = 5$ bereits eine Wahrscheinlichkeit von 0.9924 erhalten, so wird deutlich, dass hier die Differenz in Bezug auf die maximalen Level ebenfalls minimal ist. Für $t = 0.5$ war dies aber noch nicht so, da dort der höchste Wert immerhin 0.3312 und der niedrigste 0.2964 betrug, also die Differenz deutlich größer war. Klarer sieht man dies im konkreten Vergleich der verschiedenen Zeitpunkte t, wie er in Abbildung 5.5 gegeben ist.

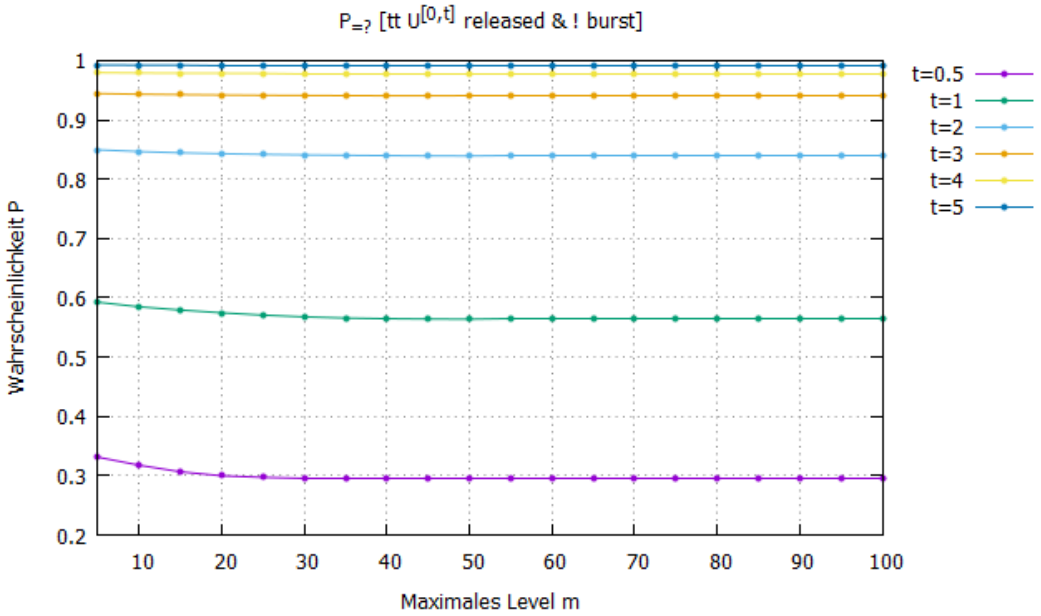


Abbildung 5.5.: $P_{=?}(tt\ \mathcal{U}^{[0,t]}$ released & no burst), $t \in \{0.5, 1, 2, 3, 4, 5\}$

Da wir bereits wissen, dass für $t \in \{0.5, 1, 2, 3, 4, 5\}$ ab $m = 100$ keine gravierenden Änderungen mehr eintreten und wir uns mehr für die Anfangsdifferenz interessieren, werden hier nur Kurven bis $m = 100$ dargestellt. Wir nun eine viel größere Wahrscheinlichkeitsspanne durch die verschiedenen Kurven darstellen, daher wurde die Skalierung angepasst und sollte entsprechend beachtet werden.
Wie schon erwartet, nimmt die maximale Wahrscheinlichkeitsdifferenz für größer werdende t immer weiter ab. Während wir für $t = 0.5$ und $t = 1$ noch ein eindeutiges Abfallen der Kurve zwischen $m = 5$ und $m = 30$ festhalten können, ist für $t = 5$ kaum noch eine Differenz erkennbar. Dies führt uns zum

zweiten Analyseteil unserer Bounded-Until-Formel.

Bisher haben wir für feste t das maximale Level m variiert. Wenn wir jedoch zurück an das Schwache Max-Level-Problem denken, so war dies bei $t = 0.5$ leicht für $p = 0.3$ zu identifizieren. Für $t = 5$ ist es aber seltener ein solches Schwaches Max-Level-Problem zu erzeugen, da die Wahrscheinlichkeitsdifferenzen geringer sind. Man müsste beispielsweise $p = 0.992$ wählen, was vergleichsweise sehr spezifisch ist.
Insgesamt wünschenswert wäre es also, bei der Erhöhung von t das Auftreten eines Schwachen Max-Level-Problems immer weiter eingrenzen zu können. Daher werden wir nun m fest wählen, aber den Zeitpunkt t im Bereich von 0 bis 12 variieren, um die Wahrscheinlichkeitsdifferenzen abhängig von t zu untersuchen. Zu beachten ist dabei, dass wir für jedes t gewährleisten möchten, dass wir ein m gewählt haben, bei dem die Wahrscheinlichkeit für höhere m kaum noch variiert, das heißt zumindest annähernd konstant ist.
Deshalb haben wir für das maximale $t = 12$ nochmal m variiert und festgestellt, dass sich die Wahrscheinlichkeit ab etwa $m = 500$ höchstens minimal verändert, siehe A.10.

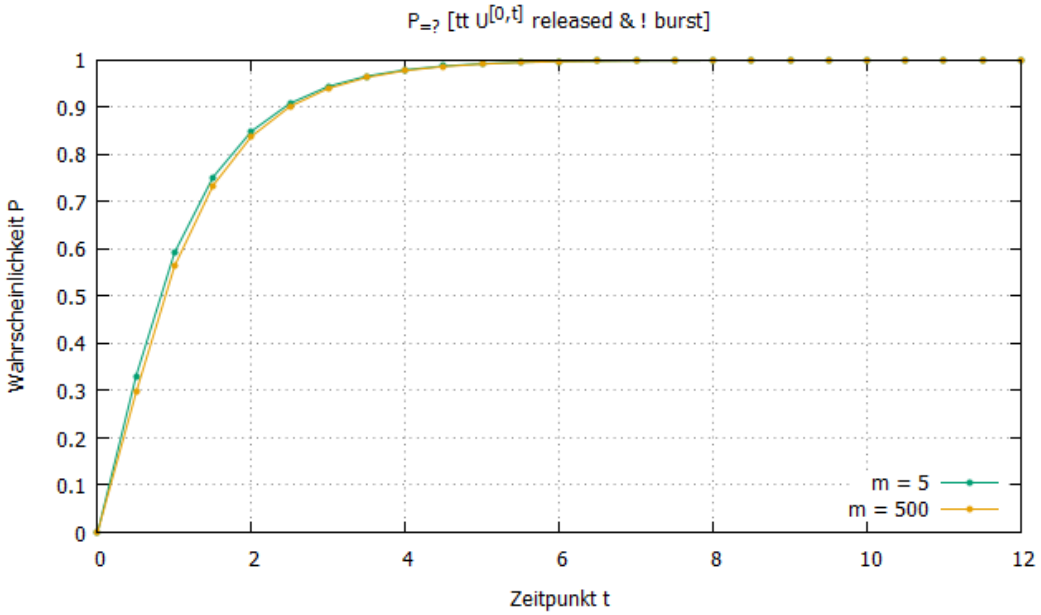


Abbildung 5.6.: $P_{=?}(tt\,\mathcal{U}^{[0,t]}$ released & no burst),
$m \in \{5, 500\}, t \in \{0.5, 1, ..., 12\}$

In Abbildung 5.6 sind für die Zeitpunkte von $t = 0$ bis $t = 12$ in 0.5er-Schritten die Kurven für $m = 5$ und $m = 500$ aufgeführt. Beide zeigen einen sehr ähnlichen Verlauf und sind monoton steigend, was unsere bisherigen Analysen unterstützt. Interessant ist jedoch, dass die Differenz der Wahrscheinlichkeiten durch die Anzahl der maximalen Leveln, also der Länge des QBD, für kleine t relativ groß und für größere t ab einem Wert von ungefähr 6 sehr klein ist.

In diesem Zusammenhang könnte man also durchaus behaupten, dass ein Auftreten des Schwachen Max-Level-Problems für größere t deutlich seltener ist. Als Fazit können wir damit ziehen, dass für unsere CSL-Formel Φ_{u1} für aussagekräftige Ergebnisse bei kleinen t deutlich höhere maximale Levelanzahlen m betrachtet werden müssen als für große t. Für kleine t hingegen sollte man deshalb zum Beispiel die Wahrscheinlichkeitsdifferenzen des QBD der Länge m und des QBD der Länge $m+1$ vergleichen. Falls diese Differenz kleiner als ein gewisser Fehler ist, kann die Wahrscheinlichkeit als aussagekräftig angesehen werden. Für größere t hingegen kann man bereits Wahrscheinlichkeiten von sehr kleinen maximalen Levelanzahlen als aussagekräftig ansehen.
Wenn wir uns nun ins Gedächtnis rufen, dass wir meistens gar keine exakten Wahrscheinlichkeiten benötigen, sondern nur

$$P_{\bowtie p}(tt\, \mathcal{U}^{[0,t]} \text{ released \& no burst}),\ \bowtie \in \{<, >, \leq, \geq\}$$

prüfen möchten, so ist die Wahrscheinlichkeitsdifferenz von keiner sehr großen Bedeutung. Im konkreten Fall dieser Formel wissen wir bereits, dass die Wahrscheinlichkeiten für größere m monoton steigend ist, dass heißt für

$$P_{\bowtie p}(tt\, \mathcal{U}^{[0,t]} \text{ released \& no burst}),\ \bowtie \in \{>, \geq\}$$

können wir auch schon für ein kleines m eine Aussage treffen, falls die Wahrscheinlichkeit für ein kleines m schon größer(gleich) p ist. Dies ist aber problematisch, falls sie es nicht ist. Denn in diesem Fall wissen wir nicht, ob die Wahrscheinlichkeit des unendlichen QBD kleiner als p ist und wir deshalb auch für große m die Schranke nicht erreichen oder ob wir einfach nur das m erhöhen müssen, um die Schranke zu erreichen.
Daraus folgt, dass wir in diesem Fall die Wahrscheinlichkeitsdifferenzen betrachten müssen, da wir ohne sie kein Wissen darüber haben, ob eine Formel noch erfüllt werden kann. Dies ist ein gravierender Nachteil gegenüber dem dynamischen Abbruchkriterium, bei dem der maximale Fehler abhängig von der aktuellen Iterationsanzahl ist und konkret berechnet werden kann.

Unsere letzten Erkenntnisse über Wahrscheinlichkeitsdifferenzen haben wir nur auf die Bounded-Until-Formel Φ_{u1} bezogen. Es könnte aber auch sein, dass die Wahrscheinlichkeitsdifferenz für größere t so gering ist, weil die Wahrscheinlichkeit selbst einfach schon sehr dicht an 1 ist und deshalb nicht mehr viel Spielraum für Differenzen da ist.
Deshalb haben wir als zweite zu prüfende Bounded-Until-Formel eine Formel ausgewählt, bei der wir erwarten, dass für verschiedene t auch die Wahrscheinlichkeit stärker variiert und möchten diese prüfen. Die Formel sollte sich ähnlich zu Formel Φ_{u1} verhalten, damit wir einen Vergleich ziehen können, und soll nun hergeleitet werden.
Damit überhaupt eine Wahrscheinlichkeitsvarianz für verschiedene t vorkommt, müssen auch Levelwechsel stattfinden, daher betrachten wir, wann wieder der Zustand *no burst* erreicht wird. Es bietet sich nun an, das Connection Management im Zustand active zu untersuchen. *active & no burst* ist innerhalb von

Level 0 nicht erreichbar, aber sobald ein Levelwechsel vollzogen wurde, kann der Zustand entweder von einer Transition mit Rate $c = 10$ ausgehend von *released & no burst* erreicht werden oder erst nach der Rückkehr zu Level 0. Ersteres ist relativ unwahrscheinlich, da dann der Packet Generator erst mit Rate $\alpha = 1$ in den Ruhemodus wechseln müsste, bevor das Connection Management mit Rate $\lambda = 100$ aktiv wird. Für die Rückkehr zu Level 0 müssen Level durchlaufen werden, bis die Queue geleert wird. Je mehr Level durchquert werden müssen, umso mehr Zeit ist aber nötig, damit so viele Transitionen gewählt werden. In diesem Kontext ist es also nachvollziehbar, dass für größere t auch die Wahrscheinlichkeiten größer werden. Damit haben wir ein ähnliches Verhalten wie bei Φ_{u1} und werden im Folgenden diese Formel untersuchen:

$$P_{=?}(tt\, \mathcal{U}^{[0,t]} \text{ active } \& \text{ no burst}),\ t \in \mathbb{R}^+ \qquad (\Phi_{u2})$$

Die Implementierung der Formel in PRISM findet sich im Anhang A.3.
Beginnen wir mit unserer Analyse für $t = 0.5$.

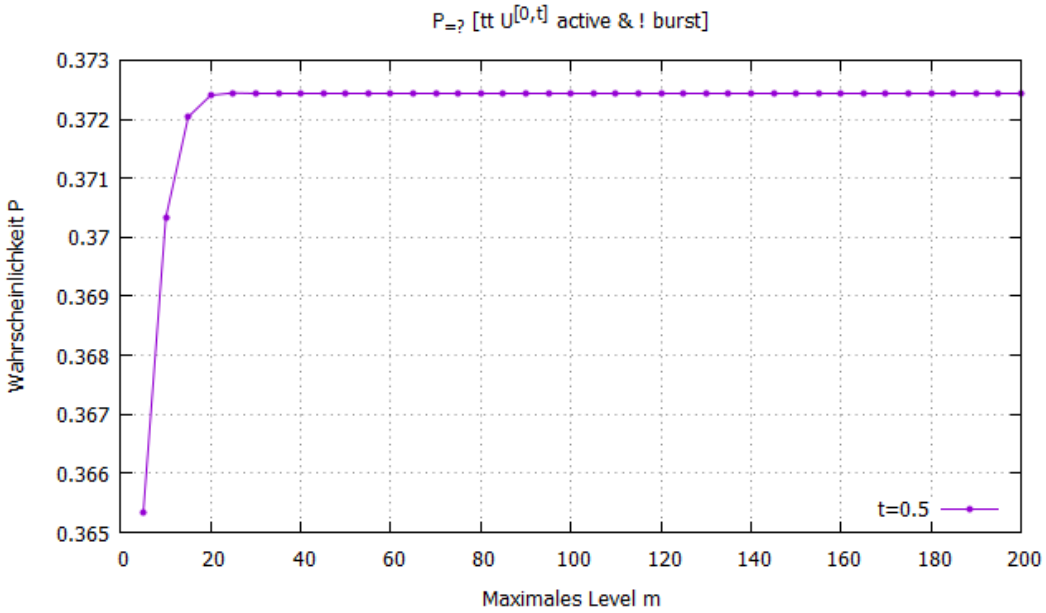


Abbildung 5.7.: $P_{=?}(tt\, \mathcal{U}^{[0,t]} \text{ active } \& \text{ no burst}),\ t = 0.5$

Wir sehen direkt, dass wir wieder den typischen Knick in der Kurve haben, weil die Wahrscheinlichkeiten für kleine m noch nicht sehr exakt sind. Allerdings sind diesmal die Wahrscheinlichkeiten monoton steigend, was ein Unterschied zu Formel Φ_{u1} ist. Ebenfalls ist hier für ein kleines t die Wahrscheinlichkeitsdifferenz deutlich kleiner. Bei Formel Φ_{u1} in Abbildung 5.1 war eine Toleranz von ungefähr 0.03483 zu verzeichnen. Hier sie tritt sie eher im Rahmen von etwa 0.00711 auf, da wir einen Minimalwert von 0.3653 und für etwa $m = 50$ konstant 0.3724 erhalten. Ein Schwaches Max-Level-Problem würde in diesem Fall bereits für $t = 0.5$ deutlich seltener auftreten als bei der ersten Formel.

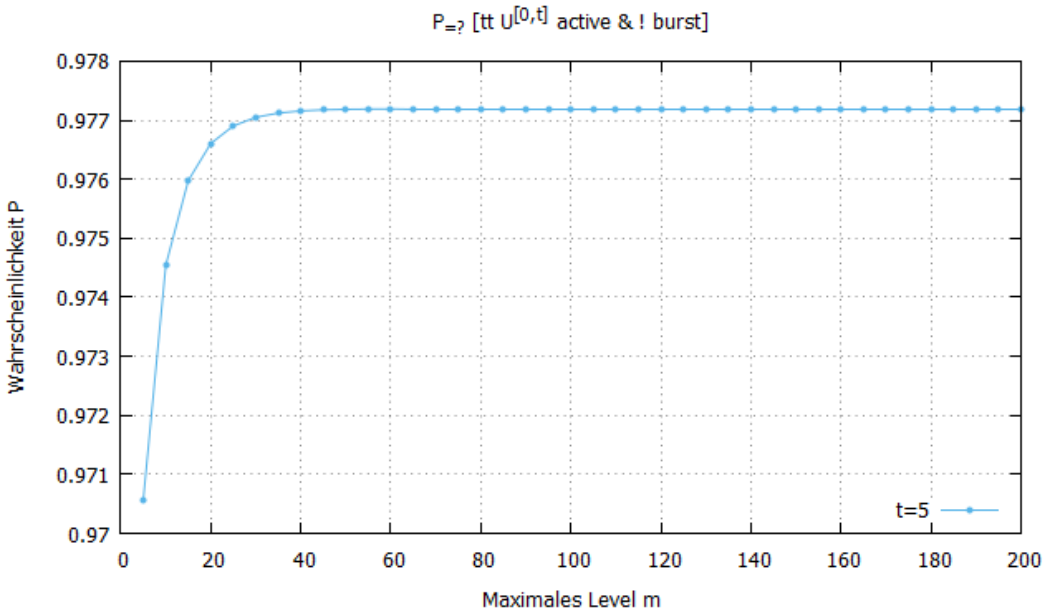


Abbildung 5.8.: $P_{=?}(tt\,\mathcal{U}^{[0,t]}$ active & no burst), $t = 5$

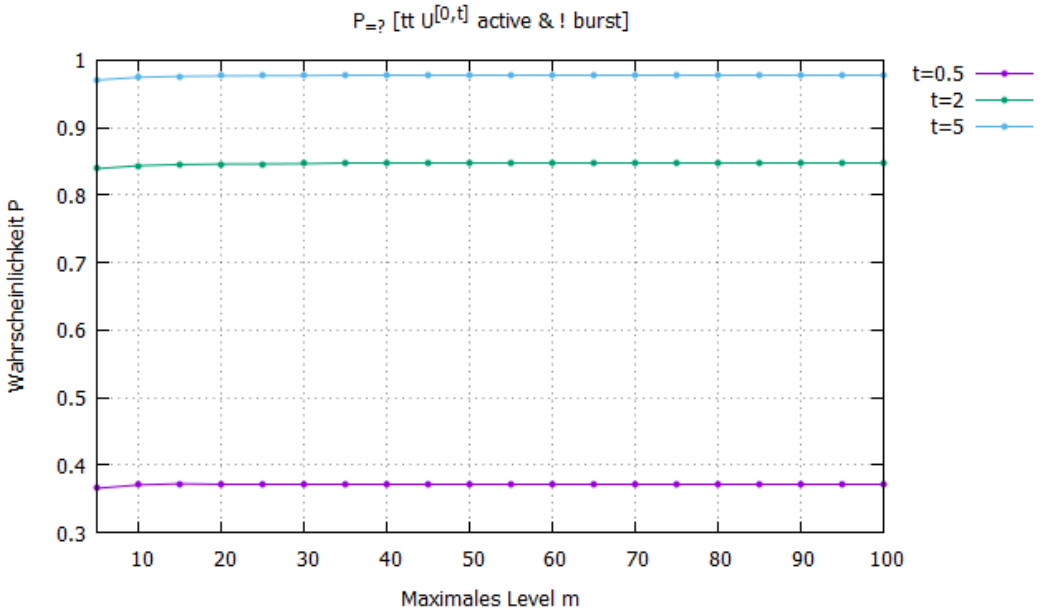


Abbildung 5.9.: $P_{=?}(tt\,\mathcal{U}^{[0,t]}$ active & no burst), $t \in \{0.5, 2, 5\}$

Um trotzdem auch das Verhalten für große t zu analysieren, haben wir auch die Werte für $t = 5$ gemessen. Der Minimalwert beträgt ungefähr 0.9706 und der Höchstwert 0.9772, was wiederum eine Differenz von etwa 0.00662 ergibt, siehe Anhang A.16. Da Rundungsfehler nicht auszuschließen sind, ist hier keine deutlich höhere oder niedrigere Wahrscheinlichkeitsdifferenz als für $t = 0.5$ zu erkennen. Es ist jedoch anzumerken, dass erst für $m = 190$ die Wahrscheinlichkeit endgültig konstant wird. Der Gesamtvergleich, zu dem wir auch die Ergebnisse für $t = 2$ hinzugenommen haben, ist in Abbildung 5.9 zu sehen.
Aufgrund der geringen Wahrscheinlichkeitsdifferenzen ist dabei nur der relevante Wertebereich bis $m = 100$ visualisiert worden.
In dieser Grafik wird nun deutlich, dass die Wahrscheinlichkeitsdifferenzen für verschiedene t kaum variieren, also eher unabhängig davon sind, siehe Tabelle 5.1. Das ist ein anderes Ergebnis, als wir es mit der ersten Until-Formel Φ_{u1} erzielt haben. Außerdem wird für die zweite Until-Formel Φ_{u2} für große t eben-

Zeitpunkte t	0.5	2	5
$P_{=?}(tt\,\mathcal{U}^{[0,t]}$ released & no burst)	0.03483	0.01086	0.00061
$P_{=?}(tt\,\mathcal{U}^{[0,t]}$ active & no burst)	0.00711	0.00744	0.00662

Tabelle 5.1.: Wahrscheinlichkeitsdifferenz von maximaler und minimaler Wahrscheinlichkeit abhängig vom Zeitpunkt t

falls auch erst für große m die Wahrscheinlichkeit konstant. Diese sind zwar kleiner als die für die erste Until-Formel Φ_{u1}, aber verhalten sich ansonsten ähnlich. Die Tabelle 5.2 stellt dies übersichtlich zusammen:

Zeitpunkte t	0.5	2	5
$P_{=?}(tt\,\mathcal{U}^{[0,t]}$ released & no burst)	60	160	>300
$P_{=?}(tt\,\mathcal{U}^{[0,t]}$ active & no burst)	50	115	190

Tabelle 5.2.: maximale Levelanzahl m abhänging vom Zeitpunkt t, bei der die Wahrscheinlichkeit P konstant wird

Unsere vorherige Vermutung, dass für große und kleine t deutlich unterschiedliche maximale Levelanzahlen m nötig sind, ist hiermit für die zweite Until-Formel Φ_{u2} widerlegt, da die Wahrscheinlichkeitsdifferenz kaum variiert.
Man könnte nun noch anhand weiterer Formeln untersuchen, ob zumindest für große t immer kleinere maximale Levelanzahlen m nötig sind. Allerdings ist es sehr relativ, was als *großes t* betrachtet wird. Aufgrund des Umfangs dieser

Arbeit werden wir deshalb keine weiteren CSL-Formeln prüfen, sondern uns auf unsere bisherigen Erkenntnisse beschränken.
Zusammengefasst haben wir also feststellen können, dass wir ohne Betrachtung der Wahrscheinlichkeitsdifferenzen kaum eine sichere Aussage über die Gültigkeit einer CSL-Formel treffen können. Dies liegt daran, dass wir im Gegensatz zur Methodik von Remke et al. [Rem+07] nicht wissen, ob Wahrscheinlichkeiten für größere m steigen oder fallen. Remke et al. [Rem+07] hingegen konnten davon ausgehen, dass für mehr Iterationen die Wahrscheinlichkeit nur noch steigen kann. Man konnte dort ebenfalls einen absoluten Fehler berechnen und diesen in die Ergebnisse mit einbeziehen. Wir haben mit PRISM und unserer simplen Approximation diese Werkzeuge nicht.
Deshalb werden wir nun die Wahrscheinlichkeitsdifferenzen nutzen, um abhängig vom maximalen Level m zu definieren, ob Wahrscheinlichkeiten aussagekräftig genug sind, um mit ihnen über die Gültigkeit einer CSL-Formel urteilen zu können. Dies geschieht in Form eines Konvergenzkriteriums, welches wir im nächsten Abschnitt vorstellen werden.

5.4. Konvergenzkriterium

Wir werden mit unseren aktuellen Werkzeugen keine exakte Aussage über die Gültigkeit einer CSL-Formel treffen können, weil wir die exakte Wahrscheinlichkeit des unendlichen QBD nicht berechnen können. Ebenfalls haben wir weniger Wissen über das Verhalten unserer Wahrscheinlichkeiten als wir es durch die Uniformisierung mit Repräsentanten hätten.
Wir möchten aber trotzdem eine Aussage über die Gültigkeit einer CSL-Formel treffen können und dafür PRISM verwenden. Daher definieren wir, wann für uns Wahrscheinlichkeiten abhängig von der maximalen Levelanzahl m konvergieren. Dies sei von der Konvergenz im mathematischen Kontext zu trennen, da wir im Folgenden nicht formell beweisen können, dass unsere Wahrscheinlichkeiten im mathematischen Kontext konvergieren. Dennoch werden wir unser Kriterium *Konvergenzkriterium* nennen, da es inhaltlich eine ähnliche Aussage wie das Kriterium der mathematischen Konvergenz besitzt.

Definition 18 (Konvergenzkriterium)**.** *Sei* $step \in \mathbb{N}^+$ *eine Schrittgröße,* $m \in \mathbb{N}^{>step}$ *eine maximale Levelanzahl,* $t \in \mathbb{R}^+$ *ein Zeitpunkt und* $\varepsilon \in (0,1)$ *ein absoluter Fehler. Sei* $\mathcal{Q}$ *unser endlicher QBD mit der maximalen Levelanzahl* m*,* Φ *eine levelunabhängige AP und* π^m *die Wahrscheinlichkeit für eine Formel der Art*

$$P_{\bowtie p}(tt\ \mathcal{U}^{[0,t]}\ \Phi), \quad \bowtie \in \{<, >, \leq, \geq\}.$$

Wir sagen, die Wahrscheinlichkeit für $\mathcal{Q}$ *mit der maximalen Levelanzahl* m *konvergiert, falls*

$$|\pi^m - \pi^{m-step}| < \varepsilon$$

gilt.

Intuitiv haben wir ein maximales Level m und eine Schrittweite und prüfen, ob die Wahrscheinlichkeiten für m und für einen Schritt weiter dicht genug beieinander liegen, das heißt ihre Wahrscheinlichkeitsdifferenz klein genug ist. Wir haben dieses Konvergenzkriterium auch als Python-Skript implementiert, um es in der Praxis testen zu können.
Die Iteration findet in der Methode

`computeProb(model,properties,result,m,steps,eps,t)`

statt. Die Eingabeparameter sind folglich eine PRISM-Datei mit dem Modell, eine PRISM-Datei mit der CSL-Formel und eine txt-Datei, in die die Ergebnisse geschrieben werden. Außerdem erwarten wir als Eingabe das erste maximale Level m, die Schrittgröße steps für die Iteration, eps für den absoluten Fehler und einen beliebigen, aber festen Zeitpunkt t. Das folgende Listing 5.6 zeigt den Iterationsanfang.

```
1  def computeProb(model,properties,result,m,steps,eps,t):
2    #--------------------------------------------------------------------------
3    # building and executing of prism command - beginning step
4    #--------------------------------------------------------------------------
5    n = m+steps      #next iteration of m
6    probm = 0      #default probability for level m
7    probn = 1      #default probability for level n
8    #create empty resultfile
9    resultfile = open(result,"w")
10   resultfile.close()
11
12   #open file for computation
13   resultfile = open(result,"r+")
14   #prism computation
15   prismcommand = "prism " + model + " " + properties + " -const m=" + str(m) +
          ":" + str(steps) +":" + str(n) + ",t=" + str(t) + " -exportresults " +
         result
16   prismprocess = subprocess.check_call(prismcommand, stdin=None, stdout=None,
         stderr=None, shell=True)
17
18   #reading of resultfile
19   content = resultfile.read()
20   probm = float(content.split( )[3])
21   probn = float(content.split( )[5])
22   probdiff = abs(probm - probn)
23   resultfile.close()
```

Listing 5.6: Konvergenzkriterium: Iterationsanfang

Es wird der nächste Iterationsschritt für das maximale Level ausgerechnet und eine Standardwahrscheinlichkeit für m und den nächsten Schritt gesetzt. Dann wird die result-Datei erzeugt, falls sie noch nicht existiert. Anschließend wird ein PRISM-Befehl aus den übergebenen Parametern und den Kommandozeilenargumenten der PRISM-Kommandozeilenversion gebaut. Mithilfe des Pythonmoduls subprocess kann dann ein neuer Prozess erzeugt werden, der die Berechnung durch den Aufruf der PRISM-Kommandozeilenversion durchführt. Die Ergebnisse werden aus der Ergebnisdatei ausgelesen und verglichen. Der nächste Iterationsschritt ist in Listing 5.7 aufgeführt.

```
1  #-------------------------------------------------------------------------
2  # iterating until the probability difference is smaller than a-priori eps
3  #-------------------------------------------------------------------------
4  while probdiff >= eps:
5    #iteration of m and n
6    m = n;
7    n += steps;
8
9    #open file for computation
10   resultfile = open(result,"r+")
11   prismcommand = "prism " + model + " " + properties + " -const m=" + str(n)
         + ",t=" + str(t) + " -exportresults " + result
12   prismprocess = subprocess.check_call(prismcommand, stdin=None, stdout=None
         , stderr=None, shell=True)
13
14   #reading of resultfile
15   content = resultfile.read()
16   print(content)
17   probm = probn
18   probn = float(content.split( )[1])
19   probdiff = abs(probm - probn)
20   resultfile.close()
21
22 #store final m and final probability (difference)
23 resultfile = open(result,"w")
24 resultfile.write("\n" + "convergence for m = " + str(m) + "\n"+ "probability
       difference = " + str(probdiff) + "\n" + "probability = " + str(probn))
25 resultfile.close()
```

Listing 5.7: Konvergenzkriterium: Iterationsschritt

Wir prüfen für jede Iteration, ob die Wahrscheinlichkeitsdifferenz bereits kleiner als unser absoluter Fehler eps ist. Falls nicht, so iterieren wir weiter. Dabei wiederholen wir ihm Prinzip denselben Schritt wie schon beim Iterationsanfang. Der einzige Unterschied ist der, dass wir nun nicht zwei Anfangswahrscheinlichkeiten ausrechnen, sondern immer nur eine neue Wahrscheinlichkeit, die wir mit der Wahrscheinlichkeit der letzten Iteration vergleichen.
Sobald unsere Iteration endet, wird das maximale Level m, für das das Konvergenzkriterium erfüllt ist, dessen Wahrscheinlichkeit und die Wahrscheinlichkeitsdifferenz $|\pi^m - \pi^{m-step}|$ in unsere Ergebnisdatei geschrieben.
Für eine bessere Benutzerfreundlichkeit wurde das Skript um eine Eingabe der Parameter über die Kommandozeile erweitert. Der vollständige Quellcode ist im Anhang A.17 zu finden.
Im nächsten und letzten Abschnitt dieses Kapitels werden wir unser Skript in Bezug auf die Laufzeit testen und einen Vergleich zu den Ergebnissen von Remke et al. [Rem+07] ziehen.

5.5. Laufzeit und Iterationsanzahl

In der Analyse von Remke et al. [Rem+07] wurde folgende CSL-Formel überprüft:

$$P_{\geq p}(tt\ \mathcal{U}^{[0,t]}\ \text{released \& no burst}), \quad t \in \{0.5, 1, 2\} \qquad (\Phi_{u3})$$

Wir werden zwar unabhängig von einer Wahrscheinlichkeitsschranke p Wahrscheinlichkeiten berechnen und daher im Folgenden

$$P_{=?}(tt\ \mathcal{U}^{[0,t]}\ \text{released \& no burst}), \quad t \in \{0.5, 1, 2\} \qquad (\Phi_{u3exakt})$$

betrachten, aber dennoch einen Vergleich zwischen den Formeln ziehen.
Im letzten Abschnitt haben wir ein Pythonskript vorgestellt, mit dem wir ein maximales Level m berechnen können, für das die Wahrscheinlichkeit der jeweiligen Formel aussagekräftig ist. Wir haben deshalb unser Skript mit der Formel $\Phi_{u3exakt}$ für $t \in \{0.5, 1, 2\}$ getestet und die Ergebnisse in Tabelle 5.3 aufgeführt. Wahrscheinlichkeiten und ihre Differenzen wurden dabei auf vier Nachkommastellen gerundet. Als Einstellungen haben wir einen maximalen Fehler von $\varepsilon = 10^{-4}$ und eine Schrittgröße von 1 gewählt und unsere Iterationen für $m = 1$ begonnen.

Zeitpunkte t	0.5	1	2
Maximales Level m	28	44	38
W'keitsdifferenz	$7.617 \cdot 10^{-5}$	$9.107 \cdot 10^{-5}$	$9.528 \cdot 10^{-5}$
Wahrscheinlichkeit	0.2965	0.5641	0.8397

Tabelle 5.3.: Ergebnisse des Pythonskripts für Formel $\Phi_{u3exakt}$, $\varepsilon = 10^{-4}$

Wir können direkt erkennen, dass sich das maximale Level m nicht klar in Relation zum Zeitpunkt t verhält. Zur Überprüfung dieser Ergebnisse haben wir unser Skript nochmal mit denselben Einstellungen bis auf den absoluten Fehler, welcher nun $\varepsilon = 10^{-3}$ beträgt, laufen lassen.

Zeitpunkte t	0.5	1	2
Maximales Level m	19	15	3
W'keitsdifferenz	$9.653 \cdot 10^{-4}$	$9.779 \cdot 10^{-4}$	$9.637 \cdot 10^{-4}$
Wahrscheinlichkeit	0.2998	0.5782	0.8502

Tabelle 5.4.: Ergebnisse des Pythonskripts für Formel $\Phi_{u3exakt}$, $\varepsilon = 10^{-3}$

In diesem Fall ließe sich sogar vermuten, dass die nötige maximale Levelanzahl m kleiner wird, wenn t größer wird, wie in Tabelle 5.4 zu sehen. Das lässt sich allerdings auch auf den kleineren absoluten Fehler zurückführen. Wir wissen

bereits aus unserer Analyse, dass für große t die Wahrscheinlichkeitsdifferenzen kleiner waren, aber dennoch nicht deutlich früher konstant wurden. Eventuell ist daher eine Genauigkeit von $\varepsilon = 10^{-3}$ nicht ausreichend. Deshalb haben wir erneut das Skript laufen lassen, diesmal für $\varepsilon = 10^{-5}$, siehe Tabelle 5.5.

Zeitpunkte t	0.5	1	2
Maximales Level m	33	55	66
W'keitsdifferenz	$5.897 \cdot 10^{-6}$	$9.791 \cdot 10^{-6}$	$9.863 \cdot 10^{-6}$
W'keitsdifferenz	0.2964	0.5637	0.8386

Tabelle 5.5.: Ergebnisse des Pythonskripts für Formel $\Phi_{u3exakt}$, $\varepsilon = 10^{-5}$

Für diesen absoluten Fehler steigt das maximale Level m mit den Zeitpunkten t an. Es kann also auch bei unserem Skript nicht klar gesagt werden, wie sich m in Relation zu t verhält. Welcher absolute Fehler ε gewählt wird, sollte deshalb von der CSL-Formel und dessen Anwendung abhängig gemacht werden. Wenn wir wissen, dass wir eine Schranke p mit nur einer Nachkommastelle haben, so sollte zum Beispiel auch bereits eine Genauigkeit von $\varepsilon = 10^{-3}$ genügen. Deshalb werden wir uns beim Vergleich der Ergebnisse mit Remke et al. [Rem+07] auf die Ergebnisse aus Tabelle 5.4 beziehen.
Um Iterationen und Laufzeiten vergleichen zu können, haben wir uns zwei Skripte in Python geschrieben, die die PRISM-Logs automatisch auswerten. Das Laufzeit-Skript sucht sich konkret die Stelle heraus, an der in den Logs die Zeit für das Model Checking steht, trägt die Zeiten in eine Tabelle ein und summiert sie zusätzlich separat auf. Analog sucht das Iterationsanzahl-Skript die Iterationszahlen heraus und summiert diese auf. Die Implementierungen der Skripte befinden sich in den Anhängen A.18 und A.19. Alle folgenden Werte wurden mit diesen Skripten ausgelesen und die Ergebnisdateien sind dem Anhang beigefügt.
Für $\varepsilon = 10^{-3}$ erhalten wir eine maximale Levelanzahl von $m = 19$ für $t = 0.5$, bei der die zugehörige Wahrscheinlichkeit das Konvergenzkriterium erfüllt. Wenn wir nun bedenken, dass wir Schrittgröße 1 gewählt haben und somit bei jedem Schritt eine Wahrscheinlichkeitsberechnung gemacht haben, ist der Aufwand sehr hoch. Konkret sind dies 3279 Iterationen. Um die Laufzeit hinzuzunehmen, so betrachten wir die Model Checking-Zeiten von PRISM. Für diese gibt PRISM häufig 0.0 Sekunden an. Das ist darauf zurückzuführen, dass PRISM nur mit einer geringen Genauigkeit die Model Checking-Zeiten berechnet und daher für sehr kleine Zeiten 0.0 Sekunden angibt. Für einige Iterationen jedoch variiert der Wert, sodass wir als Gesamtzeit $6.3 \cdot 10^{-2}$ Sekunden erhalten.

Beim Model Checking von Remke et al. [Rem+07] wiederum betrug die Rechenzeit gerade mal $1.26 \cdot 10^{-3}$ Sekunden und durch das dynamische Abbruchkriterium waren abhängig von der Wahrscheinlichkeitsschranke p maximal ungefähr 170 Iterationen nötig. Unsere Methode ist also im Vergleich deutlich schlechter, sowohl in Bezug auf die Iterationsanzahl als auch auf die Zeit.
Man muss jedoch nun auch Infragestellen, wie sinnvoll eine Schrittgröße von 1 und ein Startlevel von 1 sind. Wenn man beispielsweise sowohl das Startlevel wie auch die Schrittgröße auf 5 erhöhen würde, so hätte man nur noch ungefähr 880 Iterationen und eine Laufzeit von etwa $1.6 \cdot 10^{-2}$. Die Werte wären immer noch schlechter als die von Remke et al. [Rem+07], aber eine Verbesserung gegenüber unseren vorherigen Ergebnissen.
Betrachten wir den größten Zeitpunkt $t = 2$, so haben wir hier nur noch $m = 3$, das heißt wir mussten 1636 Iterationen durchlaufen. Bei Remke et al. [Rem+07] waren es lediglich maximal 550 Iterationen. Die Model Checking-Zeit von PRISM beträgt hier exakt 0.0 Sekunden. Dies ist wieder auf die geringe Laufzeitgenauigkeit von PRISM zurückzuführen. 0.0 Sekunden wäre sogar eine bessere Laufzeit als die von Remke et al. [Rem+07] von $4.92 \cdot 10^{-3}$. Man muss dabei aber bedenken, dass 0.0 Sekunden für eine Laufzeit nicht realistisch ist und deshalb vermutlich durch Rundungsfehler entstanden ist.
Es ist außerdem nicht bekannt, wie und womit die Laufzeiten von Remke et al. [Rem+07] gemessen wurden. Folglich ist sogar in Frage zu stellen, ob wir unsere Laufzeiten mit den Laufzeiten von Remke et al. [Rem+07] überhaupt vergleichen können. Aufgrund unserer deutlich höheren Iterationsanzahl lässt sich vermuten, dass trotz guter PRISM-Laufzeiten unser Verfahren schlechter als die Methodik von Remke et al. [Rem+07] ist.
Unterstützt wird dies dadurch, dass wir bereits aus unseren theoretischen Überlegungen in Abschnitt 5.2 wissen, dass die Methode der Uniformisierung mit Repräsentanten durch das dynamische Abbruchkriterium deutlich effizienter ist als unsere Approximation des Abschneidens. Es stellt sich also vielmehr die Frage, ob unsere Approximation trotzdem akzeptabel ist.

Unsere Iterationsgesamtzahl ist abhängig davon, wie viele maximale Level m geprüft werden. Für jedes $m \geq 2$ ist die Anzahl der Iterationen dieselbe, nur für $m = 1$ ist sie etwas kleiner. Je größer wir also die Schrittweite und unser Startlevel wählen, umso weniger Iterationen sind nötig, um eine aussagekräftige Wahrscheinlichkeit zu erhalten. Je höher m ist, umso länger dauert auch das Model Checking. Die Zeit, die wir durch weniger Iterationen einsparen, wird dementsprechend für das Starten bei zu großen m wieder ausgeglichen. Als Beispiel möchten wir unsere Berechnen für $t = 12$ und $m \in \{5, 10, 15, ..., 200\}$ für Formel $\Phi_{u3exakt}$ aufführen. Die exakten Wahrscheinlichkeiten befinden sich im Anhang A.13, die entsprechenden Laufzeiten in Anhang A.23.
Bis $m = 30$ haben wir Werte von bis zu $1.9 \cdot 10^{-2}$ Sekunden. Von $m = 35$ bis $m = 125$ variieren diese aber bereits zwischen $2.3 \cdot 10^{-2}$ und $8.1 \cdot 10^{-2}$. Ab $m = 130$ steigt die Laufzeit sogar ein einigen Fällen auf mehr als $1.4 \cdot 10^{-1}$ Sekunden an. Wenn also beispielsweise schon für $m = 100$ die Wahrscheinlichkeit

im Sinne von unserem Konvergenzkriterium aussagekräftig wäre, so wäre ein Starten mit $m = 150$ eher nachteilig statt förderlich. Als notwendige Konsequenz ist ein Trade-Off zwischen hohem Startlevel und niedriger Laufzeit nötig. Wir müssen dafür die passenden Parameter für das Pythonskript zum Konvergenzkriterium auswählen und speziell das Startlevel, die Schrittgröße und den absoluten Fehler betrachten. Der absolute Fehler kann abhängig von der Wahrscheinlichkeitsschranke p gewählt werden. Es empfiehlt sich mit einem eher niedrigen Startlevel zu beginnen, aber die Schrittgröße angemessen groß zu wählen. So wird vielen kleinen Iterationen entgegengewirkt, aber durch eine größere Schrittgröße sind auch nicht allzu viele zeitintensive Berechnungen für große m nötig.

An dieser Stelle möchten wir unsere Analyse beenden. Wir haben eine Problemstellung erfasst und als Lösungsstrategie ein Konvergenzkriterium entwickelt. Dieses wurde getestet und die Ergebnisse mit den Resultaten von Remke et al. [Rem+07] verglichen. Im Folgenden ziehen wir ein abschließendes Fazit, indem wir auch einige bisher noch nicht erwähnte Methoden und Model Checking-Algorithmen für QBDs kurz erläutern.

6. Fazit und Ausblick

Wir möchten noch einmal kurz herausstellen, welche neuen Erkenntnisse wir in dieser Arbeit gewonnen haben. Die Uniformisierung mit Repräsentanten nach Remke et al. [Rem+07] ermöglicht die effiziente Berechnung von transienten Zustandswahrscheinlichkeiten. Dadurch kann ein unendlicher QBD analysiert werden und anhand des dynamischen Abbruchkriteriums entschieden werden, ob eine Bounded-Until-Formel gültig ist.
Unsere Analyse des TCP sollte jedoch mit dem Model Checker PRISM durchgeführt werden und PRISM unterstützt aktuell keine unendlichen QBDs. Aufgrund des Umfangs dieser Arbeit war es auch nicht möglich PRISM um die Uniformisierung mit Repräsentanten zu erweitern. Deshalb haben wir den unendlichen QBD durch einen endlichen QBD mit einer maximalen Levelanzahl m approximiert.
Anhand von zwei Bounded-Until-Formeln haben wir unsere Approximation dann in Hinblick auf den Zusammenhang zwischen der maximalen Levelanzahl m und den Zeitpunkten t untersucht. Wir konnten dabei feststellen, dass sich für unterschiedliche maximale Levelanzahlen m auch gegenteilige Aussagen über die Gültigkeit einer CSL-Formel treffen lassen konnten. Dies haben wir formell als Problem festgehalten und als Lösungsansatz ein Konvergenzkriterium entwickelt. Dessen Implementierung erfolgte als Pythonskript und berechnet eine maximale Levelanzahl m, bei der wir eine Wahrscheinlichkeit als aussagekräftig für die Gültigkeit einer CSL-Formel ansehen.
Um abschließend einen Vergleich zwischen der Uniformisierung mit Repräsentanten und unserer Approximation des Abschneides zu ziehen, haben wir die Iterationsanzahlen und Laufzeiten des Model Checkings gemessen. Es ließ sich dabei feststellen, dass die Uniformisierung mit Repräsentanten deutlich effizienter ist, da dort das dynamische Abbruchkriterium verwendet werden kann. Außerdem ist die Uniformisierung mit Repräsentanten mehr für allgemeinere Anwendungen geeignet als unsere Approximation. Wenn die Performance des Model Checkings vernachlässigbar ist, kann möglicherweise auch unsere Approximation in Kombination mit dem auf dem Konvergenzkriterium basierenden Skript eingesetzt werden. Dies hätte den Vorteil, dass man ein viel genutztes und sehr gut gewartetes Tool wie PRISM verwenden könnte. Wir würden dennoch für die Anwendung in der Zukunft die Methodik von Remke et al. [Rem+07] empfehlen.
An dieser Stelle möchten wir noch kurz zusammenfassen, welche Methoden es zum Model Checking von unendlichen QBDs außer den bisher genannten gibt. Bereits 1984 haben Gross und Miller [Gro+84] eine randomisierte Methode

zur Berechnung von transienten Wahrscheinlichkeiten von endlichen CTMCs, insbesondere von endlichen QBDs, geschildert. Sie deuten dabei auch an, dass ihre Methode aufgrund der breiten Anwendbarkeit auch durchaus für unendliche Markov Prozesse verwendet werden könnte, indem man die unendlichen Markov Prozesse durch endliche Markov Prozesse approximieren würde, zeigen aber keine konkrete Ausarbeitung für unendliche QBDs.
Randomisierung wurde ebenfalls in einer älteren Arbeit von Grassmann [Gra77] behandelt, allerdings wurde dort kein konkreter Bezug zu unendlichen CTMCs hergestellt, sondern lediglich die Anwendbarkeit für endliche CTMCs mit sehr großen Zustandsräumen diskutiert.
Deutlich konkreter behandeln Zhang und Coyle [Zha+89] QBDs. Sie teilen unter anderem auch QBDs in Level ein und nehmen an, dass es *Complete Level Crossing Information*, eine Art levelunabhängige Eigenschaften, gibt. Spezieller angeschnitten wird das Time-Bounded-Model Checking von QBDs von Hahn et al. [Hah+09a] 2009. Dort wird eine Methode namens *truncation* (Abschneiden) geschildert, in der das unendliche Modell nur bis zu einer gewissen endlichen Tiefe betrachtet wird. Das ist annähernd vergleichbar zu unserer Approximation mit einem maximalen Level m. Hahn et al. stellen diese Methode jedoch für beliebig strukturierte CTMCs vor und beschränken sich nicht auf die Einteilung in Level, wie wir sie vorgenommen haben.
Ebenfalls in 2009 veröffentlichten Hahn et al. [Hah+09b] ein Paper über ihren Model Checker INFAMY, welcher auf PRISM basiert. INFAMY verwendet die von Hahn et al. schon zuvor vorgestellte Methode des Abschneidens und kann Time-Bounded-Model Checking für beliebig strukturierte CTMCs ausführen. Auch wenn mit INFAMY keine Berechnung von stationären Wahrscheinlichkeiten möglich ist, würden wir das Tool unserer eigenen Approximation vorziehen und halten die Entwicklung dieses Model Checkers für besonders interessant.
Eine weitere Form einer Approximation von unendlichen QBDs wurde 2016 von Delicaris [Del16] beschrieben. Dabei wird MDP-Abstraktion verwendet, um das ursprüngliche Verhalten des unendlichen QBD durch Aktionen in einer CTMDP zu simulieren. Durch Diskretisierung wird die CTMDP auf eine MDP reduziert und ist so ein von PRISM unterstütztes Modell. Allerdings treten auch bei dieser Approximation gewisse Wahrscheinlichkeitsungenauigkeiten durch die Abstraktion und Diskretisierung auf.
Insgesamt lässt sich sagen, dass es schon einige Approximationen von unendlichen QBDs gibt und auch einige Methoden, um transiente Wahrscheinlichkeiten zu berechnen. Das effiziente Model Checking von beliebigen CSL-Operatoren wird aber auch künftig ein Gebiet mit vielen Problemen, die neue Lösungsstrategien fordern, sein. Dies ist darauf zurück zu führen, dass viele der bisher gefundenen Lösungen entweder die Struktur des QBDs einschränken oder nur für einen Teil der CSL-Operatoren verwendet werden können. Es wird immer bessere Approximationen geben, aber zunächst weiterhin offen bleiben, ob eine Methode für beliebig strukturierte QBDs und für beliebige CSL-Operatoren überhaupt gefunden werden kann.

A. Anhang

```
1  // Minimized TCP - OCDR model by restricting the maximal levels of the QBD
2  // Based on Anne Remkes paper "CSL model checking algorithms for QBDs" (
       published in Theoretical Computer Science)
3  // By Stefanie Drerup
4
5  ctmc
6  //-------------------------------------------------------
7  // Packet Generator
8  // generates packets in burst and does nothing in off
9
10 //alpha: rate for burst to off, beta: rate for off to burst
11 const double alpha = 1;
12 const double beta = 0.04;
13
14 module packet_generator
15   // State k: 0-off (idle), 1-burst (active)
16   k: [0..1] init 1;
17
18   //from off to burst
19   [] k=0 -> beta: (k'=1);
20   //from burst to off
21   [] k=1 -> alpha: (k'=0);
22 endmodule
23
24 //-------------------------------------------------------
25 // Connection Management
26 // When it is active, there is a connection established and packets can be
       transmitted. Otherwise the connection is relased.
27
28 //r: rate for releasing the connection, c: rate for establishing the
       connection
29 const double r = 10;
30 const double c = 10;
31
32 module connection_management
33   // State j: 0-released, 1-active
34   j: [0..1] init 0;
35
36   //establish the connection if packets are avaiable
37   [] i>0 & j=0 -> c: (j'=1);
38   //release the connection if there are no packets any more
39   [] i=0 & j=1 -> r: (j'=0);
40 endmodule
41
42 //-------------------------------------------------------
43 // Transfer Queue
44 // Holds the packets waiting to be sent
45
46 // m = max_level: size of the queue (maximum level of the QBD), lambda: rate
       of generating a packet, mu: rate of transmitting a packet
47 const int m;
48 const double lambda = 100;
49 const double mu = 125;
50
51 module transfer_queue
```

```
52    // State i: number of waiting packets
53    i: [0..m] init 0;
54
55    //generate a new packet while the packet generator is active (burst). If
          possible, store it in the queue (otherwise the packet is lost)
56    [] i<m & k=1 -> lambda: (i'=i+1);
57
58    //send a packet while the connection management is active and the queue not
          empty
59    [] i>0 & j=1 -> mu: (i'=i-1);
60  endmodule
61
62  //-------------------------------------------------------
63
64  // Labels for defining properties of states
65  label "burst" = k=1;
66  label "off" = k=0;
67  label "active" = j=1;
68  label "released" = j=0;
```

Listing A.1: PRISM-TCP-Modell

```
1  //TCP-model: bounded-until-formula 1
2  const double t;
3
4  P=? [ F<=t ("released"&!"burst") ]
```

Listing A.2: Implementierung von $P_{=?}(tt\, \mathcal{U}^{[0,t]}$ released & no burst$)\,; t \in \mathbb{R}^+$

```
1  //TCP-model: bounded-until-formula 2
2  const double t;
3
4  P=? [ F<=t ("active"&!"burst") ]
```

Listing A.3: Implementierung von $P_{=?}(tt\, \mathcal{U}^{[0,t]}$ active & no burst$)\,; t \in \mathbb{R}^+$

```
1  m Result
2  5 0.3312087891244604
3  10  0.3175471539235105
4  15  0.30644238459841133
5  20  0.29976126218404126
6  25  0.2970720602565338
7  30  0.2964417047730217
8  35  0.29637281960686945
9  40  0.29636982765797765
10 45  0.29636977892206406
11 50  0.2963697786221678
12 55  0.2963697786214653
13 60  0.2963697786214647
14 65  0.2963697786214647
15 70  0.2963697786214647
16 75  0.2963697786214647
17 80  0.2963697786214647
18 85  0.2963697786214647
19 90  0.2963697786214647
20 95  0.2963697786214647
21 100 0.2963697786214647
22 105 0.2963697786214647
23 110 0.2963697786214647
24 115 0.2963697786214647
25 120 0.2963697786214647
26 125 0.2963697786214647
27 130 0.2963697786214647
28 135 0.2963697786214647
```

```
140 0.2963697786214647
145 0.2963697786214647
150 0.2963697786214647
155 0.2963697786214647
160 0.2963697786214647
165 0.2963697786214647
170 0.2963697786214647
175 0.2963697786214647
180 0.2963697786214647
185 0.2963697786214647
190 0.2963697786214647
195 0.2963697786214647
200 0.2963697786214647
```

Listing A.4: PRISM-Ergebnisse für $P_{=?}(tt\,\mathcal{U}^{[0,t]}$ released & no burst$)$; $t = 0.5$

```
m Result
5 0.5923736991043467
10  0.5846806657707828
15  0.5792044406041982
20  0.574540591056105
25  0.5706104722577302
30  0.5677195079347891
35  0.5658335829855862
40  0.5647103824010107
45  0.5641001552809561
50  0.5638110794910616
55  0.5637020589670527
60  0.5636728360999632
65  0.5636677956874206
70  0.5636672716755312
75  0.563667239898577
80  0.5636672387857902
85  0.5636672387632325
90  0.5636672387629679
95  0.5636672387629661
100 0.5636672387629661
105 0.5636672387629661
110 0.5636672387629661
115 0.5636672387629661
120 0.5636672387629661
125 0.5636672387629661
130 0.5636672387629661
135 0.5636672387629661
140 0.5636672387629661
145 0.5636672387629661
150 0.5636672387629661
155 0.5636672387629661
160 0.5636672387629661
165 0.5636672387629661
170 0.5636672387629661
175 0.5636672387629661
180 0.5636672387629661
185 0.5636672387629661
190 0.5636672387629661
195 0.5636672387629661
200 0.5636672387629661
```

Listing A.5: PRISM-Ergebnisse für $P_{=?}(tt\,\mathcal{U}^{[0,t]}$ released & no burst$)$; $t = 1$

```
m Result
5 0.8494107296941377
10  0.8464493916989387
15  0.844393735490442
20  0.8428593969970529
```

```
25  0.8417056273250721
30  0.8408297962484783
35  0.8401585953624395
40  0.8396516968039178
45  0.8392820441210458
50  0.8390233753815395
55  0.8388492226802916
60  0.8387356649462632
65  0.8386633866324387
70  0.8386181521968634
75  0.8385901675925304
80  0.8385730098099291
85  0.838562592199329
90  0.8385563588433669
95  0.8385527233569008
100 0.8385506980241204
105 0.8385496545299413
110 0.8385491784107251
115 0.8385489950471269
120 0.8385489380286434
125 0.838548924206653
130 0.8385489216579143
135 0.8385489213058376
140 0.8385489212697045
145 0.8385489212669583
150 0.8385489212668039
155 0.8385489212667981
160 0.8385489212667977
165 0.8385489212667977
170 0.8385489212667977
175 0.8385489212667977
180 0.8385489212667977
185 0.8385489212667977
190 0.8385489212667977
195 0.8385489212667977
200 0.8385489212667977
```

Listing A.6: PRISM-Ergebnisse für $P_{=?}(tt\ \mathcal{U}^{[0,t]}$ released & no burst$)\ ; t = 2$

```
m Result
5 0.944369574999947
10  0.9432300293444625
15  0.9424380442514964
20  0.9418457504505479
25  0.9413992274269951
30  0.9410656684569847
35  0.9408197066987889
40  0.9406400809610614
45  0.9405098430156618
50  0.9404162419606317
55  0.9403497867573153
60  0.9403033150143252
65  0.9402713663428698
70  0.9402497878579399
75  0.9402354635383152
80  0.940226104619834
85  0.94022007308966
90  0.9402162287989966
95  0.9402137992093933
100 0.9402122731735403
105 0.9402113189505642
110 0.9402107243056971
115 0.9402103547979177
120 0.9402101258156099
125 0.9402099843339384
130 0.9402098972246447
```

```
135 0.9402098438439606
140 0.9402098113602372
145 0.9402097918164039
150 0.940209780283245
155 0.940209773696496
160 0.9402097701274774
165 0.9402097683390317
170 0.9402097675334088
175 0.9402097672160166
180 0.9402097671092083
185 0.9402097670790731
190 0.9402097670720397
195 0.9402097670706946
200 0.9402097670704846
```

Listing A.7: PRISM-Ergebnisse für $P_{=?}(tt\, \mathcal{U}^{[0,t]}$ released & no burst$)$; $t = 3$

```
m Result
5 0.9794491057160662
10  0.9790112875369386
15  0.9787066455591834
20  0.9784786114996166
25  0.9783065650301288
30  0.9781778640168548
35  0.9780828657749707
40  0.9780137339417606
45  0.9779640678778395
50  0.9779287639807375
55  0.977903891247554
60  0.977886507597742
65  0.9778744524825579
70  0.977866158814358
75  0.9778605002759984
80  0.9778566734478439
85  0.9778541091867183
90  0.9778524071589123
95  0.9778512880326928
100 0.9778505587629037
105 0.9778500874182176
110 0.9778497849407604
115 0.9778495919779119
120 0.9778494694605039
125 0.9778493919568664
130 0.9778493430682088
135 0.9778493122993758
140 0.9778492929713307
145 0.9778492808509159
150 0.977849273263007
155 0.977849268520727
160 0.9778492655622137
165 0.9778492637200656
170 0.9778492625754345
175 0.9778492618658634
180 0.9778492614271641
185 0.977849261156804
190 0.9778492609908855
195 0.9778492608896675
200 0.9778492608284899
```

Listing A.8: PRISM-Ergebnisse für $P_{=?}(tt\, \mathcal{U}^{[0,t]}$ released & no burst$)$; $t = 4$

```
m Result
5 0.9924081245852643
10  0.992240157150512
15  0.9921231490969056
```

```
20  0.99203548857483
25  0.9919693029769281
30  0.9919197619527118
35  0.9918831699484418
40  0.9918565199434886
45  0.9918373684154035
50  0.9918237696935873
55  0.9918142133187442
60  0.9918075564554688
65  0.9918029540104811
70  0.9917997926572223
75  0.9917976338067972
80  0.991796167481822
85  0.9917951766647418
90  0.9917945105855634
95  0.9917940651587069
100 0.9917937689088404
105 0.9917935729872321
110 0.991793444164621
115 0.99179335994953
120 0.9917933052033926
125 0.991793269800586
130 0.9917932470148413
135 0.9917932324100776
140 0.9917932230815589
145 0.9917932171401682
150 0.9917932133648066
155 0.9917932109703023
160 0.991793209453948
165 0.991793208494974
170 0.9917932078892311
175 0.9917932075070529
180 0.9917932072662062
185 0.9917932071146058
190 0.9917932070192991
195 0.9917932069594613
200 0.9917932069219448
205 0.9917932068984582
210 0.9917932068837769
215 0.9917932068746167
220 0.9917932068689115
225 0.9917932068653652
230 0.9917932068631659
235 0.9917932068618054
240 0.9917932068609662
245 0.99179320686045
250 0.9917932068601352
255 0.9917932068599438
260 0.9917932068598285
265 0.9917932068597625
270 0.991793206859723
275 0.9917932068597011
280 0.9917932068596905
285 0.9917932068596846
290 0.9917932068596828
295 0.9917932068596816
300 0.9917932068596814
```

Listing A.9: PRISM-Ergebnisse für $P_{=?}(tt\ \mathcal{U}^{[0,t]}$ released & no burst$)$; $t = 5$

```
m Result
200 0.9999810041907226
205 0.9999814898298405
210 0.9999819630429582
215 0.9999824998360487
220 0.999983020618745
```

```
225 0.9999835258480377
230 0.9999840847629772
235 0.9999845580891188
240 0.9999850815596691
245 0.9999855868616861
250 0.9999861341216354
255 0.9999866593538608
260 0.9999871628198704
265 0.9999876445657467
270 0.9999881043958234
275 0.9999885890562834
280 0.9999890451144849
285 0.9999894300345595
290 0.9999898659665024
295 0.999990227650223
300 0.9999905548837184
305 0.9999908738172631
310 0.99999112601153
315 0.9999913623905218
320 0.9999915413791716
325 0.9999917014522981
330 0.9999918267539977
335 0.9999919219208684
340 0.9999919919521316
345 0.9999920418232715
350 0.9999920761560968
355 0.9999920989854372
360 0.9999921136369152
365 0.9999921227068971
370 0.9999921281199918
375 0.9999921312332455
380 0.9999921329581349
385 0.9999921338785036
390 0.9999921343513454
395 0.9999921345851984
400 0.9999921346965207
405 0.9999921347475221
410 0.999992134770008
415 0.9999921347795477
420 0.9999921347834424
425 0.9999921347849724
430 0.9999921347855508
435 0.9999921347857612
440 0.999992134785835
445 0.9999921347858599
450 0.9999921347858679
455 0.9999921347858706
460 0.9999921347858713
465 0.9999921347858715
470 0.9999921347858716
475 0.9999921347858716
480 0.9999921347858716
485 0.9999921347858716
490 0.9999921347858716
495 0.9999921347858715
500 0.9999921347858715
505 0.9999921347858713
510 0.9999921347858712
515 0.9999921347858713
520 0.9999921347858712
525 0.9999921347858711
530 0.9999921347858711
535 0.9999921347858711
540 0.9999921347858711
545 0.9999921347858711
550 0.9999921347858711
```

Listing A.10: PRISM-Ergebnisse für $P_{=?}(tt\ \mathcal{U}^{[0,t]}$ released & no burst$)$; $t = 12$

```
t Result
0.5 0.3312087891244604
1 0.5923736991043467
1.5 0.7522379410651561
2 0.8494107296941377
2.5 0.9084721619528048
3 0.944369574999947
3.5 0.9661879461856128
4 0.9794491057160662
4.5 0.9875092102305324
5 0.9924081245852643
5.5 0.9953856742946926
6 0.99719542266548
6.5 0.9982953838702266
7 0.9989639379473976
7.5 0.9993702836913888
8 0.9996172597690121
8.5 0.999767371302897
9 0.9998586087734344
9.5 0.9999140627135799
10  0.9999477622107945
10.5  0.9999669075072102
11  0.9999711690091789
11.5  0.9999712336243162
12  0.9999712336439988
```

Listing A.11: PRISM-Ergebnisse für $P_{=?}(tt\ \mathcal{U}^{[0,t]}$ released & no burst$)$; $m = 5$

```
t Result
0.5 0.2963697786214647
1 0.5636672387629661
1.5 0.7345520109037675
2 0.8385489212667977
2.5 0.9017590929571705
3 0.9402097670704554
3.5 0.963608296176628
4 0.9778492607474499
4.5 0.9865172129084377
5 0.9917932068596812
5.5 0.9950046225835499
6 0.9969593700865098
6.5 0.9981492020318096
7 0.9988734394216385
7.5 0.9993142747774166
8 0.999582606467301
8.5 0.999745937062618
9 0.9998453546314781
9.5 0.9999058690325239
10  0.9999427034955011
10.5  0.9999651242357498
11  0.9999787714984855
11.5  0.9999870784401073
12  0.9999921347858715
```

Listing A.12: PRISM-Ergebnisse für $P_{=?}(tt\ \mathcal{U}^{[0,t]}$ released & no burst$)$; $m = 500$

```
m Result
5 0.9999712336439988
10  0.9999712208748327
15  0.9999711974869181
```

```
20  0.9999713086384339
25  0.9999712679862063
30  0.9999713010550841
35  0.9999713906498231
40  0.9999713996208105
45  0.9999715639074057
50  0.9999716285818718
55  0.9999717102697084
60  0.9999718039894014
65  0.9999720270734983
70  0.9999721343636069
75  0.9999722454168987
80  0.9999724779536802
85  0.9999727104821289
90  0.9999729423569377
95  0.9999731731429807
100 0.9999734025505984
105 0.9999736303899155
110 0.9999739691441643
115 0.9999741925604867
120 0.9999745243765261
125 0.9999748520007361
130 0.9999751754603682
135 0.9999754947914639
140 0.9999758100356368
145 0.9999762240884439
150 0.9999766310627317
155 0.9999770310764106
160 0.9999774242465069
165 0.9999778106887727
170 0.9999782844550212
175 0.9999786561746218
180 0.9999791118895769
185 0.9999795578740828
190 0.9999800805027957
195 0.9999805058032374
200 0.9999810041907226
```

Listing A.13: PRISM-Ergebnisse für $P_{=?}(tt\,\mathcal{U}^{[0,t]}$ released & no burst$)\,; t = 12$

```
m Result
5 0.3653368455462555
10  0.37033211053616416
15  0.37203931965101816
20  0.3724050755200043
25  0.37244479746271664
30  0.37244682291298026
35  0.37244686876120936
40  0.3724468692094114
45  0.3724468692113028
50  0.3724468692113064
55  0.3724468692113064
60  0.3724468692113064
65  0.3724468692113064
70  0.3724468692113064
75  0.3724468692113064
80  0.3724468692113064
85  0.3724468692113064
90  0.3724468692113064
95  0.3724468692113064
100 0.3724468692113064
105 0.3724468692113064
110 0.3724468692113064
115 0.3724468692113064
120 0.3724468692113064
125 0.3724468692113064
```

```
130 0.3724468692113064
135 0.3724468692113064
140 0.3724468692113064
145 0.3724468692113064
150 0.3724468692113064
155 0.3724468692113064
160 0.3724468692113064
165 0.3724468692113064
170 0.3724468692113064
175 0.3724468692113064
180 0.3724468692113064
185 0.3724468692113064
190 0.3724468692113064
195 0.3724468692113064
200 0.3724468692113064
```

Listing A.14: PRISM-Ergebnisse für $P_{=?}(tt\,\mathcal{U}^{[0,t]}$ active & no burst$)$; $t = 0.5$

```
m Result
5 0.8391205814220797
10  0.8435857016147239
15  0.8451947608704004
20  0.8459020146014724
25  0.8462370796438259
30  0.8464029075291285
35  0.8464872205962822
40  0.8465300742670555
45  0.846551198556049
50  0.8465610084750308
55  0.8465651878876441
60  0.8465667848816233
65  0.8465673221443745
70  0.8465674788845025
75  0.8465675180358032
80  0.8465675263165787
85  0.8465675277850222
90  0.8465675280013988
95  0.846567528027677
100 0.8465675280302881
105 0.8465675280304988
110 0.8465675280305123
115 0.8465675280305133
120 0.8465675280305133
125 0.8465675280305133
130 0.8465675280305133
135 0.8465675280305133
140 0.8465675280305133
145 0.8465675280305133
150 0.8465675280305133
155 0.8465675280305133
160 0.8465675280305133
165 0.8465675280305133
170 0.8465675280305133
175 0.8465675280305133
180 0.8465675280305133
185 0.8465675280305133
190 0.8465675280305133
195 0.8465675280305133
200 0.8465675280305133
```

Listing A.15: PRISM-Ergebnisse für $P_{=?}(tt\,\mathcal{U}^{[0,t]}$ active & no burst$)$; $t = 2$

```
m Result
5 0.9705710855856027
10  0.9745454010458428
```

```
15  0.9759763791945789
20  0.9766052939472571
25  0.9769028977294923
30  0.9770483066126353
35  0.9771204227528026
40  0.9771564486473041
45  0.9771745098079556
50  0.9771835807692071
55  0.9771881407996696
60  0.9771904344000738
65  0.9771915884559765
70  0.9771921692960222
75  0.9771924616894084
80  0.9771926088792721
85  0.9771926829432455
90  0.9771927201656898
95  0.9771927388242094
100 0.9771927481336388
105 0.9771927527436848
110 0.9771927550013788
115 0.977192756090261
120 0.9771927566050883
125 0.9771927568425767
130 0.977192756948961
135 0.9771927569950289
140 0.9771927570142331
145 0.97719275702191
150 0.9771927570248423
155 0.9771927570259088
160 0.9771927570262776
165 0.9771927570263979
170 0.9771927570264354
175 0.9771927570264459
180 0.977192757026449
185 0.9771927570264494
190 0.9771927570264497
195 0.9771927570264497
200 0.9771927570264497
```

Listing A.16: PRISM-Ergebnisse für $P_{=?}(tt\, \mathcal{U}^{[0,t]}$ active & no burst$)$; $t = 5$

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#With this script a probability dependant of the maximum level m can be computed.
#If the probability difference of two steps is smaller than a a-priori error the iterations will stop.
#It is possible to change the paths of the model, the properties and the result.txt
#Also the beginning maximum level, the step size, the a-priori error and the time step (settings) are changeable
#try python mprism.py -h for more usage information
import subprocess, getopt, sys

#example: python mprism2.py -f ./models/tcp.prism -p ./properties/tcp.exactuntil.csl -r result.txt -m 1 -s 1 -e 1e-4 -t 0.5

# ------------------------------------------------------------------------
# parsing commands of command line call and start computation
# ------------------------------------------------------------------------

def main(argv):
   model = ''
   properties = ''
   result = 'result.txt'
```

```
21      m = 1
22      steps = 1
23      eps = 1e-4
24      t = 0.5
25      ready = False
26      try:
27          opts, args = getopt.getopt(argv,"hf:p:r:m:s:e:t:",["model=","property=",
                "result=","startlevel=","stepsize=","eps=","timestep="])
28      except getopt.GetoptError:
29          print("""\
30  try
31       python mprism.py -h
32  for more usage informations
33  """)
34          sys.exit(2)
35      for opt, arg in opts:
36          if opt == '-h':
37              print("""\
38  usage: python mprism.py OPTIONS
39       -f,--model=       path of modelfile
40       -p,--property=      path of propertyfile (only 1 property)
41       -r,--result=      path of resultfile [default: result.txt]
42       -m,--startlevel=       startlevel for computation [int, >0, default: 1]
43       -s,--stepsize=      stepsize of the iteration [int, >0, default: 1]
44       -e,--eps=         max error for convergence [default: 1e-4]
45       -t,--timestep=      timestep [float, >0, default: 0.5]
46
47  for example: python mprism.py -f ./model.prism -p ./property.csl -r result.txt
         -m 1 -s 1 -e 1e-4 -t 0.5
48  """)
49              sys.exit()
50          elif opt in ("-f", "--model"):
51              model= arg
52          elif opt in ("-p", "--property"):
53              properties = arg
54          elif opt in ("-r", "--result"):
55              result = arg
56          elif opt in ("-m", "--startlevel") and int(arg)>0:
57              m = int(arg)
58          elif opt in ("-s", "--stepsize")and int(arg)>0:
59              steps = int(arg)
60          elif opt in ("-e", "--eps") and float(arg)>0 and 1>float(arg):
61         eps = float(arg)
62          elif opt in ("-t", "--timestep") and float(arg)>0:
63              t = float(arg)
64          ready = True
65      if ready:
66       computeProb(model,properties,result,m,steps,eps,t)
67      else:
68       print("""\
69  try
70       python mprism.py -h
71  for more usage informations
72  """)
73
74  # ------------------------------------------------------------------------
75  # manual definition of paths of prism model, properties and result file
76  # ------------------------------------------------------------------------
77  #model = "tcp.prism"
78  #properties = "tcp.exactuntil.csl"
79  #result = "result.txt"
80
81  # ----------------------------------------------------
82  # manual definition of model checking settings
83  # ----------------------------------------------------
84  #m = 1        #beginning maximum level
85  #steps = 1      #number of steps between m and a further m
```

```
#eps = 10**(-4)   #maximum error of probability difference for m and n
#t = 0.5        #time step


def computeProb(model,properties,result,m,steps,eps,t):
  #-------------------------------------------------------------------------
  # building and executing of prism command - beginning step
  #-------------------------------------------------------------------------
  n = m+steps     #next iteration of m
  probm = 0     #default probability for level m
  probn = 1     #default probability for level n
  #create empty resultfile
  resultfile = open(result,"w")
  resultfile.close()

  #open file for computation
  resultfile = open(result,"r+")
  #prism computation
  prismcommand = "prism " + model + " " + properties + " -const m=" + str(m) +
       ":" + str(steps) +":" + str(n) + ",t=" + str(t) + " -exportresults " +
      result
  prismprocess = subprocess.check_call(prismcommand, stdin=None, stdout=None,
      stderr=None, shell=True)

  #reading of resultfile
  content = resultfile.read()
  probm = float(content.split( )[3])
  probn = float(content.split( )[5])
  probdiff = abs(probm - probn)
  resultfile.close()

  #-------------------------------------------------------------------------
  # iterating until the probability difference is smaller than a-priori eps
  #-------------------------------------------------------------------------
  while probdiff >= eps:
    #iteration of m and n
    m = n;
    n += steps;

    #open file for computation
    resultfile = open(result,"r+")
    prismcommand = "prism " + model + " " + properties + " -const m=" + str(n)
         + ",t=" + str(t) + " -exportresults " + result
    prismprocess = subprocess.check_call(prismcommand, stdin=None, stdout=None
        , stderr=None, shell=True)

    #reading of resultfile
    content = resultfile.read()
    print(content)
    probm = probn
    probn = float(content.split( )[1])
    probdiff = abs(probm - probn)
    resultfile.close()

  #print final m and final probability difference
  print("convergence for m = " + str(m) + " with a probability difference of "
       + str(probdiff))

#call main-method with sys.argv except of python-filename
if __name__ == "__main__":
   main(sys.argv[1:])
```

Listing A.17: Python-Skript zur Implementierung des Konvergenzkriteriums

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```
3
4  #this script reads a log file, filters all model checking times and sums them
       up
5  #then the scripts writes the output in a resultfile
6  #paths of logfile and resultfile plus the parameter can be changed
7  #try python sumtime.py -h for more usage information
8  import getopt, sys, re
9
10 #example: python sumtime.py -f ./log.txt -r ./result.txt -p m
11
12 # ------------------------------------------------------------------------
13 # parsing commands of command line call and start computation
14 # ------------------------------------------------------------------------
15
16 def main(argv):
17    log = ''
18    result = './result.txt'
19    param = 'm'
20    ready = False
21    try:
22       opts, args = getopt.getopt(argv,"hf:r:p:",["logfile=","resultfile=","
              parameter="])
23    except getopt.GetoptError:
24       print("""\
25 try
26      python sumtime.py -h
27 for more usage informations
28 """)
29       sys.exit(2)
30    for opt, arg in opts:
31       if opt == '-h':
32          print("""\
33 usage: python sumtime.py OPTIONS
34      -f,--logfile=       path of logfile
35      -r,--resultfile=      path of resultfile [default: ./result.txt]
36      -p,--parameter      'm' for maximum level, 't' for time step
37                  [default: 'm']
38
39 for example: python sumtime.py -f ./log.txt -r ./result.txt -p m
40 """)
41          sys.exit()
42       elif opt in ("-f", "--logfile"):
43          log = arg
44       elif opt in ("-r", "--resultfile"):
45          result = arg
46       elif opt in ("-p", "--parameter") and (opt == 'm' or opt == 't'):
47          param = arg
48       ready = True
49    if ready:
50     computeTime(log, result, param)
51    else:
52     print("""\
53 try
54      python sumtime.py -h
55 for more usage informations
56 """)
57
58
59 def computeTime(log, result, param):
60   summedtime = 0
61
62   #create empty resultfile
63   resultfile = open(result,"w")
64   resultfile.close()
65
66   #open files for computation
67   resultfile = open(result,"r+")
```

```
  logfile = open(log,"r")

  #reading line for line and looking for model checking time and parameter
  #if found, it writes them in the output file
  #finally the total time is also computed
  resultfile.write("parameter" + "\t" + "time for model checking" + "\n")
  paramvalue = ''
  timevalue = ''
  for line in logfile:
    if param == 'm':
      #necessary cause there is two times the phrase "model constant" for one
          model check
      if re.search('Model checking: ', line):
        paramvalue = logfile.next()[19:-1]
    elif param == 't':
      if re.search('Property constants: t=', line):
        paramvalue = line[22:-1]
    if re.search('Time for model checking:', line):
      summedtime += float(line[25:-9])
      timevalue = line[25:-9]
      resultfile.write(paramvalue + "\t" + timevalue + "\n")

  resultfile.write("\n")
  resultfile.write("total time: " + str(summedtime))

  #close files
  resultfile.close()
  logfile.close()

#call main-method with sys.argv except of python-filename
if __name__ == "__main__":
   main(sys.argv[1:])
```

Listing A.18: Python-Skript zur Berechnung der Laufzeit

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#this script reads a log file, filters all iterations and sums them up
#then the scripts writes the output in a resultfile
#paths of logfile and resultfile plus the parameter can be changed
#try python sumtime.py -h for more usage information
import getopt, sys, re

#example: python sumtime.py -f ./log.txt -r ./result.txt -p m

# ------------------------------------------------------------------------
# parsing commands of command line call and start computation
# ------------------------------------------------------------------------

def main(argv):
   log = ''
   result = './result.txt'
   param = 'm'
   ready = False
   try:
      opts, args = getopt.getopt(argv,"hf:r:p:",["logfile=","resultfile=","
          parameter="])
   except getopt.GetoptError:
      print("""\
try
     python sumtime.py -h
for more usage informations
""")
      sys.exit(2)
   for opt, arg in opts:
```

```
      if opt == '-h':
         print("""\
usage: python sumtime.py OPTIONS
     -f,--logfile=      path of logfile
     -r,--resultfile=     path of resultfile [default: ./result.txt]
     -p,--parameter     'm' for maximum level, 't' for time step
                [default: 'm']

for example: python sumtime.py -f ./log.txt -r ./result.txt -p m
""")
         sys.exit()
      elif opt in ("-f", "--logfile"):
         log = arg
      elif opt in ("-r", "--resultfile"):
         result = arg
      elif opt in ("-p", "--parameter") and (opt == 'm' or opt == 't'):
         param = arg
      ready = True
   if ready:
    computeIterations(log, result, param)
   else:
    print("""\
try
     python sumtime.py -h
for more usage informations
""")


def computeIterations(log, result, param):
  summediter = 0

  #create empty resultfile
  resultfile = open(result,"w")
  resultfile.close()

  #open files for computation
  resultfile = open(result,"r+")
  logfile = open(log,"r")

  #reading line for line and looking for model checking iterations and
      parameter
  #if found, it writes them in the output file
  #finally the total iterations are also computed
  resultfile.write("parameter" + "\t" + "iterations" + "\n")
  paramvalue = ''
  itervalue = ''
  for line in logfile:
    if param == 'm':
      #necessary cause there is two times the phrase "model constant" for one
          model check
      if re.search('Model checking: ', line):
        paramvalue = logfile.next()[19:-1]
    elif param == 't':
      if re.search('Property constants: t=', line):
        paramvalue = line[22:-1]
    if re.search('Iterative method:', line):
      summediter += int(line[18:-58])
      itervalue = line[18:-58]
      resultfile.write(paramvalue + "\t" + itervalue + "\n")

  resultfile.write("\n")
  resultfile.write("total iterations: " + str(summediter))

  #close files
  resultfile.close()
  logfile.close()

```

```
#call main-method with sys.argv except of python-filename
if __name__ == "__main__":
    main(sys.argv[1:])
```

Listing A.19: Python-Skript zur Berechnung der Iterationsanzahl

```
parameter time for model checking
1 0.0
2 0.016
3 0.0
4 0.0
5 0.0
6 0.0
7 0.0
8 0.0
9 0.0
10  0.0
11  0.0
12  0.0
13  0.0
14  0.016
15  0.0
16  0.015
17  0.0
18  0.0
19  0.016

total time: 0.063
```

Listing A.20: Laufzeiten für $\Phi_{u3exakt}$, $t = 0.5,\ m \in \{1, 2, ..., 19\}$

```
parameter time for model checking
1 0.016
2 0.0
3 0.0
4 0.0
5 0.0
6 0.0
7 0.0
8 0.0
9 0.0
10  0.0
11  0.016
12  0.0
13  0.0
14  0.0
15  0.0

total time: 0.032
```

Listing A.21: Laufzeiten für $\Phi_{u3exakt}$, $t = 1,\ m \in \{1, 2, ..., 15\}$

```
parameter time for model checking
1 0.0
2 0.0
3 0.0

total time: 0.0
```

Listing A.22: Laufzeiten für $\Phi_{u3exakt}$, $t = 2,\ m \in \{1, 2, 3\}$

```
parameter time for model checking
```

```
5 0.011
10  0.008
15  0.01
20  0.013
25  0.016
30  0.019
35  0.023
40  0.026
45  0.061
50  0.032
55  0.035
60  0.038
65  0.042
70  0.045
75  0.049
80  0.052
85  0.055
90  0.082
95  0.06
100 0.064
105 0.068
110 0.07
115 0.075
120 0.074
125 0.081
130 0.101
135 0.1
140 0.103
145 0.094
150 0.102
155 0.111
160 0.117
165 0.111
170 0.116
175 0.142
180 0.155
185 0.14
190 0.14
195 0.146
200 0.148

total time: 2.935
```

Listing A.23: Laufzeiten für $\Phi_{u3exakt}$, $t = 12,\ m \in \{5, 10, 15, ..., 200\}$

```
parameter iterations
1 111
2 176
3 176
4 176
5 176
6 176
7 176
8 176
9 176
10  176
11  176
12  176
13  176
14  176
15  176
16  176
17  176
18  176
19  176

```

```
total iterations: 3279
```

Listing A.24: Iterationen für $\Phi_{u3exakt}$, $t = 0.5,\ m \in \{1, 2, ..., 19\}$

```
parameter iterations
1 193
2 315
3 315
4 315
5 315
6 315
7 315
8 315
9 315
10  315
11  315
12  315
13  315
14  315
15  315

total iterations: 4603
```

Listing A.25: Iterationen für $\Phi_{u3exakt}$, $t = 1,\ m \in \{1, 2, ..., 15\}$

```
parameter iterations
1 346
2 645
3 645

total iterations: 1636
```

Listing A.26: Iterationen für $\Phi_{u3exakt}$, $t = 2,\ m \in \{1, 2, 3\}$

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

[Bai+03] Christel Baier u. a. „Model-checking algorithms for continuous-time Markov chains“. In: *Software Engineering, IEEE Transactions on* 29.6 (2003), S. 524–541.

[Del16] Joanna Delicaris. „Erreichbarkeit in QBDs mithilfe von MDP-Abstraktion“. Bachelorthesis. Westfälische Wilhelms-Universität Münster, 2016.

[Gra77] Winfried K Grassmann. „Transient solutions in Markovian queueing systems“. In: *Computers & Operations Research* 4.1 (1977), S. 47–53.

[Gro+84] Donald Gross und Douglas R Miller. „The randomization technique as a modeling tool and solution procedure for transient Markov processes“. In: *Operations Research* 32.2 (1984), S. 343–361.

[Hah+09a] E Moritz Hahn u. a. „Time-bounded model checking of infinite-state continuous-time Markov chains“. In: *Fundamenta Informaticae* 95.1 (2009), S. 129–155.

[Hah+09b] Ernst Moritz Hahn u. a. „INFAMY: An infinite-state Markov model checker“. In: *Computer Aided Verification*. Springer. 2009, S. 641–647.

[Hei+96] Geert Heijenk und Boudewijn R Haverkort. „Design and evaluation of a connection management mechanism for an ATM-based connectionless service“. In: *Distributed Systems Engineering* 3.1 (1996), S. 53.

[Kwi+11] Marta Kwiatkowska, Gethin Norman und David Parker. „PRISM 4.0: Verification of Probabilistic Real-time Systems“. In: *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*. Hrsg. von G. Gopalakrishnan und S. Qadeer. Bd. 6806. LNCS. Springer, 2011, S. 585–591.

[Kwi+07] Marta Kwiatkowska, Gethin Norman und David Parker. „Stochastic model checking“. In: *Formal methods for performance evaluation*. Springer, 2007, S. 220–270.

[Rem+07] Anne Remke, Boudewijn R. Haverkort und Lucia Cloth. „CSL model checking algorithms for QBDs“. In: *Theoretical Computer Science* 382.1 (2007), S. 24–41.

[Zha+89] Ji Zhang und Edward J Coyle. „Transient analysis of quasi-birth-death processes“. In: *Stochastic Models* 5.3 (1989), S. 459–496.

Eidesstattliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit über

CSL Model Checking von
QBDs in PRISM

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.

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.

Stefanie Eva Drerup, Münster, 10. Mai 2016