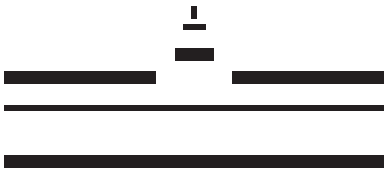


Westfälische
Wilhelms-Universität
Münster

Erreichbarkeit in QBDs mithilfe von MDP-Abstraktion

Bachelorarbeit
zur Erlangung des akademischen Grades
Bachelor of Science

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

Betreuung/Erstkorrektur:
Prof. Dr. Anne Remke

Zweitkorrektur:
Prof. Dr. Markus Müller-Olm

Eingereicht von:
Joanna Georgia Delicaris

Münster, 12. Mai 2016

Zusammenfassung

Um komplexe Systeme wie beispielsweise Kommunikationsprotokolle bereits vor ihrer physischen Realisierung auf Verlässlichkeit, Abhängigkeiten oder Performanz zu testen wird *Model Checking* verwendet. Hierfür wird das betrachtete System als stochastisches Modell mit Zuständen und probabilistischen Zustandsübergängen modelliert. Formuliert man nun mithilfe einer Logik wie *Probabilistic Computational Tree Logic* Anforderungen an das Modell, so kann in einem mathematischen Verfahren die Korrektheit oder auch Wahrscheinlichkeit einer solchen Formel berechnet werden. Eine häufig zu prüfende Anforderung ist die Erreichbarkeit einer Zustandsmenge innerhalb einer bestimmten Zeit: *time-bounded Reachability*. Betrachtet man komplexe, zeitgemäße Strukturen wie Kommunikationprotokolle, so stellt man fest, dass endliche Modelle lange nicht ausreichen, um die Anwendungen unseres Lebens zu modellieren. Model Checking auf unendlichen Modellen jedoch ist nicht vollständig erschlossen. Wir werden deshalb *MDP-Abstraktion* vorstellen, eine Technik, mit der für unendliche Warteschlangenmodelle, sogenannte *Quasi-Death-Birth-Processes*, eine endliche Repräsentation entwickelt werden kann. In einem weiteren Verfahren kann das erhaltene Modell weiter vereinfacht und diskretisiert werden. Durch diese Maßnahme können mithilfe eines Tools, dem *PRISM Model Checker*, Eigenschaften validiert werden.

Inhaltsverzeichnis

1. Einleitung

Kommunikationsprotokolle sind komplexe Systeme, über die Aussagen bezüglich Performanz und Abhängigkeiten getroffen werden können. Diese Systeme können als stochastische Modelle dargestellt werden, indem die Struktur des Systems als Zustände modelliert wird und das Eintreten möglicher Ereignisse mit probabilistischer Verteilung in Form von Übergängen zwischen den Zuständen veranschaulicht wird. Um Eigenschaften und Spezifikationen, die man an das Modell stellen möchte, zu verifizieren und validieren wird das sogenannte *Model Checking* verwendet. Durch diesen Prozess können Systeme schon vor ihrer physischen Umsetzung überprüft werden, was bei immer komplexeren und sicherheitskritischen Systemen an Relevanz gewinnt.
Beim Model Checking wird zwischen *symbolischem* und *explizitem* Model Checking unterschieden: symbolisches Model Checking verwendet für die Repräsentation der Zustände symbolische Datenstrukturen, sogenannte *binary-decision diagrams* (BDDs), während das explizite Model Checking, wie der Name schon sagt, die verwendeten Daten explizit speichert. [1]
Ein Beispiel für solche stochastischen Modelle sind *Continuos-time Markov Chains* (CTMCs). Mit CTMCs können viele verschiedenen, komplexen Strukturen in einem Modell festgehalten werden. Während Model Checking bisher überwiegend für endliche Modelle angewandt wurde, sind unendliche Modelle für eine Vielzahl an Systemen der passendere Ansatz. Kommunikationssysteme, wie beispielsweise *Transmission Control Protocol* (TCP), verwenden sehr große oder unendliche Warteschlangen. Für solche Systeme sind verschiedene Aspekte von Interesse, unter anderem die Serverauslastung, Warteschlangenlänge oder die Wartezeit eines Nutzers auf ein Ereignis.
Eigenschaften dieser Art lassen sich mit *Probabilistic Computational Temporal Logic* (PCTL) formalisieren. Eine wichtige Eigenschaft der Performanzevaluation ist *Time-bounded Reachability*. Diese Eigenschaft beschreibt, ob eine Zustandsmenge innerhalb des Modells innerhalb einer bestimmten Zeit erreichbar ist. Hiermit kann man im Bereich von Kommunikationsprotokollen überprüfen, wie hoch die Wahrscheinlichkeit für das Erreichen eines bestimmten Levels ist oder wie wahrscheinlich es ist, einen Zustand des Systems zu erlangen, in dem Pakete über das Protokoll verschickt werden.
Ein typisches Modell eines Warteschlangensystems ist ein sogenannter *Quasi-Birth-Death Process* (QBD). Durch ihre starke, level-basierte Struktur ist Model Checking auf QBDs einfacher, als auf großen unstrukturierten Modellen. [10]
Die starke Struktur der QBDs wollen wir nutzen, um sie in endliche Modelle zu überführen. Dafür verwenden wir die in [7] eingeführte *MDP-Abstraktion*. Durch Zusammenfassen aller korrespondierenden Zustände ab einem bestimmten Level eines QBD in einer abstrakten Ebene entsteht ein endliches Modell mit Nichtdeterminismus. Dieses nennt man

Continuos-time Markov Decision Process (CTMDP). In jedem Zustand des abstrakten Levels kann man zwischen verschiedenen Aktionen wählen, die ursprüngliche Zustände des QBD modellieren. Dadurch können beim Model Checking keine eindeutigen Wahrscheinlichkeiten berechnet werden. Wir können jedoch die maximale und minimale Wahrscheinlichkeit für das Erfüllen einer Eigenschaft auf dem CTMDP berechnen.
Tools wie der *PRISM Model Checker* sind entwickelt worden, um automatisiertes Model Checking durchzuführen. Da PRISM nicht in der Lage ist, Kontinuität in Kombination mit Nichtdeterminismus zu verarbeiten, werden wir ein weiteres Verfahren einführen, um die erhaltenen Modelle zu vereinfachen und die in [8] beschriebene Diskretisierung auf CTMDPs erläutern. Die daraus entstehende Modellstruktur ist ein *Markov Decision Process* (MDP). Dieser verhält sich diskret zu einer bei der Diskretisierung gewählten Zeitschrittdauer, weshalb immer erst nach Ablauf dieser Zeit genau ein Zustandsübergang stattfinden kann.
Auf einem MDP können anschließend mithilfe von Model Checking Verfahren wie *Value Iteration* oder *Policy Iteration* Wahrscheinlichkeiten für die Erreichbarkeit einer Zustandsmenge berechnet werden.

Zielsetzung Im Verlauf dieser Arbeit wollen wir eine Möglichkeit erläutern, Model Checking für zeit-gebundene Erreichbarkeit auf unendlichen QBDs durchführen zu können.

Übersicht Zunächst werden wir in **Kapitel 2** die Grundlagen behandeln. Dafür führen wir in **2.1** *Discrete-time Markov Chains* (DTMCs), CTMCs und QBDs ein. In **2.2** erläutern wir die in [7] erläuterte MDP-Abstraktion und definieren, wie sich diese Abstraktionsmethode konkret auf QBDs auswirkt. In **2.3** gehen wir auf die Diskretisierung von CTMDPs ein, wie sie in [8] vorgestellt wurde.
In **Kapitel 3** beschreiben wir, wie man Erreichbarkeit formalisiert und entsprechende Wahrscheinlichkeiten auf MDPs berechnet. Zunächst gehen wir in **3.1** auf Pfade, Scheduler und induzierte DTMCs ein, da diese die Grundlage für die folgenden Berechnungen schaffen. Daraufhin wird in **3.2** die Logik PCTL eingeführt, mit der wir Eigenschaften formalisieren werden. Das Berechnen von Wahrscheinlichkeiten mit Verfahren wie Value Iteration wird dann in **3.3** dargelegt.
Nachdem wir alle nötigen Werkzeuge für eine Analyse beschrieben haben, wollen wir diese Verfahren in einer Fallstudie anwenden. Dafür betrachten wir in **Kapitel 4** das *Transmission Control Protocol* (TCP). In **4.1** erläutern wir die Modellierung des Protokolls als CTMC. Anschließend wenden wir in **4.2** die Abstraktion an und in **4.3** diskretisieren wir den erhaltenen CTMDP.
In **Kapitel 5** führen wir eine Analyse des Modells durch. Da wir für die Berechnungen PRISM verwenden, führen wir in **5.1** die Grundlagen der *PRISM Language* ein. In **5.2** setzen wir das Modell des Protokolls um, beschreiben die konkrete Eigenschaft, die wir überprüfen wollen und gehen kurz auf die Parameter, die für die Berechnungen zu betrachten sind, ein. Im nächsten Unterkapitel **5.3** wollen wir dann unsere Resultate vorstellen und analysieren. Anschließend werden wir in **5.4** auf die Laufzeiten verschiedener Lösungsverfahren von

PRISM für unser Problem eingehen und in **5.5** auf einen anderen Lösungsansatz eingehen und Unterschiede zu unserer Idee aufzeigen.
In **Kapitel** 6 fassen wir abschließend den behandelten Inhalt zusammen und geben einen Ausblick auf weitere Betrachtungsansätze.

Verwandte Arbeit Wir wollen in diesem Abschnitt auf andere Arbeiten, die sich mit dem Model Checking unendlicher Systeme beschäftigen, eingehen. Konventionelle Model Checking Verfahren können nicht auf unendliche Modelle angewandt werden. Es gibt jedoch bereits einige Ansätze, gut strukturierte unendliche CTMCs zu prüfen. In [11] werden neue Algorithmen für das Model Checking von CSL-Formeln eingeführt. *Continuos Stochastic Logic* (CSL) ist eine Erweiterung der Logik PCTL. Für den zeitgebundenen Until-Operator, insbesondere also für zeitgebundene Erreichbarkeit, wird hier eine neue Methode namens *Uniformisierung mit Repräsentanten* eingeführt. Diese arbeitet mit einem dynamischen Abbruchkriterium, wobei das Model Checking einer Formel nur so lange durchgeführt werden muss, bis man mit Sicherheit sagen kann, dass sie bereits erfüllt bzw. nicht erfüllt ist. Dadurch kann auch auf unendlichen Systemen Model Checking durchgeführt werden, wenn sie eine starke Struktur haben.
Wie auch in [11] werden in [4] unendliche QBDs betrachtet. Darin werden die QBDs ab einer bestimmten Ebene abgeschnitten. Mithilfe eines a priori gewählten Fehlers wird untersucht, wieviele Level für eine aussagekräftige Wahrscheinlichkeitsberechnung von Nöten sind. In [6] wird ebenfalls ein Verfahren vorgestellt, das unendliche CTMCs abschneidet. Dabei wird in einem dynamischen Ansatz der optimale Punkt zum Abschneiden der CTMC gefunden.
Im Gegensatz zu diesen Verfahren wollen wir versuchen durch das Einführen einer abstrakten Ebene die unendliche Warteschlange besser zu simulieren und den Datenverlust durch das Abschneiden einzugrenzen.
In [7] hingegen werden unendliche CTMCs mit einer baumartigen Struktur betrachtet. Diese sind etwas komplexer als gewöhnliche QBDs. Hier wird ein Verfahren zum Zusammenfassen von Zuständen vorgestellt, um das Aufblähen des Zustandsraumes für eine größere Tiefe bzw. Levelzahl zu vermeiden. Diese Abstraktionmethode wollen auf unser Problem anpassen und zusammen mit der Diskretisierung aus [8] verwenden, um ein diskretes und endliches Modell zu erhalten. Wir versuchen so, einen anderen Ansatz für das Problem des Model Checkings unendlicher CTMCs, wie es in [11] diskutiert wurde, zu entwickeln.

2. Theorie und Grundlagen zur Abstraktion und Diskretisierung von QBDs

Im folgenden Kapitel werden wir die Grundlagen für die spätere Anwendung formalisieren und erklären. Wir definieren zunächst in 2.1 diskrete und kontinuierliche Markov-Ketten und Quasi-Birth-Death-Prozesse, führen anschließend in 2.2 eine bestimmte Form der Abstraktion von kontinuierlichen Markov-Ketten ein, die wir verwenden werden, um ein endliches Modell zu erhalten und definieren dafür auch ein nichtdeterministisches Modell: Markov-Entscheidungsprozesse. Anschließend legen wir in 2.3 die Diskretisierung dieser Modelle dar, durch die wir diskrete nichtdeterministische Modelle erhalten, auf denen wir später Wahrscheinlichkeiten für Erreichbarkeit bestimmter Zustände bestimmen wollen.

2.1. DTMCs, CTMCs, QBDs

Im Verlauf der Arbeit betrachten wir Zustandsübergangsmodelle, die komplexe Systeme darstellen, in denen mit Wahrscheinlichkeitsverteilungen oder Raten ausgehend von einem Zustand eine Zustandsänderung eintreten kann. Solche Prozesse können in einem stochastischen Modell als Markov-Kette modelliert werden. Für alle Markov-Ketten gilt die sogenannte Markov Eigenschaft. Diese besagt, dass die Wahrscheinlichkeit einer Transition in einem Zustand ausschließlich von diesem Zustand abhängt. Der bisherige Verlauf innerhalb der Markov-Kette ist nicht von Bedeutung für die Wahrscheinlichkeit oder Rate eines Übergangs. Ein solches zeitdiskretes Modell ist eine *Discrete-time Markov Chain*:

2.1.1 Definition Discrete-time Markov Chain (DTMC)
Eine DTMC ist ein Tupel $\mathcal{D} = (\mathcal{S}, \mathbb{P}, q, L)$, wobei $\mathcal{S}$ ist eine Zustandsmenge ist. $\mathbb{P}\colon \mathcal{S}\times\mathcal{S} \to [0,1]$ ist eine Übergangsfunktion, die die Zustandsübergänge widerspiegelt. Für jeden Zustand $s \in \mathcal{S}$ gilt $\sum_{s'\in\mathcal{S}} \mathbb{P}(s,s') = 1$. $q \in \mathrm{Distr}(\mathcal{S})$ aus der Menge aller Verteilungen über $\mathcal{S}$ ist die Startverteilung. Sei außerdem $L\colon \mathcal{S} \to 2^A$ eine Labelfunktion, die jedem Zustand eine Menge von atomaren Eigenschaften zuordnet. Dabei ist A die Menge aller atomaren Eigenschaften. [1]

In einem diskreten Modell finden Zustandsübergänge in regelmäßigen Abständen gemäß einer fest definierten Zeiteinheit statt. Um realitätsnahe Modelle zu erhalten muss kontinuierliche Zeit betrachtet werden. Für kontinuierliche Markov-Ketten bedeutet dies, dass

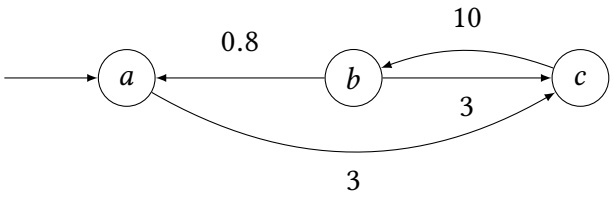


Abbildung 2.1.: Beispiel einer CTMC mit Zustandsraum $\mathcal{S} = \{a, b, c\}$, Startzustand a, der Ratenfunktion $\mathcal{R}\colon \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geqslant 0}$ mit $\mathcal{R}(a, c) = 3, \mathcal{R}(b, a) = 0.8, \mathcal{R}(b, c) = 3, \mathcal{R}(c, b) = 10$, sonst 0.

ein Zustandsübergang zu jedem beliebigen Zeitpunkt stattfinden kann und nicht, wie in einem diskreten Modell, nur nach Ablauf einer festen Zeitschrittdauer. Wir führen nun die Definition kontinuierlicher Markov-Ketten ein:

2.1.2 Definition Continuos-time Markov Chain (CTMC) [7], [6]
Eine CTMC ist ein Tupel $C = (\mathcal{S}, \mathcal{R}, q, L)$. Hierbei ist $\mathcal{S}$ ein Zustandsraum, $\mathcal{R}\colon \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geqslant 0}$ die Ratenfunktion, die die Zustandsübergänge widerspiegelt, $q \in \text{Distr}(\mathcal{S})$ die Startverteilung und $L : \mathcal{S} \to 2^A$ eine Labelfunktion, die jedem Zustand eine Menge von atomaren Eigenschaften aus A, der Menge aller atomaren Eigenschaften, zuordnet.

Die Ausgangsrate E eines Zustands $s \in S$ einer CTMC ist die Summe aller ausgehenden Transitionen:

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

In einer CTMC hat jeder Zustand eine Verweildauer. Diese Verweilzeiten sind exponentialverteilt – die Wahrscheinlichkeit, einen Zustand s innerhalb einer Zeit t zu verlassen ist demnach gegeben durch $1 - e^{-E(s)\cdot t}$. Für eine Transition mit der Rate μ beträgt die Wahrscheinlichkeit $\left(1 - e^{-E(s)\cdot t}\right) \cdot \mu/E(s)$. In Abbildung 2.1 wird ein Beispiel einer einfachen CTMC gezeigt. Ein Übergang von Zustand b in Zustand c geschieht hier mit einer Rate von 3.
Durch Modellierung von Warteschlangen erhält man CTMCs einer bestimmten Struktur mit sich wiederholenden Ebenen. Die Aufnahme eines neuen Elementes in die Warteschlange induziert im Modell eine Transition in die nächsthöhere Ebene, das Abarbeiten eines Elementes löst einen Zustandsübergang in eine tiefere Ebene aus. Aufgrund dieses Verhaltens werden diese Modelle auch Quasi-Birth-Death-Prozesse genannt. Die folgende Definition orientiert sich an QBDs aus [11].

2.1.3 Definition Quasi Birth Death Process (QBD)
Ein QBD ist eine CTMC unendlicher Länge der folgenden Struktur: Ein QBD besteht aus einer unendlichen Anzahl von Ebenen, auch Level genannt, mit endlich vielen Zuständen, Transitionen existieren nur innerhalb einer Ebene und zwischen direkt aufeinanderfolgenden Ebenen.
Ein QBD Q ist demnach ein Tupel $(\mathcal{S}, \mathcal{R}, q, L)$. Der Zustandsraum $\mathcal{S}$ lässt sich aufteilen in disjunkte Teilmengen $S_i \subset \mathcal{S}$ für $i \in \mathbb{N}$. Die Ratenfunktion $\mathcal{R}(s, s')$ hat nur Einträge > 0 für $s \in S_i$ mit $s' \in S_{i+1} \cup S_i \cup S_{i-1}$, für $s' \in \mathcal{S} \setminus (S_{i+1} \cup S_i \cup S_{i-1})$ gilt demnach $\mathcal{R}(s, s') = 0$. Dadurch ist die Ratenfunktion $\mathcal{R}$ eine Blockmatrix mit Untermatrizen $\mathcal{R}_{i,i-1}, \mathcal{R}_{i,i}, \mathcal{R}_{i,i+1}$:

$$\mathcal{R} = \begin{pmatrix} \mathcal{R}_{0,0} & \mathcal{R}_{0,1} & & \\ \mathcal{R}_{1,0} & \mathcal{R}_{1,1} & \mathcal{R}_{1,2} & \\ & \ddots & \ddots & \ddots \end{pmatrix}.$$

Dabei ist $\mathcal{R}_{i,j}$ die Matrix $\mathcal{R}(s_i, s_j)$ mit $s_i \in S_i, s_j \in S_j$.

Eine wichtige Eigenschaft eines QBD ist *Levelunabhängigkeit*, welche eine Aussage über unterschiedliche Strukturen der Ebenen trifft.

2.1.4 Definition Levelunabhängigkeit [11]
Ein QBD ist ab einer bestimmten Ebene $i \in \mathbb{N}$ levelunabhängig. Dies bedeutet, dass sich ab diesem Punkt die Struktur der Ebenen wiederholt: Für jeden Zustand s_k dieser Ebene S_i existiert in allen folgenden Ebenen $S_j, j > i$, ein entsprechender Zustand $s_k \in S_j$. Auch die Transitionen und atomaren Eigenschaften verhalten sich entsprechend.

Im Folgenden betrachten wir ausschließlich QBDs mit Levelunabhängigkeit ab dem zweiten Level, einzig das erste Level darf sich also von den anderen unterscheiden. Dieses Level nennen wir Grundlevel, alle folgenden Ebenen Wiederholende Level.

In Abbildung 2.2 erkennt man die in 2.1.3 beschriebene Struktur eines typischen QBD. Der QBD ist levelunabhängig ab dem zweiten Level, alle folgenden Ebenen haben die gleiche, sich wiederholende, Struktur. Nur die erste Ebene, das Grundlevel, hat eine abweichende Struktur.

2.2. MDP-Abstraktion

Um eine CTMC zu abstrahieren, kann man eine Abstraktionsmethode verwenden, bei der die CTMC in einen *Continuos-time Markov Decision Process* (CTMDP) umgewandelt wird. Diese Methode heißt *MDP-Abstraktion*. Hierbei werden Zustände nach einer gewählten Zustandsraumpartition zusammengefasst. Um die Unterschiede der zusammengefassten Zustände zu modellieren, werden in den abstrakten Zuständen sogenannte Aktionen eingeführt.

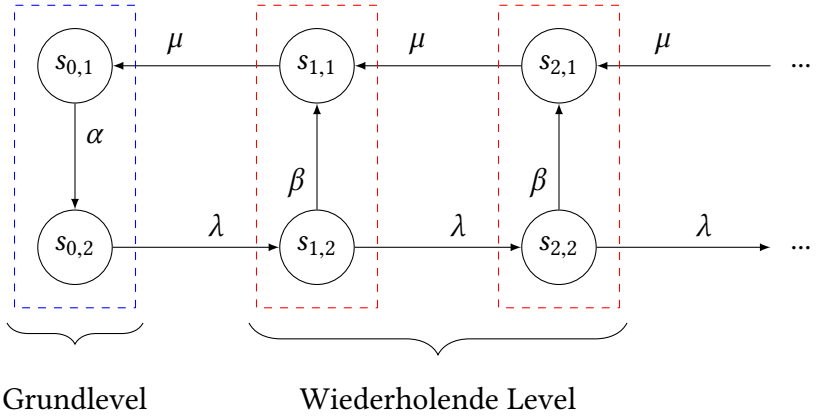


Abbildung 2.2.: Beispiel eines QBD mit Zustandsraum $\mathcal{S} = \{s_{i,j} \mid i \in \mathbb{N}, j \in \{1,2\}\}$. Jede Ebene i dieses QBD hat zwei Zustände, $s_{i,1}$ und $s_{i,2}$. Die Transitionen in ein höheres Level finden mit Rate λ statt, die in ein niedrigeres mit Rate μ.

2.2.1 Definition Continuos-time Markov Decision Process (CTMDP) [8]
Ein CTMDP $\mathcal{M}$ ist ein Tupel $\mathcal{M} = (\mathcal{S}, \text{Act}, \mathcal{R}, q, L)$ für das gilt:

- $\mathcal{S}$ ist ein endlicher Zustandsraum,
- Act ist eine endliche Menge von Aktionen, wobei Act(s) für $s \in S$ die Menge der möglichen Aktionen eines Zustandes angibt,
- $\mathcal{R}\colon \mathcal{S} \times \text{Act} \times \mathcal{S} \to \mathbb{R}_{\geqslant 0}$ ist eine dreidimensionale Ratenfunktion
- $q \in \text{Distr}(\mathcal{S})$ ist eine Initialverteilung
- und $L\colon \mathcal{S} \to 2^A$ eine Labelfunktion, wobei A eine Menge von atomaren Eigenschaften ist.

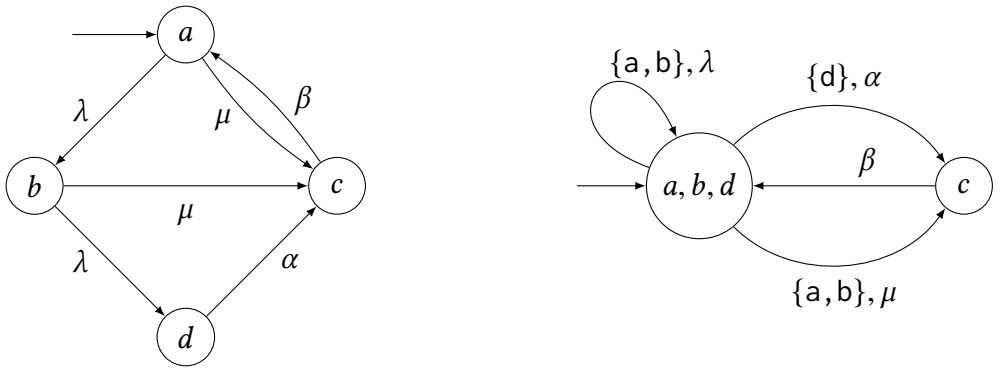
Die Ausgangsrate E eines CTMDP-Zustandes $s \in \mathcal{S}$ hängt auch von der gewählten Aktion $\alpha \in \text{Act}(s)$ ab:

$$E(s, \alpha) = \sum_{s' \in S} \mathcal{R}(s, \alpha, s')$$

Ein CTMDP heißt locally uniform, falls die Ausgangsrate E eines Zustands $s \in \mathcal{S}$ nicht von der Wahl der Aktion abhängt. Es gilt also für beliebige $\alpha, \beta \in \text{Act}(s)$:

$$E(s, \alpha) = E(s, \beta).$$

Im Folgenden seien CTMDPs falls nicht anders angegeben immer locally uniform, wir verwenden daher $E(s)$ für die Ausgangsrate eines Zustandes $s \in \mathcal{S}$.



(a) CTMC C (b) CTMDP C_M

Abbildung 2.3.: CTMC $C = (\mathcal{S}, \mathcal{R}, q, L)$ mit $\mathcal{S} = \{a, b, c\}$, Startzustand a und die zugehörige abstrahierte CTMDP C_M.

Ein CTMDP ist ein nichtdeterministisches Modell. In jedem Zustand gibt es die Möglichkeit, aus einer oder mehr *Aktionen* auszuwählen. Eine Aktion α aus Act(s) gibt für den Zustand s eine Menge von möglichen Transitionen an. Diese Transitionen sind, wie in einer CTMC, mit der Rate $r = \mathcal{R}(s, \alpha, s')$ exponentialverteilt.
Wenn ein CTMDP durch MDP-Abstraktion einer CTMC entsteht, spiegelt für jeden Zustand s, der in einen abstrakten Zustand integriert wurde, genau eine Aktion die Transitionen dieses Zustandes s dar. Falls zwei Zustände die gleiche Verteilung haben, so werden diese in einer Aktion zusammengefasst. Führte zuvor eine Transition in einen Zustand s, der nun mit anderen Zuständen zu einem abstrakten Zustand s' zusammengefasst wurde, so führt diese Transition im abstrahierten Modell in den neuen Zustand s'. [7]
In Abbildung 2.3 wird die MDP-Abstraktion an einem einfachen Beispiel illustriert. Als Zustandsraumpartition werden in diesem Beispiel die Zustände a, b und d zusammengefasst. Da a und b beide je eine μ-Transition nach c und eine λ-Transition nach b bzw. d, also in den neuen abstrakten Zustand (a, b, d) haben, können sie in einer Aktion zusammengefasst werden. d hat eine andere Verteilung, daher ensteht eine eigene, mit `d` betitelte Aktion an der ursprünglich von d ausgehenden Transition.

MDP-Abstraktion angewandt auf QBDs

Wir werden mithilfe der MDP-Abstraktion QBDs in endliche Modelle überführen. Dafür wählen wir die Zustandsraumaufteilung so, dass nach einer bestimmten Ebene m, die wir im Folgenden Abstraktionslevel nennen werden, alle sich entsprechenden Zustände in einem abstrakten Level a zusammengefasst werden. Das abstrakte Level hat eine ähnliche Struktur mit der gleichen Anzahl von Zuständen wie die Wiederholenden Ebenen. Die Transitionen sind in diesem Level aufgrund der Abstraktion anders.
Durch die so gewählte Partition entstehen nur im abstrakten Level verschiedene Aktionen. Formal definieren wir für alle Zustände, in denen sich durch die Abstraktion nichts ver-

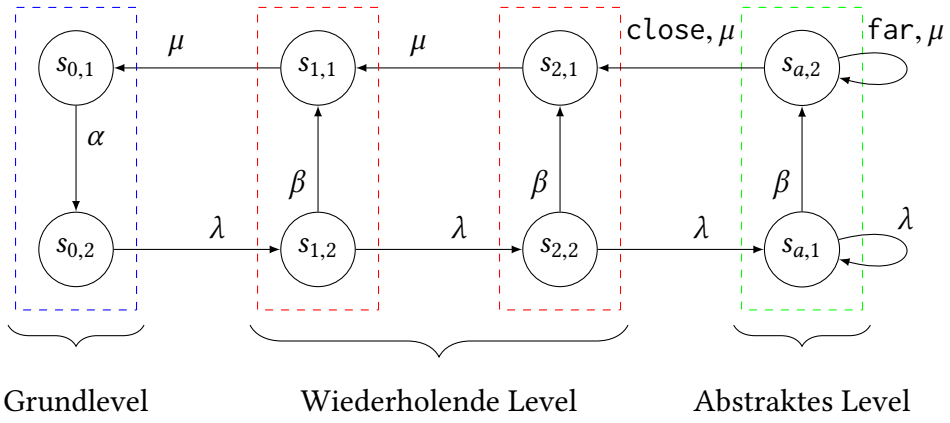


Abbildung 2.4.: Beispiel eines CTMDP, der durch Anwendung von MDP-Abstraktion mit Abstraktionslevel $m = 2$ auf den in Abbildung 2.2 eingeführten QBD entsteht.

ändert, eine Aktion `default`, da wir in einem CTMDP, der so entsteht, in jedem Zustand eine Aktion wählen können müssen. Dies gilt insbesondere für alle Zustände aus den beibehaltenen Leveln mit den Zustandsmengen $S_0, \ldots, S_m$.
Für abstrakte Zustände, deren korrespondieren Zustände keine Transitionen in ein vorheriges Level haben, ändert sich durch die Abstraktion sehr wenig, da keine Wahlmöglichkeit entsteht. Solche Zustände besitzen die gleichen Transitionen wie die korrespondierenden Zustände der Wiederholenden Ebenen, nur entsprechende Übergänge in ein höheres Level zeigen hier ins abstrakte Level. Diese Zustände erhalten aufgrund der nur sehr geringen strukturellen Veränderungen auch nur die Aktion `default`.
Falls für einen abstrakten Zustand die ihm entsprechenden Zustände eine oder mehr Transitionen in ein vorheriges Level haben, entstehen zwei verschiedene Aktionen:

- `close`: Diese Aktion modelliert genau den Zustand des ersten nicht mehr bestehenden Levels $m + 1$. Demnach sind die Transitionen ins vorherige Level sind mit `close` möglich. Die Transitionen in ein weiteres oder innerhalb des Levels zeigen nun auf den korrespondierenden Zustand der abstrakten Ebene, da die Ziellevel auch durch das abstrakte Level modelliert werden.

- `far`: Diese Aktion stellt alle Zustände der übrigen wegfallenden Level $> m + 1$ dar. Von diesen Zuständen aus kann die Ebene m nicht mehr erreicht werden. Auch die Transitionen, die im QBD in ein niedrigeres Level zeigen, erreichen hier die jeweils korrespondierenden Zustände in der abstrakten Ebene S_a.

In Abbildung 2.4 ist dieses Verhalten dargestellt. Man erkennt hier die MDP-Abstraktion mit den Aktionen `far` und `close` anhand des QBDs aus Abbildung 2.2. Aus Übersichtsgründen werden hier und in allen weiteren Grafiken die Transitionen der `default`-Aktion nicht explizit gekennzeichnet.

Auf Grundlage der MDP-Abstraktion, wie sie in [7] eingeführt und hier vorgestellt wurde, entsteht für unsere Betrachtungen bezüglich QBDs die folgende Definition, die Zustandsraum und Ratenfunktion des abgeleiteten CTMDPs formalisiert.

2.2.2 Definition MDP-Abstraktion eines QBD
Durch MDP-Abstraktion mit Abstraktionslevel m eines QBD $\mathcal{Q} = (\mathcal{S}, \mathcal{R}, q, L)$ mit $\mathcal{S} = \bigcup_{i \in \mathbb{N}} S_i$ entsteht ein CTMDP $\mathcal{M} = (\hat{\mathcal{S}}, \hat{\mathcal{R}}, \text{Act}, q, L)$, wobei $\hat{\mathcal{S}} = S_0 \cup \cdots \cup S_m \cup S_a$. Seien $s_{i,1}, \ldots, s_{i,n} \in S_i$ für $i \in \{0, \ldots, m, a\}$ die Zustände einer Ebene i. S_a ist die Zustandsmenge des abstrakten Levels und enthält zu $s_{i,1}, \ldots, s_{i,n} \in S_i$ für $i \in \{1, \ldots, m\}$ korrespondierende Zustände $s_{a,1}, \ldots, s_{a,n} \in S_a$.

Die Ratenfunktion $\hat{\mathcal{R}}$ ist folgendermaßen definiert:

Für Zustände $s_{i,k} \in S_i, s' \in \mathcal{S}$ mit $i \in \{1, \ldots, m-1\}, k \in \{1, \ldots, n\}$:

$$\hat{\mathcal{R}}(s_{i,k}, \texttt{default}, s') := \mathcal{R}(s_{i,k}, s')$$

Für Zustände $s_{m,k} \in S_m, k, p \in \{1, \ldots, n\}$:

$$\hat{\mathcal{R}}(s_{m,k}, \texttt{default}, s_{m-1,k}) := \mathcal{R}(s_{m,k}, s_{m-1,p})$$
$$\hat{\mathcal{R}}(s_{m,k}, \texttt{default}, s_{m,k}) := \mathcal{R}(s_{m,k}, s_{m,p})$$
$$\hat{\mathcal{R}}(s_{m,k}, \texttt{default}, s_{a,k}) := \mathcal{R}(s_{m,k}, s_{m+1,p})$$

Für Zustände $s_{a,k} \in S_a$ mit $i \in \{1, \ldots, m\}, k, p \in \{1, \ldots, n\}$:

Falls $\mathcal{R}(s_{i,k}, s_{i-1,z}) = 0$ mit $z \in \{1, \ldots, n\}$:

$$\hat{\mathcal{R}}(s_{a,k}, \texttt{default}, s_{a,p}) := \mathcal{R}(s_{i,k}, s_{i,p}) + \mathcal{R}(s_{i,k}, s_{i+1,p})$$
$$\hat{\mathcal{R}}(s_{a,k}, \texttt{close}, s_{m,p}) := 0$$
$$\hat{\mathcal{R}}(s_{a,k}, \texttt{close}, s_{a,p}) := 0$$
$$\hat{\mathcal{R}}(s_{a,k}, \texttt{far}, s_{a,p}) := 0$$

Sonst:

$$\hat{\mathcal{R}}(s_{a,k}, \texttt{default}, s_{a,p}) := 0$$
$$\hat{\mathcal{R}}(s_{a,k}, \texttt{close}, s_{m,p}) := \mathcal{R}(s_{i,k}, s_{i-1,p})$$
$$\hat{\mathcal{R}}(s_{a,k}, \texttt{close}, s_{a,p}) := \mathcal{R}(s_{i,k}, s_{i,p}) + \mathcal{R}(s_{i,k}, s_{i+1,p})$$
$$\hat{\mathcal{R}}(s_{a,k}, \texttt{far}, s_{a,p}) := \mathcal{R}(s_{i,k}, s_{i-1,p}) + \mathcal{R}(s_{i,k}, s_{i,p}) + \mathcal{R}(s_{i,k}, s_{i+1,p})$$

Alle nicht explizit definierten Raten sind 0.

Mit dieser Definition haben wir nun alle Grundlagen, die zum Abstrahieren einer CTMC benötigt werden, kennengelernt und die Definition der MDP-Abstraktion für unsere Problemstellung, Model Checking von QBDs, formalisiert. Um dies Analysieren zu können, gehen wir im nächsten Unterkapitel auf die Diskretisierung ein.

2.3. Diskretisierung von CTMDPs

Wie in [2] werden wir CTMDPs diskretisieren, um ein Modell zu erhalten, auf das der PRISM Model Checker angewandt werden kann. Wir werden die Diskretisierung auf die durch MDP-Abstraktion von QBDs erhaltenen CTMDPs anwenden. Dabei wird die Übergangsfunktion anhand eines möglichst klein gewählten Zeitschrittes diskretisiert. Wir beziehen also die Wahrscheinlichkeit, einen Schritt zu machen, in die Übergangswahrscheinlichkeit mit ein. Dieses Vorgehen ist in dieser Form nur möglich, da die betrachtete CTMDP die Eigenschaft hat, locally uniform zu sein. Durch Anwenden der Methode entsteht aus dem kontinuierlichen CTMDP ein diskreter Markov-Entscheidungsprozess mit einer speziellen Übergangsfunktion. Wir definieren zunächst diskrete Markov-Entscheidungsprozesse:

2.3.1 Definition Markov Decision Process (MDP) [1]
Ein MDP $\mathcal{M}$ ist ein Tupel $\mathcal{M} = (\mathcal{S}, \mathrm{Act}, \mathbb{P}_\tau, q, L)$. Hier ist $\mathcal{S}$ ein endlicher Zustandsraum, Act eine Menge von Aktionen, wobei $\mathrm{Act}(s)$ für $s \in \mathcal{S}$ die Menge der möglichen Aktionen eines Zustandes angibt, $\mathbb{P} : \mathcal{S} \times \mathrm{Act} \times \mathcal{S} \to [0,1]$ die Übergangsfunktion, $q \in \mathrm{Distr}(\mathcal{S})$ die Startverteilung und $L : \mathcal{S} \to 2^A$ eine Labelfunktion, die jedem Zustand eine atomare Eigenschaft aus der Menge A zuweist. A ist die Menge aller atomaren Eigenschaften.

In der folgenden Definition wird die Übergangsfunktion beschrieben, wie sie durch die Diskretisierung entsteht.

2.3.2 Definition Diskretisierte Übergangsfunktion [8]
Die diskretisierte Wahrscheinlichkeitsfunktion $\mathbb{P}_\tau : \mathcal{S} \times \mathrm{Act} \times \mathcal{S} \to [0,1]$ ist definiert als

$$\mathbb{P}_\tau(s,a,s') = \begin{cases} \left(1 - e^{-E(s)\cdot\tau}\right) \cdot \mathbb{P}(s,a,s'), & \text{für } s \neq s', a \in \mathrm{Act}(s) \\ \left(1 - e^{-E(s)\cdot\tau}\right) \cdot \mathbb{P}(s,a,s') + e^{-E(s)\cdot\tau}, & \text{für } s = s', a \in \mathrm{Act}(s) \\ 0, & \text{für } a \notin \mathrm{Act}(s), \end{cases}$$

wobei $\tau \in \mathbb{R}_{\geq 0}$ die Dauer eines Zeitschrittes ist. $\mathbb{P} : \mathcal{S} \times \mathrm{Act} \times \mathcal{S} \to [0,1]$ gibt die zeitlich unabhängige Wahrscheinlichkeit an, mit der gewählten Aktion $a \in \mathrm{Act}(s)$ von Zustand s aus zu Zustand s' zu wechseln. Es gilt:

$$\mathbb{P}(s,a,s') = \frac{\mathcal{R}(s,a,s')}{E(s)}.$$

Die Exponentialverteilung $1 - e^{-E(s)\cdot\tau}$ gibt hierbei die Wahrscheinlichkeit für das Eintreten eines Ereignisses innerhalb des Zeitintervalls der Länge τ an. τ ist die Granularität der diskreten Zeit, mit der wir Kontinuität simulieren wollen, weshalb τ möglichst klein gewählt werden muss.
Werden mithilfe des erhaltenen Modells Berechnungen durchgeführt, so ist die Wahl von τ verantwortlich für die Genauigkeit der Ergebnisse. Für ein zu groß gewähltes τ entsteht ein Fehler. Dieser Fehler lässt sich berechnen, worauf wir im Folgenden näher eingehen

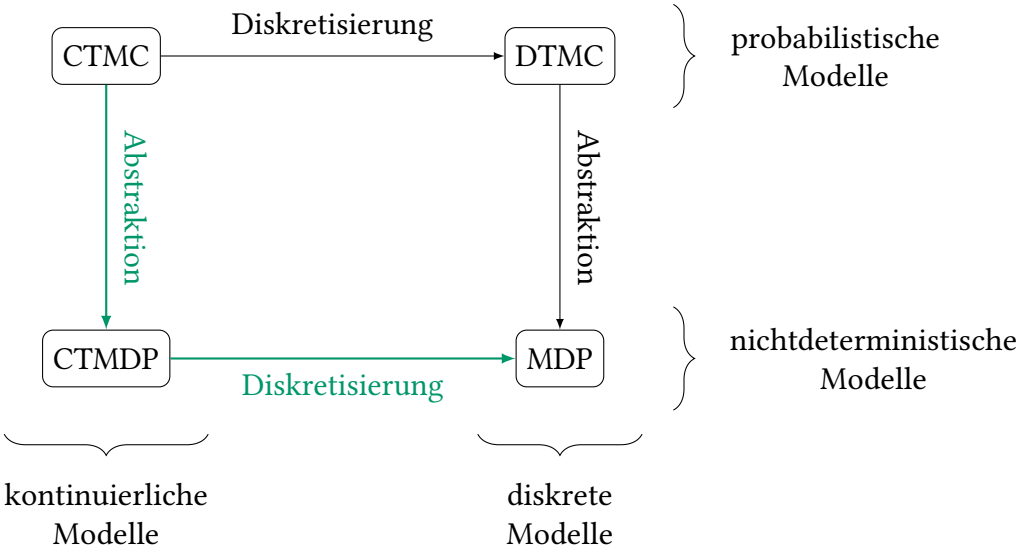


Abbildung 2.5.: In der Arbeit vorgestellte stochastische Zustandsübergangsmodelle. Mit MDP-Abstraktion erhält man aus einem probabilistischem Modell ein nichtdeterministisches. Mit Diskretisierung kann man ein kontinuierliches Modell in ein diskretes überführen.

werden, nachdem wir die Berechnung von Wahrscheinlichkeiten eingeführt haben.

Fazit Wir haben nun verschiedene Zustandsübergangsmodelle im diskreten und kontinuierlichen Raum kennengelernt die jeweils probabilistisch oder auch nichtdeterministisch sein können. Außerdem haben wir zwei Methoden erlernt: MDP-Abstraktion fasst Zustände eines probabilistischen Modells in abstrakten Zuständen zusammen. Dadurch können wir ein unendliches Modell wie einen QBD in ein endliches überführen. Diskretisierung kann aus einem kontinuierlichen Modell wie einem CTMDP mithilfe einer Granularität τ die Übergangsfunktion und damit das Modell in ein diskretes überführen. In Abbildung 2.5 ist dies grafisch festgehalten.

3. Erreichbarkeit in MDPs

Auf einem stochastischen Modell wie einem MDP können wir verschiedene relevante Eigenschaften auf ihre Eintrittswahrscheinlichkeiten überprüfen. Um diese Berechnungen durchführen zu können, werden wir in 3.1 noch einige Grundlagen darlegen. Anschließend definieren wir in 3.2 die Logik PCTL sowie zeitgebundene Erreichbarkeit für MDPs und führen in 3.3 Möglichkeiten zur Berechnung der Wahrscheinlichkeiten dieser Eigenschaft ein.

3.1. Pfade, Scheduler und induzierte DTMCs

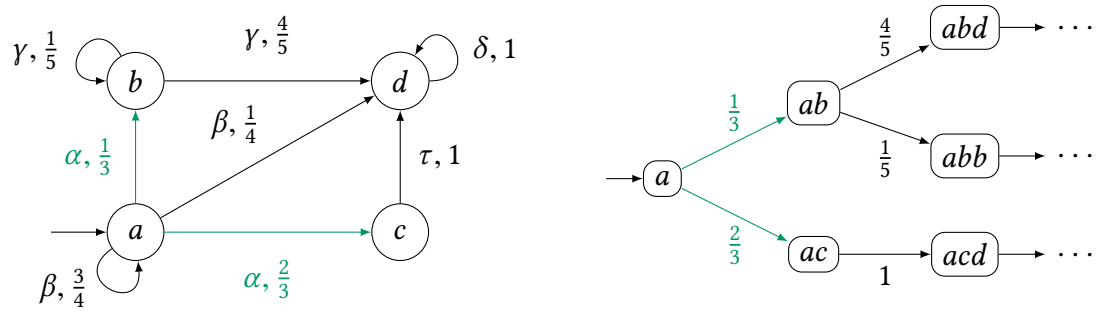
Die folgende Definition eines Pfades erfolgt in Anlehnung an [1].

3.1.1 Definition Pfad
Ein Pfad $\pi = s_0 s_1 \dots$ auf einem MDP $\mathcal{M}$, ausgehend von einem Zustand s_0 ist eine Sequenz von Zuständen, sodass gilt:

$$\forall s_i \in \pi \; \exists \alpha \in \mathrm{Act} : \mathbb{P}(s_i, \alpha, s_{i+1}) \in \mathbb{R}_{\geqslant 0}.$$

Die Menge aller möglichen Pfade auf einem MDP $\mathcal{M}$ nennen wir $\mathrm{Paths}_{\mathcal{M}}$. Die Menge aller Pfade, die von einem Zustand s aus möglich sind nennen wir $\mathrm{Paths}_{\mathcal{M}}(s)$.

Ein Pfad ist also eine Folge von Zuständen, wobei für aufeinanderfolgende Zustände eine Transition existieren muss. Da MDPs nichtdeterministische Modelle sind, hängt der Pfadverlauf in einem MDP in allen Zuständen mit Wahlmöglichkeit nicht nur von der Wahrscheinlichkeitsverteilung, sondern auch von der Wahl einer Aktion ab. Um diese Wahl in die Berechnung von Wahrscheinlichkeiten miteinbeziehen zu können, kann man sie durch einen Scheduler formalisieren. Auch die Definition für Scheduler ist angelehnt an die entsprechende Definition in [1].



(a) MDP $\mathcal{M}$ (b) Induzierte DTMC $\mathcal{M}_\sigma$

Abbildung 3.1.: Beispiel einer induzierten DTMC durch einen Scheduler σ. Der Scheduler wählt hier in Zustand a der MDP die Aktion α, wodurch ausgehend von diesem Zustand die beiden Pfade ab und ac zu möglichen Pfaden werden. Diese Aktion ist im MDP und in der induzierten DTCM markiert.

3.1.2 Definition Scheduler
Ein Scheduler (auch: Adversary, Policy) für einen MDP $\mathcal{M} = (\mathcal{S}, \text{Act}, \mathbb{P}, q, L)$ ist eine Funktion $\sigma : \text{Paths}_\mathcal{M} \to \text{Act}$, sodass für einen gegebenen Pfad $\pi = s_0 \dots s_n \in \text{Paths}_\mathcal{M}$ gilt:

$$\sigma(\pi) \in \text{Act}(s_n).$$

Ein σ–Pfad ist eine Sequenz $\pi = s_0\alpha_0 s_1\alpha_1 s_2 \dots$ von Zuständen $s_i \in \mathcal{S}$ und Aktionen $\alpha_i \in \text{Act}$ mit:

$$\alpha_i = \sigma(s_0 \dots s_i)\ \forall i > 0.$$

Die Menge aller σ–Pfade auf einem MDP $\mathcal{M}$, ausgehend von einem Zustand s nennen wir $\text{Paths}_\sigma(s)$. Es gilt $\text{Paths}_\sigma(s) \subseteq \text{Paths}_\mathcal{M}(s)$.

Einen Scheduler kann man sich wie eine Person vorstellen, die entscheidet, welche Aktion in einem Zustand gewählt wird. Es gibt eine Vielzahl von Kriterien um Scheduler zu beschreiben. Eines davon ist *Gedächtnislosigkeit*. Das bedeutet, dass die Entscheidungen des Schedulers unabhängig von den bisher getroffenen Entscheidungen sind.

3.1.3 Definition Memoryless Scheduler [1]
Ein Scheduler auf einem MDP $\mathcal{M} = (\mathcal{S}, \text{Act}, \mathbb{P}, q, L)$ wird memoryless (oder auch gedächtnislos) genannt, wenn für alle Pfade $\pi, \pi' \in \text{Paths}_\mathcal{M}$ mit $s \in \mathcal{S}$ gilt:

$$\sigma(\pi s) = \sigma(\pi' s).$$

Ein Scheduler σ auf einem MDP $\mathcal{M}$ induziert eine DTMC $\mathcal{M}_\sigma$, da er den Nichtdeterminismus durch die Wahl einer Aktion auf jedem möglichen Pfad auflöst. Diese DTMC

formalisiert das Verhalten des MDP unter dem Scheduler σ. [1] Sie ist folgendermaßen definiert:

3.1.4 Definition Induced Discrete-Time Markov Chain [1]
Die durch einen Scheduler σ induzierte DTMC $\mathcal{M}_\sigma$ eines MDP $\mathcal{M} = (\mathcal{S}, \text{Act}, \mathbb{P}, q, L)$ ist ein Tupel

$$\mathcal{M}_\sigma = (\text{Paths}_\sigma(q), \mathbb{P}_\sigma, q, L_\sigma).$$

Dabei ist

- $\text{Paths}_\sigma(q)$ der Zustandsraum der induzierten DTMC. Die Zustände repräsentieren alle möglichen Pfade auf dem MDP für den Scheduler σ.
- $\mathbb{P}_\sigma : \text{Paths}_\sigma(q) \to [0, 1]$ die Übergangsfunktion. Eine Transition repräsentiert einen weiteren Schritt auf dem MDP mit der Wahl einer Aktion durch den Scheduler. Es gilt $\mathbb{P}_\sigma(\pi, \pi s_{n+1}) = \mathbb{P}(s_n, \sigma(\pi), s_{n+1})$ für alle $\pi = s_0 \dots s_n \in \text{Paths}_\sigma(q)$.
- q die Initialverteilung.
- $L_\sigma : \text{Paths}_\sigma(q) \to A$ eine Labelfunktion, wobei jedem Zustand $\pi = s_0 \dots s_n \in \text{Paths}_\sigma(q)$ der induzierten DTMC das Label des letzten Pfadzustandes zugewiesen wird. Es gilt $L_\sigma(\pi) = L(s_n)$.

Die durch einen Scheduler σ induzierte DTMC besteht also aus den möglichen σ-Pfaden auf dem MDP. Ist der Scheduler gedächtnislos, so kann eine vereinfachte induzierte DTMC betrachtet werden: Diese hat den gleichen Zustandsraum wie der MDP. In jedem Zustand bleibt dann nur die Verteilung derjenigen Aktion bestehen, die der Scheduler wählt.
In Abbildung 3.1 ist ein Beispiel für einen MDP und die zugehörige induzierte DTMC für den Scheduler mit der folgenden Funktion zu sehen:

$$\sigma(\pi) = \begin{cases} \alpha, & \text{falls } \pi = s_0 \dots s_n a \\ \gamma, & \text{falls } \pi = s_0 \dots s_n b \\ \tau, & \text{falls } \pi = s_0 \dots s_n c \\ \delta, & \text{falls } \pi = s_0 \dots s_n d \end{cases}, \qquad s_0, \dots, s_n \in \mathcal{S} = \{a, b, c, d\}, \pi \in \text{Paths}_\sigma(q).$$

3.2. Logik

Um nun eine Wahrscheinlichkeit dafür berechnen zu können, von einem Zustand aus einen anderen Zustand zu erreichen, müssen wir zunächst formalisieren, was es heißt, *erreichbar* zu sein. Dafür führen wir *Probabilistic Computational Tree Logic* (PCTL) ein, eine Logik, mit der man Wahrheitsaussagen auf MDPs formulieren kann.

3.2.1 Definition Probabilistic Computational Tree Logic (PCTL) [1]
Probabilistic Computational Tree Logic (PCTL) ist eine Logik zum Beschreiben von Eigenschaften von Zuständen in Markov-Ketten und insbesondere MDPs. In PCTL gibt es Zustands- und Pfadformeln, die der hier definierten Syntax folgen.

$$\text{Zustandsformeln: } \phi ::= \texttt{true} \mid \texttt{a} \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid P_J(\psi),$$

wobei $a \in A$ eine atomare Eigenschaft ist, A die Menge aller atomaren Eigenschaften, ψ eine Pfadformel, ϕ, ϕ_1 und ϕ_2 Zustandsformeln und $J \subseteq [0, 1]$ ein Intervall mit Grenzen aus $\mathbb{Q}$.

$$\text{Pfadformeln: } \psi ::= \mathtt{X}\phi \mid \phi_1 \ \mathtt{U} \ \phi_2 \mid \phi_1 \ \mathtt{U}^{\leqslant n} \ \phi_2$$

Hier sind ϕ, ϕ_1 und ϕ_2 Zustandsformeln und n ein Schwellwert, der die Anzahl von Zeitschritten angibt, innerhalb derer eine Eigenschaft geprüft werden soll. Die Bedeutung der temporalen Verknüpfungen $\mathtt{X}$, $\mathtt{U}$ und $\mathtt{U}^{\leqslant n}$ wird in der folgenden Definition erläutert.

3.2.2 Definition Erfüllen von Zustands- und Pfadformeln [1]
Eine Zustand s erfüllt (satisfies) eine Zustandsformel ϕ, falls gilt $s \models \phi$, d.h. die Zustandsformel ϕ gilt in s. Diese Relation ist folgendermaßen definiert:

$s \models \mathtt{true}$,	für alle Zustände s.
$s \models \mathtt{a}$,	genau dann, wenn $\mathtt{a} \in L(s)$.
$s \models \neg\phi$,	genau dann, wenn $s \not\models \phi$.
$s \models \phi_1 \wedge \phi_2$,	genau dann, wenn $s \models \phi_1 \wedge s \models \phi_2$.
$s \models P_{\mathtt{J}}(\psi)$,	genau dann, wenn $Pr(s, \psi) \in \mathtt{J}$.

Dabei ist $Pr(s, \psi) = Pr_s\{\pi \in \text{Paths}(s) \mid \pi \models \psi\}$ die Wahrscheinlichkeit, von s aus einen Pfad zu gehen, der ψ erfüllt.

Ein Pfad π in einer MDP $\mathcal{M}$ erfüllt eine Pfadformel ψ, falls gilt $\pi \models \psi$:

$\pi \models \mathtt{X}\phi$,	falls $\pi[1] \models \phi$
$\pi \models \phi_1 \ \mathtt{U} \ \phi_2$,	falls $\exists j \geqslant 0 : (\pi[j] \models \phi_2 \wedge (\forall 0 \leqslant k < j : \pi[k] \models \phi_1))$
$\pi \models \phi_1 \ \mathtt{U}^{\leqslant n} \ \phi_2$,	falls $\exists j \in \mathbb{N} : 0 \leqslant j \leqslant n : (\pi[j] \models \phi_2 \wedge (\forall 0 \leqslant k < j : \pi[k] \models \phi_1))$

Dabei ist $\pi = s_0 s_1 \ldots$ ein Pfad und $\pi[i]$ bezeichnet für $i \geqslant 0$ den Zustand s_i.

Mithilfe dieser Formeln sind nach Definition verschiedene Eigenschaften formal spezifiziert.

$\mathtt{X}\phi$: Ein Pfad erfüllt diese Formel, falls der nächste Zustand des Pfades ϕ erfüllt. Diese Eigenschaft nennt man next.

$\phi_1 \ \mathtt{U} \ \phi_2$: Ein Pfad erfüllt diese Formel, falls für alle Zustände entlang des Pfades ϕ_1 gilt, bis für einen Zustand auf dem Pfad ϕ_2 gilt. Zu dieser Eigenschaft sagt man until.

$\phi_1 \ \mathtt{U}^{\leqslant n} \phi_2$: Ein Pfad erfüllt diese Formel, falls innerhalb von n Schritten auf dem Pfad ein ϕ_2 Zustand erreicht wird und bis zu diesem Zustand für alle Zustände ϕ_1 gilt. Aufgrund der zeitlichen Beschränktheit nennt man diese Eigenschaft bounded until.

Erreichbarkeit ist ein Spezialfall der Unil-Formeln. Dabei gilt $\phi_1 = \mathtt{true}$. Die Erreichbarkeit einer Zustandsformel ϕ ist demnach $\mathtt{true} \ \mathtt{U} \ \phi$. Wir können auch *zeitgebundene Erreichbarkeit* formalisieren:

$$\mathtt{true} \ \mathtt{U}^{\leqslant n} \ \phi.$$

Ein Pfad erfüllt diese Formel, wenn innerhalb von n Schritten ein Zustand auf dem Pfad die Zustandsformel ϕ erfüllt. Man spricht deshalb auch vom *Erreichen* von ϕ.

3.3. Berechnung von Erreichbarkeitswahrscheinlichkeiten

Wie bereits erwähnt können durch den Nichtdeterminismus in MDPs keine eindeutigen Wahrscheinlichkeiten berechnet werden. Betrachtet man allerdings in jedem Zustand die beste bzw. schlechteste Wahl für das Erfüllen der Eigenschaft, so kann eine maximale und eine minimale Wahrscheinlichkeit berechnet werden. Diese Wahl ist formalisiert durch einen Scheduler und die entsprechende induzierte DTMC. Der Scheduler mit dem besten Verhalten in Bezug auf die Pfadformel induziert diejenige DTMC, auf der man die maximale Wahrscheinlichkeit für das Erfüllen der Pfadformel berechnen kann. Hierfür berechnet man das Supremum (bzw. Infimum) der Wahrscheinlichkeiten, die durch alle möglichen Scheduler induziert werden.

3.3.1 Definition Wahrscheinlichkeit für eine Pfadformel [5]
Für einen MDP $\mathcal{M} = (\mathcal{S}, \text{Act}, \mathbb{P}, q, L)$ ist p_{max} die maximale und p_{min} minimale Wahrscheinlichkeit, von einem Zustand $s \in \mathcal{S}$ aus eine Pfadformel zu erfüllen. Es gilt:

$$p_{max}(s, \psi) = \sup_{\sigma \in Adv} Pr^{\sigma}(s, \psi),$$
$$p_{min}(s, \psi) = \inf_{\sigma \in Adv} Pr^{\sigma}(s, \psi),$$

wobei ψ eine Pfadformel ist und $Pr^{\sigma}(s, \psi)$ die Wahrscheinlichkeit bezeichnet, von s ausgehend einen σ-Pfad zu gehen, der ψ erfüllt.

Wir betrachten *time-bounded reachability*, die Eigenschaft, einen bestimmten Zustand oder eine Menge innerhalb einer vorgegebenen Zeit oder auch Anzahl von Schritten zu erreichen. Es gilt also $\psi = \texttt{true}\ \mathsf{U}^{[0,t]}\phi$. Zum Berechnen der maximalen und minimalen Wahrscheinlichkeiten für die Erreichbarkeit einer Zustandseigenschaft in einem MDP gibt es verschiedene Verfahren, die auf der Lösung eines Gleichungssystems basieren. Um alle benötigten Informationen zum Lösen dieser Gleichungen zu haben, muss zuvor noch die Berechnung einer bestimmten Teilzustandsmenge durchgeführt werden. $S^{\min=0}$ bzw. $S^{\max=0}$ beschreiben die Menge aller Zustände, von denen aus die Zielzustandsmenge (unter Betrachtung der jeweiligen gewünschten Optimalität min bzw. max) nicht erreicht werden kann. Für die Berechnung dieser Mengen verwenden wir die in [5] vorgestellten Algorithmen. Wir passen die Darstellung des Algorithmus an die in dieser Arbeit verwendete Notation an.
$S^{\min=0}$ enthält diejenigen Zustände, für die die Wahrscheinlichkeit die Zielmenge zu erreichen 0 ist, falls immer die möglichst schlechteste Wahl zum Erreichen der Zielmenge getroffen wird. Der Algorithmus für die Berechnung von $S^{\min=0}$ (vgl. Listing 3.1) erwartet hier als Eingabe eine MDP, zusammen mit der Zielmenge B. Es wird eine Menge R definiert, die am Ende alle Zustände enthalten soll, von denen aus man selbst mit der schlechtesten Wahl B irgendwie erreichen kann. Zu Beginn des Algorithmus gilt trivialerweise $R = B$. In der Iteration werden diese Menge nun mit allen Zuständen vereinigt, von denen aus

```
1  Eingabe: MDP M = (S, Act, P, q, L), Zielmenge B ⊆ S, mit s ⊨ φ ∀s ∈ B
2  Ausgabe: die Menge S^{min=0} = {s ∈ S | p_min(s, true U φ) = 0}
3  R := B;
4  do
5     R' := R;
6     R := R' ∪ {s ∈ S | ∀α ∈ Act(s) : (∃s' ∈ R' : P(s, α, s') > 0)};
7  while R ≠ R';
8  return S \ R;
```

Listing 3.1: Berechnung von $S^{\min=0}$

```
1  Eingabe: MDP M = (S, Act, P, q, L), Zielmenge B ⊆ S, mit s ⊨ φ ∀s ∈ B
2  Ausgabe: die Menge S^{max=0} = {s ∈ S | p_min(s, true U φ) = 0}
3  R := B;
4  do
5  R' := R;
6  R := R' ∪ {s ∈ S | ∃α ∈ Act(s) : (∃s' ∈ R' : P(s, α, s') > 0)};
7  while R ≠ R';
8  return S \ R;
```

Listing 3.2: Berechnung von $S^{\max=0}$

für alle Aktionen eine direkte Transition in die Menge R existiert. Dies wird über einen Zwischenspeicher R' realisiert. Solange sich die Menge R innerhalb der Iteration verändert hat, wird dieses Verfahren wiederholt. R enthält nun alle Zustände, die B, bei Betrachtung der minimalen Wahrscheinlichkeit, erreichen können. $\mathcal{S} \setminus B$ ist daher genau $S^{\min=0}$.
Der Algorithmus für $S^{\max=0}$ (vgl. Listing 3.2) verhält sich sehr ähnlich. In der Iteration werden zur Menge R jedoch nur diejenigen Zustände hinzugefügt, für die es **zumindest eine Aktion** gibt, die eine direkte Transition in die Menge R liefert. Bei der Betrachtung der maximalen Wahrscheinlichkeit für das Erreichen der Zielmenge würde hier immer diese Transition gewählt werden, weshalb so die gewünschte Menge $S^{\max=0}$ entsteht.

Mithilfe dieser Teilmengen können wir nun die *Bellman Equations* anwenden, um die gewünschten Wahrscheinlichkeiten zu berechnen:

3.3.2 Definition Bellman Equations [1], [5]
Gegeben sei ein endlicher MDP $\mathcal{M} = (\mathcal{S}, \text{Act}, \mathbb{P}, q, L)$, $s \in \mathcal{S}$ und $B \subseteq \mathcal{S}$ eine Teilmenge von Zuständen, für welche die Zustandsformel ϕ gilt.
Der Vektor $(x_s)_s \in \mathcal{S}$ mit $x_s = Pr^{\max}(s, \texttt{true}\ \mathsf{U}^{[0,t]}\ \phi)$ gibt die eindeutige Lösung der folgenden Gleichungen an. Diese werden auch Bellman Equations genannt.

$$x_s = \begin{cases} 1, & \text{für } s \in B \\ 0, & \text{für } s \notin B, s \in S^{\max=0} \\ \max\{\sum\limits_{s' \in \mathcal{S}} \mathbb{P}(s,a,s') \cdot x_{s'} \mid a \in \text{Act}(s)\}, & \text{sonst.} \end{cases}$$

Die Definition für die minimale Wahrscheinlichkeit eine Zustandsmenge zu erreichen ist analog.

Es gibt verschiedene Ansätze, diese Gleichungen zu lösen. Im Folgenden wollen wir näher auf die sogenannte *Value Iteration* eingehen, aber auch andere Verfahren kurz erläutern.

Value Iteration Eine mögliche Lösungsvorschrift approximiert in einem iterativen Verfahren den Lösungsvektor. Wir stellen hier die Definition der Value Iteration nach [5] vor.

3.3.3 Definition VALUE ITERATION [5]
Sei $\mathcal{S}$ die Zustandsmenge eines MDP und $B \subseteq \mathcal{S}$ eine Teilmenge von Zuständen, für welche die Zustandsformel ϕ gilt. Es gilt

$$p_{\min}(s, \texttt{true U}\ \phi) = \lim_{n\to\infty} x_s^{(n)},$$

wobei a eine atomare Eigenschaft ist. Dabei gilt für $x_s^{(n)}$:

$$x_s^{(n)} = \begin{cases} 1, & \text{für } s \in B \\ 0, & \text{für } s \notin B, s \in S^{\max=0} \\ 0, & \text{für } s \notin B, s \notin S^{\max=0}, n = 0 \\ \max\{\sum\limits_{s'\in\mathcal{S}} \mathbb{P}(s,a,s') \cdot x_{s'}^{(n-1)} \mid a \in \mathrm{Act}(s)\}, & \text{für } s \notin B, p_{\max}(s,\psi) > 0, n > 0 \end{cases}$$

Möchte man mithilfe der Value Iteration ein *unbounded* Until berechnen, so führt man das iterative Verfahren bis zum Eintreten einer Abbruchbedingung aus. Dies könnte beispielsweise $x_s^{(n)} - x_s^{(n-1)} \leqslant \varepsilon$ mit einem zuvor gewählten ε sein. Bezieht man sich jedoch auf *bounded* Until, so wird die Iteration genau für die entsprechende Anzahl von Zeitschritten ausgeführt. Es gilt dann $p_{\min}(s, \texttt{true U}^{[0,t]}\ \texttt{a}) = x_s^{(t)}$.
Berechnet man mit Value Iteration eine maximale oder minimale Wahrscheinlichkeit für zeitgebundene Erreichbarkeit, geht man demnach folgendermaßen vor: Der Vektor $x^{(0)}$ hat Einträge = 1 für die Zustände, die bereits in der Zielmenge sind. Für alle anderen Zustände sind die Einträge 0. Insbesondere weiß man, dass für Zustände s, die die Zielmenge nicht erreichen können gilt: $x_s^{(i)} = 0$ mit $i \in \mathbb{N}$. Dies sind die Zustände $s \in S^{\max=0}$ bzw. $S^{\min=0}$. Im nächsten Iterationsschritt wird für jeden Zustand ein neuer Wert berechnet. Dafür werden alle Aktionen dieses Zustandes betrachtet, jedoch nur die Wahrscheinlichkeit der jeweiligen Aktion, die den maximalen bzw. minimalen Wert liefert, wird gespeichert. Die entsprechende Wahrscheinlichkeit einer Aktion ist die Summe aus jeweils der Übergangswahrscheinlichkeit in einen anderen Zustand multipliziert mit der für diesen Zustand geltenenden Wahrscheinlichkeit aus dem letzten Iterationsschritt.

Gauss-Seidel Das Gauss-Seidel Verfahren ist eine Variation der Value Iteration. Dabei werden in jeder Iteration die aktuell berechneten Werte eines Zustandes verwendet. Im Gegensatz dazu verwendet die traditionelle Value Iteration immer die Ergebnisse des letzten Iterationsschrittes, auch, wenn bereits neuere Ergebnisse für einen einzelnen Zustand gibt. Demnach können alle berechneten Wahrscheinlichkeiten direkt in den Lösungsvektor geschrieben werden, es wird also nur Speicher für einen einzelnen Vektor benötigt. [5]

Policy Iteration Ein weiteres iteratives Verfahren zur Lösung, welches jedoch nicht über die Werte des Lösungsvektors, sondern über Scheduler iteriert, ist *Policy Iteration*. Dabei werden nur gedächtnislose Scheduler betrachtet. Zunächst wird ein beliebiger Scheduler für den MDP ausgewählt und für diesen der Lösungsvektor x berechnet. Dafür berechnet man die Wahrscheinlichkeiten auf der entsprechenden induzierten DTMC. In einem nächsten Schritt werden nacheinander alle Zustände betrachtet. In jedem Zustand überprüft man, ob es eine Aktion gibt, die hier eine höhere (bzw. niedrigere) Wahrscheinlichkeit induziert. Falls dies der Fall ist wird der Scheduler auf die Wahl dieser Aktion angepasst. Das Verfahren terminiert, wenn der Scheduler in jedem Zustand die beste bzw. schlechteste Wahl trifft. [5]

Granularität und Fehler durch Diskretisierung

Berechnet man die Wahrscheinlichkeit für eine zeitgebundene Formel auf einem diskretisierten Modell, so muss sich der Parameter für die Angabe der Grenzzeit ändern. Damit die Formel $\texttt{true}\ \mathsf{U}^{[0,\text{time}]}\ \phi$ auf dem MDP für die entsprechend korrekte Anzahl von Zeitschritten geprüft wird, müssen wir diese zunächst berechnen. Die Anzahl der Schritte steps, die eine Formel getestet werden muss, berechnet sich aus der zu testenden Zeit time und der gewählten Zeitschrittdauer τ. Es gilt:

$$\text{steps} = \frac{\text{time}}{\tau}.$$

Um kontinuierliche Zeit optimal zu simulieren, muss die Granularität der Diskretisierung, also τ, möglichst klein gewählt werden. Wählt man τ zu groß, so schränkt man die Anzahl der möglichen Schritte stark ein.

3.3.4 Definition Fehler der Diskretisierung [8]
Der Fehler ε, der durch die Wahl von τ bei der Diskretisierung entsteht, kann folgendermaßen berechnet werden:

$$\varepsilon = \frac{(E_{\max} \cdot \text{time})^2}{2 \cdot \text{steps}}.$$

Dabei ist $E_{\max} = \max_{s \in \mathcal{S}} E(s)$ die maximale Ausgangsrate der CTMDP und steps die Anzahl der Zeitschritte, die gemacht werden muss, um die gewünschte Zeit time zu testen.

Fazit In diesem Kapitel haben wir erarbeitet, wie man auf einem MDP Wahrscheinlichkeiten für die Erreichbarkeit einer Zustandsmenge berechnet. Zusammen mit den Grundlagen zu Abstraktion und Diskretisierung, die wir in Kapitel 2 erläutert haben, sind wir nun in der Lage aus einem unendlichen QBD ein diskretes Modell zu erlangen und auf Basis diesen Modells die Wahrscheinlichkeit von PCTL-Formeln überprüfen.

4. Anwendung

Neben stochastischen Modellen kennen wir nun auch Möglichkeiten, Wahrscheinlichkeiten für interessante Eigenschaften, die wir in PCTL formalisieren können, zu berechnen. In diesem Kapitel werden wir die MDP-Abstraktion und Diskretisierung auf ein realitätsnahes Beispiel anwenden. Dafür betrachten wir in 4.1 die Modellierung des *Transmission Control Protocols* als QBD. In 4.2 abstrahieren wir das vorgestellte Modell. Abschließend stellen wir in 4.3 die Diskretisierung mitsamt der erhaltenen Übergangsfunktion vor.

4.1. Modellierung des *Transmission Control Protocols*

In einer Modellierung des Kommunikationsprotokolls TCP wird das Verhalten des Connection Managements *on-demand connection with delayed release* (ODCR) analysiert. [11] Das System besteht aus einem Packet Generator, einer unendlichen Warteschlange und

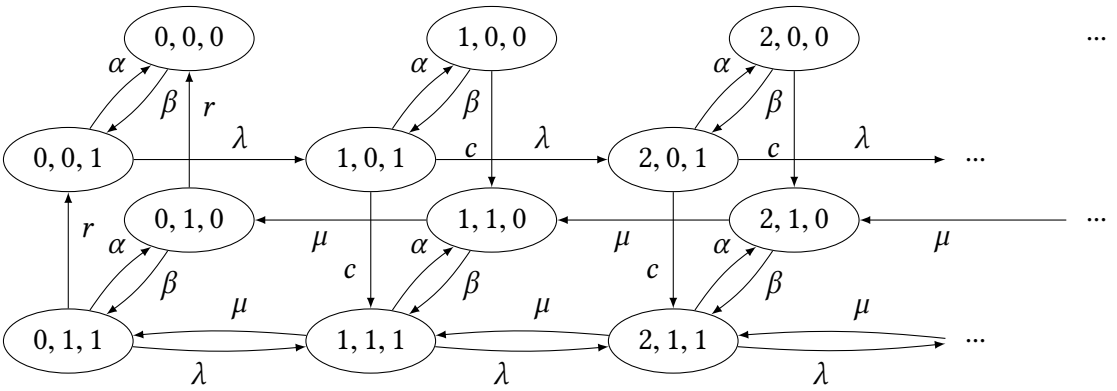


Abbildung 4.1.: Darstellung des TCP-Modells als CTMC nach [11].

einem Connenction Management. Der Zustandsraum ist $\mathcal{S} = \{(i, j, k) \mid i \in \mathbb{N}, j, k \in \{0, 1\}\}$. Hier steht i für die Anzahl der Elemente in der Warteschlange, $j = 0$ dafür, dass das Connection Management *released*, also inaktiv, und $j = 1$ dafür, dass es *active*, also aktiv, ist. Ist das Connection Management aktiv, so versendet es Pakete mit einer Rate von μ. Es wechselt außerdem mit Rate r von aktiv zu inaktiv, und entsprechend umgekehrt mit Rate c. In diesem Kontext steht k für den Zustand des Packet Generators: Für $k = 1$ werden Pakete mit der Rate λ erzeugt. Der Packet Generator wechselt mit Rate α von *burst* zu *off*. Vom ausgeschalteten Modus wechselt er mit der Rate β wieder in den Erzeugungsmodus.

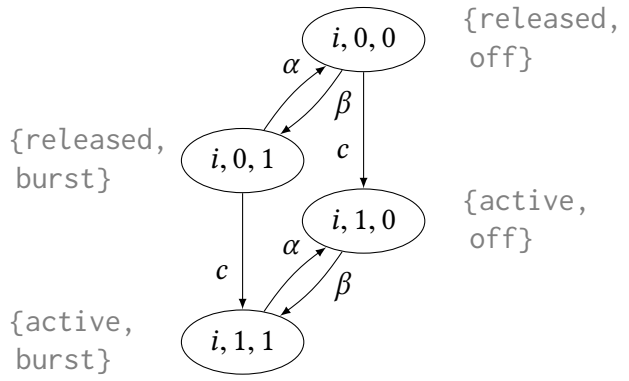


Abbildung 4.2.: Labels der CTMC $\mathcal{T}$ des TCP für alle Level $i \in \mathbb{N}_{\geqslant 0}$.

Das Protokoll ist hier als unendliche CTMC $\mathcal{T} = (\mathcal{S}, \mathcal{R}, q, L)$ modelliert, wobei $\mathcal{S}$ der oben genannte Zustandsraum ist. Jede Ebene stellt alle Zustände des Connection Managements und des Packet Generators für eine feste Anzahl $i \in \mathbb{N}$ an Warteschlangenelementen dar. Sei $(0, 0, 1)$ der Startzustand: In diesem Zustand ist der Packet Generator im Erzeugungsmodus, das Connection Management jedoch ist noch *released*, also ausgeschaltet. Mit der Labelfunktion L weisen wir den Zuständen aus $\mathcal{S}$ jeweils die passenden Eigenschaften aus $A = \{\texttt{burst}, \texttt{off}, \texttt{released}, \texttt{active}\}$ zu, wie in Abbildung 4.2 für die Wiederholenden Ebenen dargestellt. Die Labels verhalten sich im Grundlevel äquivalent.
Die Ratenfunktion $\mathcal{R} \colon S \times S \to \mathbb{R}$ der unendlichen CTMC hat aufgrund der QBD-Struktur die Form einer Blockmatrix. Es gilt

$$\mathcal{R} = \begin{pmatrix} \mathcal{R}_{0,0} & \mathcal{R}_{0,1} & & \\ \mathcal{R}_{1,0} & \mathcal{R}_{1,1} & \mathcal{R}_{1,2} & \\ & \ddots & \ddots & \ddots \end{pmatrix}.$$

Dabei sehen die Untermatrizen für $i \in \mathbb{N}_{\geqslant 0}, s, s' \in \mathcal{S}$ folgendermaßen aus:

$$\mathcal{R}_{0,0} = \begin{pmatrix} 0 & \beta & 0 & 0 \\ \alpha & 0 & 0 & 0 \\ r & 0 & 0 & \beta \\ 0 & r & \alpha & 0 \end{pmatrix}, \qquad \mathcal{R}_{i,i+1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda \end{pmatrix},$$

$$\mathcal{R}_{i,i-1} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & \mu \end{pmatrix}, \qquad \mathcal{R}_{i,i} = \begin{pmatrix} 0 & \beta & c & 0 \\ \alpha & 0 & 0 & c \\ 0 & 0 & 0 & \beta \\ 0 & 0 & \alpha & 0 \end{pmatrix}.$$

4.2. MDP-Abstraktion

Wir wenden auf dieses Modell die MDP-Abstraktion an. Dadurch erhalten wir einen endlichen QBD, in dem auf das Abstraktionslevel m ein abstraktes Level folgt. Der Zustandsraum des abstrakten Levels ist $S_a = \{(j,k) \mid j,k \in \{0,1\}\}$. Intuitiv sind die korrespondierenden Zustände zu einem Zustand (j,k) der abstrakten Ebene die Zustände $(i,j,k), i \in \{1,\dots,m\}$ der Wiederholenden Level.

Die Zustände des abstrakten Levels erhalten nun die verschiedenen Aktionen `default`, `close` und `far`. In Abbildung 4.3 ist eine Visualisierung der Abstraktion dargestellt. Aus Übersichtsgründen kennzeichnen wir nur die Aktionen `close` und `far` besonders, alle anderen Transitionen haben die Aktion `default`.

$(0,0)$: Da die Zustände $(i,0,0) \in S_i, i \in \{1,\dots,m\}$ keine Transitionen in ein vorheriges Level haben, erhält dieser Zustand die `default`-Aktion. Er hat die gleichen Transitionen wie die entsprechenden Zustände, sie führen in die korrespondierenden Zustände des abstrakten Levels.

$(0,1)$: Auch die zu diesem Zustand korrespondierenden Zustände der Ebenen 1 bis m haben keine Transitionen, die in ein niedrigeres Level führen. $(0,1)$ erhält demnach auch die Aktion `default`. Die α- und c-Transitionen führen in die entsprechenden abstrakten Zustände. Die λ-Transition der passenden Zustände aus anderen Leveln führt in ein höheres Level, für den abstrakten Zustand bedeutet das, dass diese Transition auch zum entsprechenden abstrakten Zustand führt.

$(1,0)$: Die diesem Zustand entsprechenden Zustände haben eine μ-Transition ins jeweilige vorherige Level. Dieser Zustand hat also die Aktionen `close` und `far` und modelliert damit verschiedene Erreichbarkeiten des Abstraktionslevels m. Die `close`-Aktion steht für diesen Zustand im $(m+1)$-ten Level, mit $(\texttt{close}, \mu)$ kann man also in den Zustand $(m,1,0)$ gelangen. Mit der `far`-Aktion ist diese Transition nicht mehr möglich. Die μ-Transition wird hier zu einem Selfloop. Mit beiden Aktionen gibt es die Transitionen, die in der Ebene bleiben: $(\texttt{close}, \alpha)$ und $(\texttt{far}, \alpha)$ führen zu $(1,1)$.

$(1,1)$: Dieser Zustand erhält genau wie $(1,0)$ die $(\texttt{close}, \mu)$-Transition ins Abstraktionslevel. Zusätzlich gibt es hier eine λ-Transition, die in den Wiederholenden Ebenen i ins $(i+1)$-te Level zeigt. Für `close` entsteht demnach ein Selfloop mit der Rate λ. Da für die Aktion `far` die μ- und λ-Transition auf das abstrakte Level umgeleitet werden müssen, entsteht ein Selfloop mit der Rate $\lambda + \mu$.

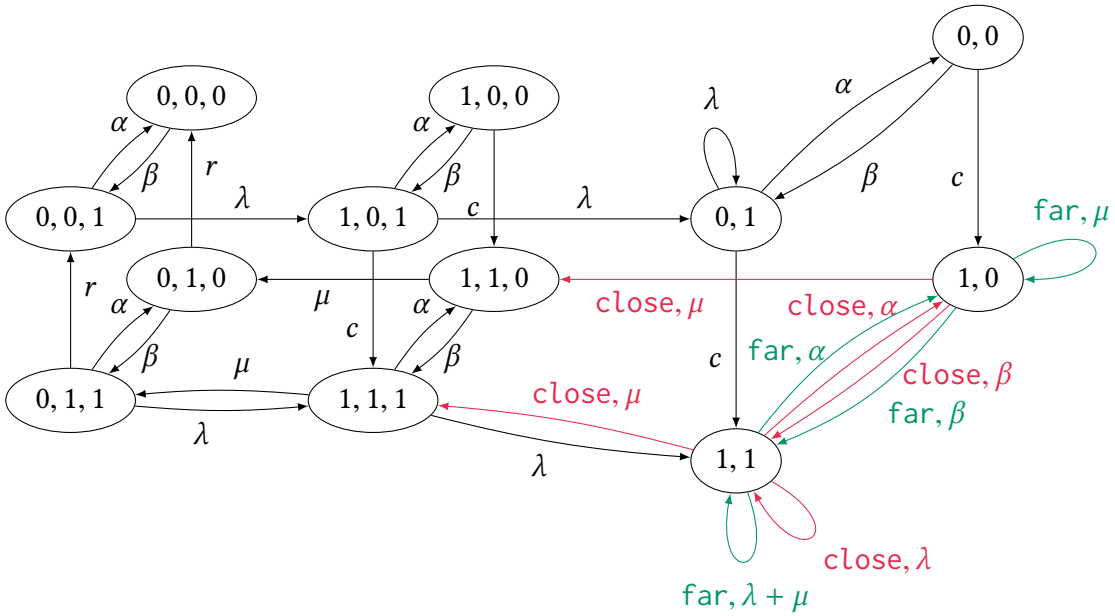


Abbildung 4.3.: TCP Modell als CTMDP mit Abstraktionslevel $m = 1$.

Ratenfunktion der CTMDP

Durch die Abstraktion ergibt sich für die Ratenfunktion $\mathcal{R} : \mathcal{S} \times \mathrm{Act} \times \mathcal{S} \to \mathbb{R}_{>0}$ der CTMPD bei einem Abstraktionslevel m folgende Form:

$$\mathcal{R} = \begin{pmatrix} \mathcal{R}_{0,0} & \mathcal{R}_{0,1} & & & \\ \mathcal{R}_{1,0} & \mathcal{R}_{1,1} & \mathcal{R}_{1,2} & & \\ & \ddots & \ddots & \ddots & \\ & & \mathcal{R}_{m,m-1} & \mathcal{R}_{m,m} & \mathcal{R}_{m,a} \\ & & & \mathcal{R}_{a,m} & \mathcal{R}_{a,a} \end{pmatrix}.$$

Diese Matrix ist offensichtlich dreidimensional, für Act ergeben sich drei verschiedene Ebenen, da gilt $\mathrm{Act} = \{\texttt{default}, \texttt{close}, \texttt{far}\}$. Die Untermatrizen $\mathcal{R}_{i,i+1}, \mathcal{R}_{i,i}$ und $\mathcal{R}_{i-1,i}$ für $i \in \{0, \ldots, m\}$, wie aus Definition 2.4 zu entnehmen, entsprechen in der `default`-Ebene denen des QBD aus 4.1. In den anderen Ebenen sind diese Matrizen 0. Die Matrizen $\mathcal{R}_{a,m}$ und

$\mathcal{R}_{a,a}$ haben Einträge in allen drei Ebenen:

$$\mathcal{R}_{a,m}^{\texttt{default}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \qquad \mathcal{R}_{a,a}^{\texttt{default}} = \begin{pmatrix} 0 & \beta & c & 0 \\ \alpha & 0 & 0 & c \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\mathcal{R}_{a,m}^{\texttt{close}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & \mu \end{pmatrix}, \qquad \mathcal{R}_{a,a}^{\texttt{close}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta \\ 0 & 0 & \alpha & \lambda \end{pmatrix},$$

$$\mathcal{R}_{a,m}^{\texttt{far}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \qquad \mathcal{R}_{a,a}^{\texttt{far}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mu & \beta \\ 0 & 0 & \alpha & \lambda + \mu \end{pmatrix}.$$

4.3. Diskretisierung

Den nun erhaltene CTMPD $\mathcal{M} = (\mathcal{S}, \mathrm{Act}, \mathcal{R}, q, A, L)$ wollen wir mit dem in Kapitel 2.3 vorgestellten Verfahren diskretisieren. Dafür erinnern wir uns an die Bedingung für Diskretisierung: Der CTMDP muss lokal einheitlich sein. In 2.2.1 haben wir definiert, was es heißt, locally uniform zu sein. Die Ausgangsrate E muss in allen Zuständen unabhängig von der Wahl der Aktion sein. Da wir bei der MDP-Abstraktion alle Transitionen erhalten indem wir sie, falls nötig, in das abstrakte Level umleiten, ist diese Eigenschaft natürlicherweise gegeben. Wir können also Diskretisierung anwenden und erhalten einen MDP $\mathcal{M}_\tau = (\mathcal{S}, \mathrm{Act}, \mathcal{R}, q, A, L)$. Die Übergangsfunktion $\mathbb{P}_\tau \colon \mathcal{S} \times \mathrm{Act} \times \mathcal{S} \to [0, 1]$ dieses MDPs hat die folgende Form:

$$\mathbb{P} = \begin{pmatrix} \mathbb{P}_{0,0} & \mathbb{P}_{0,1} & & & \\ \mathbb{P}_{1,0} & \mathbb{P}_{1,1} & \mathbb{P}_{1,2} & & \\ & \ddots & \ddots & \ddots & \\ & & \mathbb{P}_{m,m-1} & \mathbb{P}_{m,m} & \mathbb{P}_{m,a} \\ & & & \mathbb{P}_{a,m} & \mathbb{P}_{a,a} \end{pmatrix}.$$

Die Untermatrizen haben im Wesentlichen dieselben Transitionen wie die der Ratenfunktion $\mathcal{R}$ – in diskretisierter Form. Hinzu kommen bei der Diskretisierung Selfloops, die mit einer Wahrscheinlichkeit die Verweilzeit der Ratenfunktion modellieren. Die parametrisierten Werte der Untermatrizen ergeben sich aus der Definition der diskretisierten Übergangsfunktion (vgl. 2.3.2).

Für $i \in \{1, \dots, m\}$ gilt:

$$\mathbb{P}_{0,0}^{\texttt{default}} = \begin{pmatrix} e^{-\beta\cdot\tau} & 1-e^{-\beta\cdot\tau} & 0 & 0 \\ \left(1-e^{-(\alpha+\lambda)\cdot\tau}\right)\cdot\frac{\alpha}{\alpha+\lambda} & e^{-(\lambda+\alpha)\cdot\tau} & 0 & 0 \\ \left(1-e^{-(r+\beta)\cdot\tau}\right)\cdot\frac{r}{r+\beta} & 0 & e^{-(r+\beta)\cdot\tau} & \left(1-e^{-(\beta+r)\cdot\tau}\right)\cdot\frac{\beta}{\beta+r} \\ 0 & \left(1-e^{-(r+\alpha+\lambda)\cdot\tau}\right)\cdot\frac{r}{r+\alpha+\lambda} & \left(1-e^{-(\alpha+r+\lambda)\cdot\tau}\right)\cdot\frac{\alpha}{\alpha+r+\lambda} & e^{-(r+\lambda+\alpha)\cdot\tau} \end{pmatrix},$$

$$\mathbb{P}_{0,1}^{\texttt{default}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \left(1-e^{-(\lambda+\alpha)\cdot\tau}\right)\cdot\frac{\lambda}{\lambda+\alpha} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \left(1-e^{-(\lambda+\alpha+r)\cdot\tau}\right)\cdot\frac{\lambda}{\lambda+\alpha+r} \end{pmatrix},$$

$$\mathbb{P}_{i,i-1}^{\texttt{default}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \left(1-e^{-(\mu+\beta)\cdot\tau}\right)\cdot\frac{\mu}{\mu+\beta} & 0 \\ 0 & 0 & 0 & \left(1-e^{-(\mu+\lambda+\alpha)\cdot\tau}\right)\cdot\frac{\mu}{\mu+\lambda+\alpha} \end{pmatrix},$$

$$\mathbb{P}_{i,i}^{\texttt{default}} = \begin{pmatrix} e^{-(c+\beta)\cdot\tau} & \left(1-e^{-(\beta+c)\cdot\tau}\right)\cdot\frac{\beta}{\beta+c} & \left(1-e^{-(c+\beta)\cdot\tau}\right)\cdot\frac{c}{c+\beta} & 0 \\ \left(1-e^{-(\alpha+\lambda+c)\cdot\tau}\right)\cdot\frac{\alpha}{\alpha+\lambda+c} & e^{-(\alpha+\lambda+c)\cdot\tau} & 0 & \left(1-e^{-(c+\alpha+\lambda)\cdot\tau}\right)\cdot\frac{c}{c+\alpha+\lambda} \\ 0 & 0 & e^{-(\beta+\mu)\cdot\tau} & \left(1-e^{-(\beta+\mu)\cdot\tau}\right)\cdot\frac{\beta}{\beta+\mu} \\ 0 & 0 & \left(1-e^{-(\alpha+\lambda+\mu)\cdot\tau}\right)\cdot\frac{\alpha}{\alpha+\lambda+\mu} & e^{-(\lambda+\mu+\alpha)\cdot\tau} \end{pmatrix},$$

$$\mathbb{P}_{i,i+1}^{\texttt{default}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \left(1-e^{-(\lambda+\alpha+c)\cdot\tau}\right)\cdot\frac{\lambda}{\lambda+\alpha+c} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \left(1-e^{-(\lambda+\alpha+\mu)\cdot\tau}\right)\cdot\frac{\lambda}{\lambda+\alpha+\mu} \end{pmatrix}.$$

Die Matrizen $\mathbb{P}_{i,i-1}$,$\mathbb{P}_{i,i}$ und $\mathbb{P}_{i,i+1}$ sind in den Ebenen für die Aktionen `close` und `far` in allen Einträgen gleich 0. Für die Untermatrizen $\mathbb{P}_{a,m}$, $\mathbb{P}_{a,a}$ des abstrakten Levels gilt dies nicht: Sie haben Einträge in allen drei Ebenen. Diese sehen folgendermaßen aus:

$$\mathbb{P}_{a,m}^{\texttt{default}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\mathbb{P}_{a,a}^{\texttt{default}} = \begin{pmatrix} e^{-(c+\beta)\cdot\tau} & \left(1-e^{-(\beta+c)\cdot\tau}\right)\cdot\frac{\beta}{\beta+c} & \left(1-e^{-(c+\beta)\cdot\tau}\right)\cdot\frac{c}{c+\beta} & 0 \\ \left(1-e^{-(\alpha+c)\cdot\tau}\right)\cdot\frac{\alpha}{\alpha+c} & e^{-(c+\alpha)\cdot\tau} & 0 & \left(1-e^{-(c+\alpha)\cdot\tau}\right)\cdot\frac{c}{c+\alpha} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\mathbb{P}_{a,m}^{\texttt{close}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \left(1-e^{-(\mu+\beta)\cdot\tau}\right)\cdot\frac{\mu}{\mu+\beta} & 0 \\ 0 & 0 & 0 & \left(1-e^{-(\mu+\alpha+\lambda)\cdot\tau}\right)\cdot\frac{\mu}{\mu+\alpha+\lambda} \end{pmatrix},$$

$$\mathbb{P}_{a,a}^{\texttt{close}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & e^{-(\beta+\mu)\cdot\tau} & \left(1-e^{-(\beta+\mu)\cdot\tau}\right)\cdot\frac{\beta}{\beta+\mu} \\ 0 & 0 & \left(1-e^{-(\alpha+\lambda+\mu)\cdot\tau}\right)\cdot\frac{\alpha}{\alpha+\lambda+\mu} & \left(1-e^{-(\lambda+\alpha+\mu)\cdot\tau}\right)\cdot\frac{\lambda}{\lambda+\alpha+\mu}+e^{-(\alpha+\mu+\lambda)\cdot\tau} \end{pmatrix},$$

$$\mathbb{P}_{a,m}^{\texttt{far}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\mathbb{P}_{a,a}^{\texttt{far}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \left(1-e^{-(\mu+\beta)\cdot\tau}\right)\cdot\frac{\mu}{\mu+\beta}+e^{-(\beta+\mu)\cdot\tau} & \left(1-e^{-(\beta+\mu)\cdot\tau}\right)\cdot\frac{\beta}{\beta+\mu} \\ 0 & 0 & \left(1-e^{-(\alpha+\lambda+\mu)\cdot\tau}\right)\cdot\frac{\alpha}{\alpha+\lambda+\mu} & \left(1-e^{-(\lambda+\mu+\alpha)\cdot\tau}\right)\cdot\frac{\lambda+\mu}{\lambda+\mu+\alpha}+e^{-(\lambda+\mu+\alpha)\cdot\tau} \end{pmatrix}.$$

Fazit In diesem Kapitel haben wir eine Modellierung des Transmission Control Protocols betrachtet. Die Darstellung von TCP als unendlicher QBD haben wir aus [11] übernommen. Auf diesem QBD haben wir die in Kapitel 2 vorgestellten Ideen zum Vereinfachen eines Modells kombiniert und so aus dem QBD zunächst einen CTMDP und anschließend einen MDP erhalten. Mit den theoretischen Ausführungen aus Kapitel 3 wollen wir nun Berechnungen auf dem erhaltenen vereinfachten Modell durchführen. Dafür werden wir den PRISM Model Checker nutzen.

5. Tests und Analyse mit dem PRISM Model Checker

Wir werden im folgenden Kapitel PRISM verwenden, um *time-bounded reachability* auf dem abstrahierten und diskretisierten Modell des TCP zu testen und eine Aussage über die Qualität der berechneten Werte zu treffen.
Dafür führen wir in 5.1 PRISM ein und gehen darauf ein, wie Modelle und Eigenschaften in PRISM spezifiziert werden. Anschließend stellen wir in 5.2 die Implementierung des TCP-Modells vor und konkretisieren die Eigenschaft mitsamt der relevanten Parameter, für die wir Wahrscheinlichkeiten berechnen wollen.
Wenn wir alle Grundlagen geklärt haben, um PRISM für unser Problem verwenden zu können, werden wir dazu übergehen, in 5.3 Tests durchzuführen. Wir berechnen mithilfe des Model Checkers verschiedene Werte und wollen versuchen, diese Werte einzuordnen. In 5.4 werden wir noch auf verschiedene Lösungsmethoden von PRISM sowie ihre verschiedenen Laufzeiten eingehen.
Die Berechnungen werden wir ausschließlich auf einem Lenovo X1 Carbon mit Intel Core i7, 2.6 GHz und 8 GB RAM durchführen. Für die Berechnungen verwenden wir die in Anhang B aufgeführten und erläuterten bash-Skripte, die wir entwickelt haben, um strukturiertes Testen zu vereinfachen. Alle Konsolenausgaben sowie Resultate von PRISM finden sich auf der beigelegten CD.

5.1. PRISM Model Checker

Der PRISM Model Checker ist ein Tool zur automatischen Verifizierung von Systemen mit stochastischem Verhalten. PRISM kann für CTMCs, DTMCs und MDPs Wahrscheinlichkeiten zu verschiedenen Eigenschaften berechnen. Dafür übergibt man dem Tool eine Modellbeschreibung und eine Liste von Eigenschaften, die getestet werden sollen. Zum Berechnen der Wahrscheinlichkeiten gibt es eine Anzahl von Lösungsansätzen in PRISM, sowie intern verschiedene Möglichkeiten, die Modelle zu verarbeiten, um Berechnungen durchzuführen. Diese verschiedenen Ansätze werden *Engines* genannt. PRISM verwendet standardmäßig die *Hybrid Engine*, welche eine Kombination aus *symbolischem* und *explizitem* Model Checking realisiert. [9] Für MDPs sind die von PRISM verwendeten Lösungsverfahren *Value Iteration*, *Policy Iteration*, *Modified Policy Iteration* und *Gauss-Seidel*. Diese Verfahren haben wir bereits in 3.3 kennengelernt. Bei der Wahl eines Verfahrens muss man auch die Wahl der Engine in Betracht ziehen. Nur Value Iteration, welche standardmäßig für Berechnungen auf MDPs ausgewählt ist, arbeitet mit der Hybrid Engine. Deshalb

werden wir für unsere Tests auch Value Iteration auf der Hybrid Engine verwenden. Da die anderen Verfahren die Explicit Engine nutzen, werden wir für die Laufzeitanalyse Value Iteration auf der Hybrid Engine mit allen vier Verfahren auf der Explicit Engine vergleichen.

5.1.1. Modelle in PRISM

An einem Beispiel wollen wir die Eingabe von Modellen in PRISM verdeutlichen. PRISM verwendet dafür eine eigene Syntax, die in [12] ausführlich erläutert und dokumentiert wird. Wir stellen in einem kurzen Abschnitt die für uns relevanten Elemente dar. Eine beispielhafte Modellbeschreibung in PRISM einer einfachen MDP aus Abbildung 3.1 ist in Listing 5.1 abgebildet.

```
1  mdp
2
3  module beispiel
4  s: [0..3] init 0;
5  //Transitionen von a (s=0) aus
6  [alpha] s=0 -> 1/3: (s'=1) + 2/3: (s'=2);
7  [beta] s=0 -> 1/4: (s'=3) + 3/4: true;
8  //Transitionen von b (s=1) aus
9  [gamma] s=1 -> 4/5: (s'=3) + 1/5: true;
10 //Transitionen von c (s=2) aus
11 [tau] s=2 -> 1: (s'=3);
12 //Transitionen von d (s=3) aus
13 [delta] s=3 -> 1: true;
14 [sigma] s=3 -> 1: (s'=0);
15 endmodule
16
17 //labels
18 label "goal" = s=2;
```

Listing 5.1: Implementierung einer einfachen MDP (vgl. Abbildung 3.1) in PRISM

Wir führen kurz die für uns wichtigen Elemente der Syntax ein:

`mdp`	Hier geben wir den Typ des Modells an. In diesem Fall ist das Modell ein Markov Decision Process, kurz `mdp`.
`module beispiel`	Mit diesem Befehl teilt man PRISM mit, dass hier ein Modell beginnt. Es können mehrere Module eingegeben werden, die PRISM anschließend intern zu einem Modell verarbeitet. Nach dem Befehl wird der Name des Moduls angegeben.
`s: [0..3] init 0;`	Hier wird der Zustandsraum $\{0, 1, 2, 3\}$ spezifiziert. Die Zustände werden im Folgenden mit s aufgerufen. 0 ist der Startzustand, welcher mit `init` gekennzeichnet wird.. Durch das Verwenden von mehreren Variablen mit verschiedenen Eigenschaften kann man alle Kombinationen dieser Parameter in Zustände überführen.

`[alpha] s=0 -> 1/3: (s'=1) + 2/3: (s'=2);`	Mit diesem Befehl werden alle Transitionen eines Zustandes zu einer bestimmten Aktion beschrieben. Die Syntax ist hier: `[<action>] <current state> -> <probability>: (<goal state>) + <probability>: (<goal state>);`. Wir beschreiben hier also die Transitionen von a (s=0) aus, die unter der Aktion α verfügbar sind.
`[gamma] s=1 -> 4/5: (s'=3) + 1/5: true;`	Dieser Befehl beschreibt ebenfalls eine Transition. Verwendet man anstatt eines Zielzustandes den Parameter `true`, so bedeutet dies, dass sich nichts an der Position im Modell ändert: Diese Transition ist ein Selfloop.
`endmodule`	Dieser Befehl schließt die Eingabe für das Modul.
`label "goal" = s=2;`	Mit dem `label`-Befehl bezeichnet man eine Zuweisung durch die Labelfunktion. `label` erwartet zunächst eine Eigenschaft, gefolgt von einem Zustand. Wir weisen hier dem Zustand c (s=2) das Label *goal* zu. Die Syntax lautet: `label "<label>" = <state>;`.

5.1.2. PCTL-Formeln als Eigenschaften in PRISM

Für die Formalisierung von Eigenschaften liefert PRISM ebenfalls eine eigene Syntax. Für die exakte Berechnung von Wahrscheinlichkeiten p_{max} und p_{min} der zeitgebundenen Erreichbarkeit einer Zustandsformel wird der folgende Ausdruck verwendet:

```
Pmax=? [ true U[0,t] <state formula> ]
Pmin=? [ true U[0,t] <state formula> ]
```

Dabei müssen in der Zustandsformel verwendete Label in der Form `"label"` geschrieben werden. Mit der Eingabe `Pmax=?` berechnet PRISM die exakte maximale Wahrscheinlichkeit für das Eintreten der darauffolgenden Formel. `Pmin=?` steht analog für die minimale Wahrscheinlichkeit. Wollen wir beispielsweise die maximale Wahrscheinlichkeit für das Erreichen einer Menge von Zuständen, die mit dem Label `goal` versehen sind, innerhalb einer bestimmten Zeit $t = 5$ berechnen, so sieht die Eigenschaft folgendermaßen aus:

```
Pmax=? [ true U[0,5] "goal" ]
```

Es können alternativ auch Eigenschaften gegen einen konkreten Grenzwert getestet werden. Dies könnte dann die folgende Form haben:

```
P>0.9  [ true U[0,t] "goal" ]
P<0.35 [ true U[0,t] "goal" ]
```

Für `P>p` berechnet PRISM die minimale Wahrscheinlichkeit, und prüft, ob sie über dem Grenzwert p liegt. In diesem Fall ist die Ausgabe keine konkrete Wahrscheinlichkeit sondern der Boolean *true* bzw. *false*. Für `P<p` wird geprüft, ob die maximale Wahrscheinlichkeit unter dem Schwellwert p liegt. [5]

5.2. Realisierung des TCP-Modells in PRISM

In diesem Abschnitt wollen wir kurz den Quelltext des Modells sowie die Eigenschaften, für die wir Wahrscheinlichkeiten berechnen werden, vorstellen.

5.2.1. TCP als MDP-Modell

Der vollständige Text des Modellcodes ist in Anhang A.1 hinterlegt. Wir werden im Folgenden Auszüge daraus präsentieren und kurz erläutern.
In Listing 5.2 sind die Voreinstellungen des Modells konkretisiert. Wir definieren den Typ durch `mdp`. Anschließend wird die Konstante e definiert, sowie die intern verwendete Formel `ex` für eine bessere Übersichtlichkeit. Es werden die konstanten Parameter des TCP-Modells mit Werten belegt, wie sie in [11] auch verwendet wurden. Außerdem wird mit `const int steps = floor(time / tau);` die Anzahl von Zeitschritten berechnet, die abhängig von den gegebenen Parametern τ und time überprüft werden muss. `floor`, also Abrunden, ist hier nötig, damit der Wert in einen Integer geschrieben werden kann. Die Ergebnisse werden dadurch aber nicht verfälscht: Wenn der Quotient nicht ohnehin eine ganze Zahl ist, bedeutet das, dass ein geringer Abschnitt der zu prüfenden Zeit übrig bleibt, da dieser Abschnitt kleiner ist als die Dauer unseres Zeitschrittes. In diesem Fall kann tatsächlich kein weiterer Zeitschritt gemacht werden, durch Abrunden erhalten wir also genau die gewünschte Anzahl an Schritten.
Hier wird auch der Eingabeparameter `maxlevel` definiert, welcher das Abstraktionslevel des QBD bezeichnet.

```
1  mdp
2
3  const double e = 2.71828182845904523536028747135266249775724709369995957496 6;
4  const double tau;
5  const double time;
6  formula ex = pow(e,-tau);
7  const int steps = floor(time / tau);
8
9  const double alpha = 1;
10 const double beta = 0.04;
11 const double r = 10;
12 const double c = 10;
13 const int maxlevel;
14 const int a = maxlevel+1;
15 const double lambda = 100;
16 const double mu = 125;
```

Listing 5.2: PRISM Quellcode des TCP-Modells mit der Definition von Konstanten, Eingabeparametern und dem Typ des Modells.

In Listing 5.3 beginnt die Modulbeschreibung, welche das eigentliche Modell definiert. Hier wird der modellierte Zustandsraum beschrieben, also die Zustände bestehen aus der Parametermenge `i,j,k`, die für die Anzahl der Elemente in der Warteschlange `(i)`, den Zustand des Connection Managements `(j)` und den Zustand des Packet Generators `(k)` stehen. Der Startzustand ist (0, 0, 1), also `i=0,j=0,k=1`. Mit `a` wird das letzte zu erzeugende Level angegeben: Das `abstrakte Level` des CTMDP.

```
18 module tcp
19 i: [0..a] init 0; // Warteschlange
20 j: [0..1] init 0; // Connection Management
21 k: [0..1] init 1; // Packet Generator
```

Listing 5.3: PRISM Quellcode des TCP-Modells mit Modulbeginn und Zustandsraum.

In Listing 5.4 beschreiben wir die Transitionen der Zustände im Grundlevel. Diese fallen ausnahmslos unter die `default`-Aktion. Im PRISM-Code verwenden wir für diese Aktion ein leeres Label: `[]`. Hier ist für alle vier Zustände der Grundebene beschrieben, wann ein Übergang stattfindet. Dabei gilt:

$$\texttt{(1-pow(ex,alpha+lambda))*alpha/(alpha+lambda)} = \left(1 - e^{-(\alpha+\lambda)\cdot\tau}\right) \cdot \frac{\alpha}{\alpha+\lambda}.$$

Die Transition `(pow(ex,beta)): true;` beschreibt durch das `true` einen Selfloop, dieser modelliert die Verweilzeit in einem Zustand, wie in 2.3 formalisiert wurde.

```
24 // Grundlevel
25 [] i=0 & j=0 & k=0
26 -> (1-pow(ex,beta)) : (k'=1) // von off zu burst
27 +  (pow(ex,beta)): true;
28
29 [] i=0 & j=0 & k=1
30 -> (1-pow(ex,alpha+lambda))*alpha/(alpha+lambda): (k'=0) // von burst zu off
31 +  (1-pow(ex,alpha+lambda))*(lambda/(alpha+lambda)): (i'=1) // ins naechste Level
32 +  (pow(ex,alpha+lambda)): true;
33
34 [] i=0 & j=1 & k=0
35 -> (1-pow(ex,beta+r))*(beta/(beta+r)): (k'=1) // von off zu burst
36 +  (1-pow(ex,beta+r))*(r/(beta+r)): (j'=0) // von active zu released
37 +  pow(ex,beta+r): true;
38
39 [] i=0 & j=1 & k=1
40 -> (1-pow(ex,alpha+r+lambda))*alpha/(alpha+r+lambda): (k'=0) // von burst zu off
41 +  (1-pow(ex,alpha+lambda+r))*(r/(alpha+lambda+r)): (j'=0) // von active zu released
42 +  (1-pow(ex,alpha+lambda+r))*(lambda/(alpha+lambda+r)): (i'=1) // ins naechste Level
43 +  pow(ex,alpha+r+lambda): true;
```

Listing 5.4: PRISM Quellcode des TCP-Modells mit Transitionen des Grundlevels.

Die Transitionen der Wiederholenden Level und die `default`-Transitionen des abstrakten Levels verhalten sich nahezu identisch und können im Anhang (vgl. A.1) eingesehen werden. In Listing 5.5 sind die Übergänge der Aktion `close` aufgeführt. Hier ist zu Beginn der Definition die entsprechende Aktion vermerkt. Die `far`-Transitionen verhalten sich analog.

```
81 // close-Transitionen
82 [close] i=a & j=1 & k=0
83 -> (1-pow(ex,beta+mu))*(beta/(beta+mu)): (k'=1) // von off zu burst
84 +  (1-pow(ex,beta+mu))*(mu/(beta+mu)): (i'=maxlevel) // ins "vorherige" Level (maxlevel)
85 +  pow(ex,beta+mu): true;
86
87 [close] i=a & j=1 & k=1
88 -> (1-pow(ex,lambda+alpha+mu))*(alpha/(lambda+alpha+mu)): (k'=0) // von  burst zu off
89 +  (1-pow(ex,lambda+alpha+mu))*(mu/(lambda+alpha+mu)): (i'=maxlevel) // ins "vorherige"
       Level (maxlevel)
90 +  (1-pow(ex,lambda+alpha+mu))*(lambda/(lambda+alpha+mu)): true // im abstrakten Level
       bleiben
91 +  pow(ex,lambda+alpha+mu): true;
```

Listing 5.5: PRISM Quellcode des TCP-Modells mit `close`-Transitionen des abstrakten Levels.

In Listing 5.6 wird das Ende der Modellbeschreibung dargelegt. Hier wird das Modul beendet. Anschließend werden noch die Labels spezifiziert, wie wir sie in Abbildung 4.2 für unser Modell eingeführt haben.

```
104 endmodule
105
106 // labels
107 label "burst" = k=1;
108 label "off" = k=0;
109 label "active" = j=1;
110 label "released" = j=0;
```

Listing 5.6: PRISM Quellcode des TCP-Modells mit Modulende und Labeldefinition.

5.2.2. Zeitgebundene Erreichbarkeit von (released ∧ ¬burst)

Die zu testende Eigenschaft entnehmen wir grundlegend [11], modifizieren sie jedoch für einen durch Abstraktion und Diskretisierung entstandenen MDP. Daraus erhalten wir die zwei folgenden Eigenschaften:

```
Pmax=? [ true U[0,steps] "released & !burst" ]
Pmin=? [ true U[0,steps] "released & !burst" ]
```

Hier ist steps nicht die Zeit, wie sie in [11] zum Testen der Formel angegeben ist, sondern die Anzahl von Schritten, für die die PCTL-Formel überprüft wird. In Abschnitt 5.2.3 werden wir weiter auf die Berechnung von steps eingehen.

Die Zustandsformel `"released & !burst"` stellt eine interessante Eigenschaft für unser Modell dar. Dem liegt die Bedeutung zu Grunde, dass in einem Zustand, in dem diese Formel gilt, das Connection Management ausgeschaltet (`released`) und der Packet Generator nicht im Erzeugungsmodus (`no burst`) ist. Naheliegenderweise scheinen dies ressourcenschonende Zustände unseres Modells zu sein. Dies betrifft alle Zustände $(i, 0, 0)$ für $i \in \mathbb{N}$. Diese Zustände sind auf zwei verschiedenen Wegen erreichbar:

- Man ist in einem beliebigen Level i in einem $(i, 0, 1)$-Zustand. Von dort aus kann der Zustand $(i, 0, 0)$ des selben Levels mithilfe der mit α parametrisierten Transition erreicht werden.

- Man befindet sich im Grundlevel in Zustand $(0, 1, 0)$. Hier gibt es eine mit r parametrisierte Transition in den Zustand $(0, 0, 0)$.

Betrachtet man die Struktur unseres QBD in Abbildung 4.3, so erkennt man außerdem, dass es nur im Grundlevel eine mit r parametrisierte Transition gibt. Inhaltlich bedeutet dies, dass das Connection Management in den anderen Leveln nicht ausgeschaltet werden kann. Nur wenn die Warteschlange abgearbeitet ist, kann das Connection Management wieder auf `released` springen. Um unsere Formel zu erfüllen, müssen im Modell also entweder alle Pakete versandt worden sein, oder das Connection Management darf nie eingeschaltet worden sein. In beiden Situationen muss anschließend der passende Übergang innerhalb des Levels geschehen, damit ein Zielzustand erreicht wird.
Wir werden im nächsten Abschnitt darauf eingehen, welche Auswirkungen dies auf die Wahrscheinlichkeiten haben wird.

5.2.3. Raten sowie Parameter τ, time und *maxlevel* des Modells

In diesem Abschnitt spezifizieren wir die Parameter, wie wir sie für unsere Berechnungen nutzen werden. Dies betrifft die Parameter der Wahrscheinlichkeitsfunktion unseres Modells sowie die Parameter, die für das Prüfen einer Formel nötig sind.

Raten Die Wahl der Parameter der Wahrscheinlichkeitsfunktion, die wir aus [11] übernommen haben, sind in Tabelle 5.2 aufgelistet.

Parameter	α	β	r	c	λ	μ
Rate	1	0.04	10	10	100	125

Tabelle 5.2.: Parameter der Raten- bzw. Wahrscheinlichkeitsfunktion des Modells, entnommen aus [11].

Die Werte dieser Parameter beeinflussen naheliegenderweise die Interpretation des Modells. Wir wollen anhand einiger Vergleiche kurz auf die Wirkung dieser Werte eingehen. Wir verwenden dafür der Einfachheit halber die Notation p_α für die Übergangswahrscheinlichkeit aus einem Zustand s in einen mit der Aktion α erreichbaren Zustand.
Da λ und μ sehr groß sind, kann man schlussfolgern, dass die Wahrscheinlichkeit für einen **Ebenenwechsel** sehr hoch ist. Die Wahrscheinlichkeit in ein niedrigeres Level zu wechseln ist etwas größer als die Wahrscheinlichkeit in ein höheres zu wechseln. Im

Vergleich: Ist die Wahrscheinlichkeit für eine Transition mit der ursprünglichen Rate μ eines Zustandes s beispielsweise $p_\mu \approx 0.1118$ (Wert für Zustand $(i, 1, 1)$, $\tau = 0.001$), so ist die Wahrscheinlichkeit für die λ-Transition in diesem Zustand das 0.8-fache dessen, also $p_\lambda \approx 0.0895$ und für die entsprechende α-Transition nur $1/100$: $p_\alpha \approx 0.00089$. Die Wahrscheinlichkeiten hingegen, zwischen den Zuständen des Packet Generators und des Connection Managements zu wechseln, sind deutlich geringer.
Die Parameter r und c beeinflussen die Übergänge des **Connection Managements**. Der Wechsel von `released` zu `active` unterscheidet sich hier im Wert nur gering von der Rückrichtung, da $r = c$ gilt. Die unterschiedliche tatsächliche Wahrscheinlichkeit, eine solche Transition zu wählen, wird durch die Gesamtausgangsrate des Zustandes bedingt. Man kann jedoch in Abbildung 4.3 erkennen, dass die r-Transitionen nur im Grundlevel bestehen. Das Connection Management kann also nur in diesem Level ausgeschaltet (`released`) werden.
Die Wahrscheinlichkeiten für diejenigen Transitionen, die zwischen den Zuständen `burst` und `off` des **Packet Generators** wechseln, sind mit $\alpha = 1$ und $\beta = 0.04$ äußerst gering. Da diese Raten relativ zueinander betrachtet werden müssen, erkennt man, dass ein β-Übergang im Vergleich zu einem c-Übergang deutlich unwahrscheinlicher ist, als ein c-Übergang im Vergleich zu einem λ-Übergang.
Dies ist interessant, wenn man betrachten möchte, wie sich die Wahrscheinlichkeiten eines $(i, 0, 1)$-Zustandes im Vergleich zu einem $(i, 0, 0)$-Zustand verhalten. Für die Zustände $(i, 0, 0)$ und $(i, 0, 1)$ mit $i \in \{1, \ldots, m\}$ gilt dann:

$$\text{In } (i,0,1): \qquad \frac{p_c}{p_\lambda} = \frac{\left(1 - e^{-(c+\lambda+\alpha)\cdot\tau}\right) \cdot \frac{c}{c+\lambda+\alpha}}{\left(1 - e^{-(\lambda+\alpha+c)\cdot\tau}\right) \cdot \frac{\lambda}{\lambda+\alpha+c}} = \frac{c/(\alpha+\lambda+c)}{\lambda/(\alpha+\lambda+c)} = \frac{c}{\lambda} = \frac{1}{10}.$$

$$\text{In } (i,0,0): \qquad \frac{p_\beta}{p_c} = \frac{\left(1 - e^{-(\beta+c)\cdot\tau}\right) \cdot \frac{\beta}{\beta+c}}{\left(1 - e^{-(c+\beta)\cdot\tau}\right) \cdot \frac{c}{c+\beta}} = \frac{\beta/(\beta+c)}{c/(\beta+c)} = \frac{\beta}{c} = \frac{1}{250}.$$

Vereinfachend kann man also sagen, dass für den Zustand $(i, 0, 1)$ die λ-Transition in ein nächstes Level zehnmal so wahrscheinlich ist, wie die c-Transition zum Einschalten des Connection Managements. Im Zustand $(i, 0, 0)$ jedoch ist es 250-mal so wahrscheinlich, dass sich das Connection Management einschaltet, als dass der Packet Generator in den `off`-Zustand wechselt. Dies hat zur Folge, dass von einem $(i, 0, 0)$-Zustand aus der Zustand $(0, 0, 0)$ mit einer hohen Wahrscheinlichkeit erreicht wird. Wir erinnern uns, dass für alle $(i, 0, 0)$-Zustände die Eigenschaft `"released & !burst"` gilt. Von diesen Zuständen aus ist es also sehr einfach, den $(0, 0, 0)$-Zustand zu erreichen, der intuitiv der am stärksten erwünschte Zustand eines Warteschlangenmodells ist: alle Pakete sind hier abgearbeitet und weder der Packet Generator, noch das Connection Management verbrauchen Ressourcen. Dies könnte eine Erklärung dafür sein, weshalb für eine Anwendung in der Realität die Wahrscheinlichkeit der Eigenschaft, die wir berechnen werden, relevant ist.

Test-Parameter Da wir im nächsten Kapitel Wahrscheinlichkeiten für verschiedene Instanzen der Parameter τ, time und *maxlevel* berechnen werden, wollen wir diese Parameter hier erneut erläutern.

maxlevel	Der Parameter *maxlevel* beschreibt die Anzahl der bei der Abstraktion beibehaltenen Level des ursprünglichen QBDs an. Die Abstraktionsebene ist demnach die *maxlevel*-te Ebene. Erst in der (*maxlevel*+1)-ten Ebene befinden wir uns im abstrakten Level. Um möglichst exakte Ergebnisse zu erhalten sollte dieser Parameter groß gewählt werden, da das ursprüngliche Modell unendlich groß war. Wir interessieren uns dafür, ob die Abstraktion auch für ein niedrigeres *maxlevel* ausreichend gute Ergebnisse liefert.
time	Der Parameter time gibt die Zeit an, in der eine bestimmte Zustandsformel erreicht werden soll.
τ	Der Parameter τ bezeichnet die Granulariät der Diskretisierung. Er beschreibt also, wie lang, relativ zur Zeiteinheit von time, ein diskreter Zeitschritt in unserem Modell ist. Pro Zeitschritt kann nur eine Transition innerhalb des Modells durchgeführt werden. Erst nach Ablauf von τ kann ein neuer Zustandsübergang stattfinden. Um kontinuierliche Zeit zu simulieren muss dieser Parameter möglichst klein gewählt werden.

Die entsprechende Formel aus [11] wird unter anderem für die Zeit time $\in \{1, 2\}$ getestet. Um Wahrscheinlichkeiten für die gleiche Grenzzeit zu berechnen, müssen wir einen neuen Parameter berechnen: steps steht für die Anzahl diskreter Zeitschritte, innerhalb derer die Zustandsformel `"released & !burst"` erfüllt werden soll. Bei einer festen Zeit time und einer gewählten Granularität τ ist die Anzahl der Zeitschritte der Quotient: steps $= {}^{\text{time}}/_{\tau}$. Hierbei fällt auf, dass ein zu großes τ die Anzahl der Schritte, die innerhalb des Modells gemacht werden, stark einschränkt. Betrachtet man beispielsweise eine Abstraktion nach der 10. Ebene, berechnet Werte für time $= 1$ und wählt $\tau = 0.1$, so können nur zehn Schritte innerhalb des Modells gemacht werden. Um das abstrakte Level zu erreichen, werden jedoch mindestens elf Schritte benötigt. Auch in Bezug auf die MDP-Abstraktion ist es also relevant, τ entsprechend zu wählen. In den folgenden Berechnungen werden wir diese Beobachtung berücksichtigen.

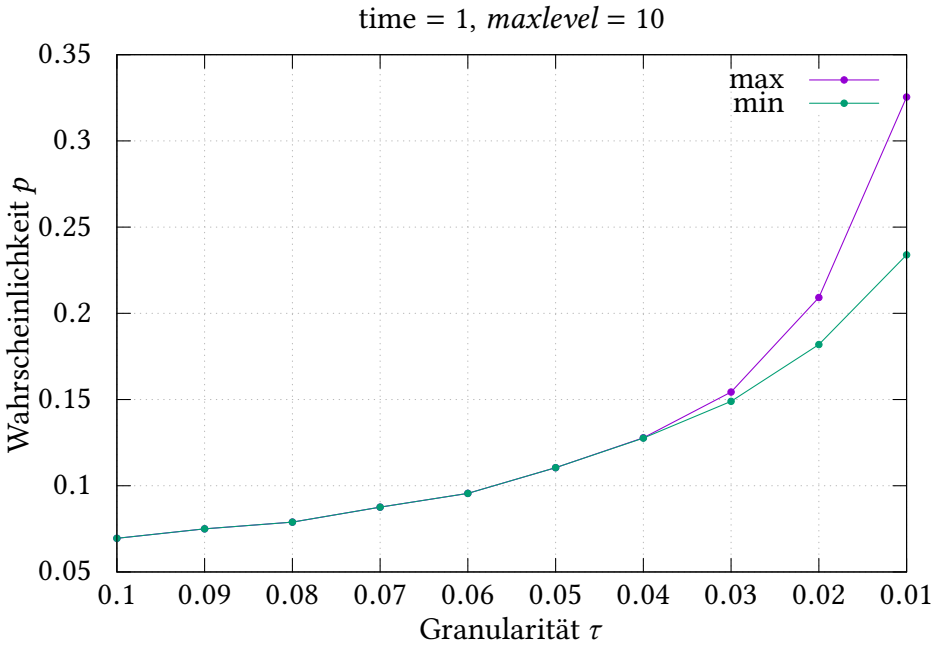


Abbildung 5.1.: Wahrscheinlichkeit p_{max} und p_{min} für ein kleiner werdendes τ, also eine feinere Granularität, bei einer Abstraktion nach 10 Leveln.

5.3. Resultate

Im folgenden Unterkapitel werden wir die unterschiedlichen Auswirkungen der Wahl von τ und *maxlevel* auf die Wahrscheinlichkeiten untersuchen. Dafür werden wir p_{min} und p_{max} für eine Reihe von verschiedenen Werten von τ und *maxlevel* testen; zunächst für einen festen Wert time = 1.
Intuitiv erwarten wir für ein feineres τ genauere Werte der berechneten Wahrscheinlichkeiten, da τ gewissermaßen die zeitliche Auflösung unseres Modells ist. Die Anzahl der Schritte, also Zustandsübergänge, die wir machen können, hängt demnach von τ ab. Ein zu groß gewähltes τ schränkt die Anzahl der möglichen Schritte stark ein und könnte durchaus Ergebnisse produzieren, die signifikant von der tatsächlichen Wahrscheinlichkeit abweichen. Diese Tendenz ist bereits in Abbildung 5.1 zu erkennen. Hier wurden Werte für τ zwischen 0.1 und 0.01 getestet. Während für den größten Wert die errechnete Wahrscheinlichkeit unter 0.1 liegt, ergeben sich für $\tau = 0.01$ maximale und minimale Wahrscheinlichkeiten um das Dreifache. Später werden wir genauer untersuchen, für welche τ wir möglichst korrekte Wahrscheinlichkeitswerte erhalten.
Variieren wir die Anzahl der beibehaltenen Level, also *maxlevel*, so erhalten wir für eine höhere Anzahl vermutlich exaktere Werte, da wir so näher an das unendliche Modell herankommen. Außerdem kann, wie bereits erwähnt, je nach Wahl von τ und time, die abstrakte Ebene gar nicht oder nur sehr unwahrscheinlich erreicht werden, wenn *maxlevel* groß genug ist. Wählen wir *maxlevel* jedoch kleiner, so erhalten wir ein kleineres Modell,

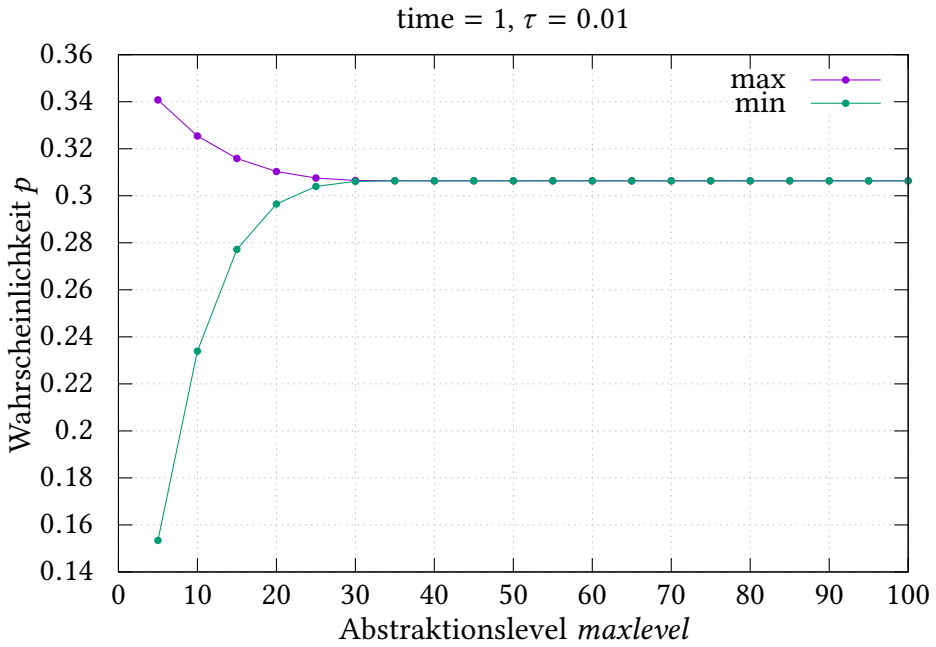


Abbildung 5.2.: Wahrscheinlichkeit p_{max} und p_{min} für $maxlevel = 0$ bis $maxlevel = 100$ bei $\tau = 0.01$.

was auch Vorteile haben kann, zum Beispiel für die Laufzeit der Tests. Uns interessiert daher, wie exakte Werte man durch MDP-Abstraktion bereits bei einer niedrigeren Levelanzahl erhalten kann. Das unterschiedliche Verhalten der maximalen und minimalen Wahrscheinlichkeiten ist in Abbildung 5.2 zu erkennen. Für ein niedriges Abstraktionslevel gibt es eine hohe Varianz zwischen maximaler und minimaler Wahrscheinlichkeit.
Wir wollen im Folgenden herausfinden, wie sich diese Werte für verschiedene Kombinationen der Granularität und dem maximalen Level verhalten.

Entwicklung der Wahrscheinlichkeiten bezüglich dem Abstraktionslevel *maxlevel*

In diesem Abschnitt betrachten wir den Einfluss von der Wahl des maximalen bei der MDP-Abstraktion beibehaltenen Levels auf die Wahrscheinlichkeiten. Dafür berechnen wir p_{max} und p_{min} mit Werten für *maxlevel* zwischen 0 und 100 und festen Werten für τ.
In Abbildung 5.3 sind diese Ergebnisse für verschiedene τ vergleichend in einem Koordinatensystem dargestellt. Für $\tau = 0.1$ liegt die Wahrscheinlichkeit konstant bei ≈ 0.069, die maximale und minimale Wahrscheinlichkeit unterscheiden sich nicht. Mit diesem τ beträgt bei einer Zeit von 1 die Anzahl möglicher Schritte 10. Naheliegenderweise ist es mit einer so geringen Schrittzahl nicht möglich, ein weiter entferntes abstraktes Level zu erreichen. Auch die zu erreichende Zustandsmenge ist so nur sehr erschwert zu erreichen.
Für $\tau = 0.01$ unterscheiden sich die maximale und die minimale Wahrscheinlichkeit bis

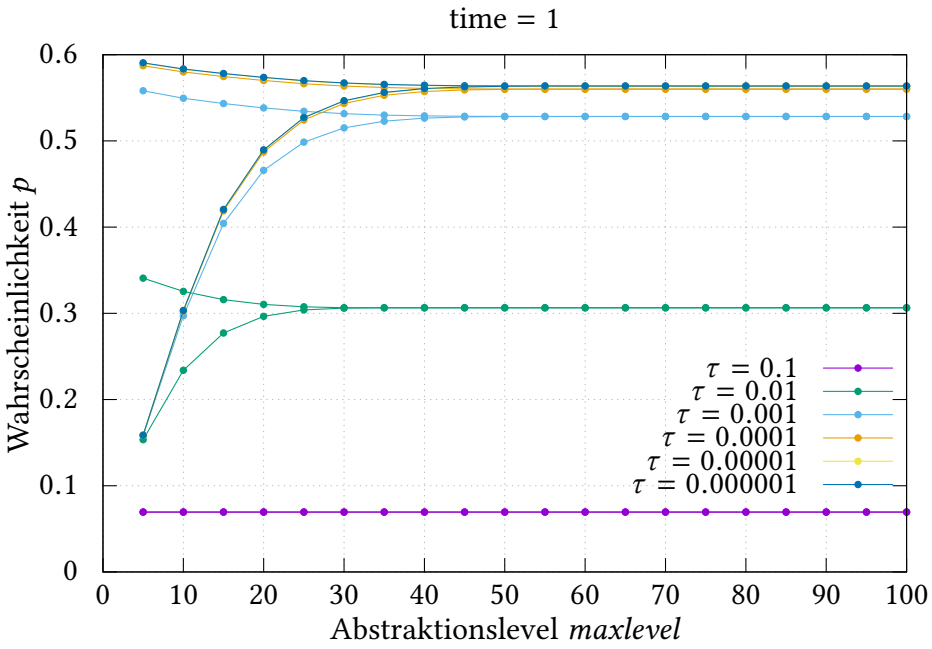


Abbildung 5.3.: Maximale und minimale Wahrscheinlichkeit für die Level 5 bis 100 für verschiedene Werte von τ.

zu einem Abstraktionslevel von 30. Ab hier beträgt die Wahrscheinlichkeit für p_{min} und $p_{max} \approx 0.3063$. Für $\tau = 0.001$ erhalten wir eine deutlich höhere Wahrscheinlichkeit von ≈ 0.5283. Die Wahrscheinlichkeitsspanne zwischen p_{min} und p_{max} ist hier für niedrige *maxlevel* sehr groß, sie reicht von 0.16 bis 0.56 bei einem Abstraktionslevel von 5. Erst ab einem Abstraktionslevel von 55 erkennt man keinen Unterschied mehr zwischen den beiden Extremen.
Für die Werte $\tau = 0.0001, \tau = 0.00001$ und $\tau = 0.000001$ kann man nur noch einen sehr geringen Unterschied erkennen. Die Wahrscheinlichkeitsspanne ist auch hier sehr hoch, für *maxlevel* $= 5$ und $\tau = 10^{-6}$ gilt $p_{min} \approx 0.1586$ und $p_{max} \approx 0.5904$. Die Spanne zieht sich bis zu einem Abstraktionslevel von 60. Dort liegt die Wahrscheinlichkeit bei ≈ 0.5636.
Betrachtet man den Verlauf dieser Werte im Vergleich, so erkennt man, dass für größere τ ein geringerer Unterschied zwischen der maximalen und der minimalen Wahrscheinlichkeit liegt. Auch der Punkt, ab dem sich die beiden Wahrscheinlichkeiten nicht mehr unterscheiden, liegt bei einem deutlich niedrigeren Abstraktionslevel als bei kleinen Werten für τ. Wir vermuten den Ursprung dessen darin, dass τ indirekt proportional zur möglichen Schrittanzahl innerhalb des Modells ist.
Im Vergleich ist außerdem zu erkennen, dass sich die konvergierende Wahrscheinlichkeit für verschiedene τ stark unterscheidet, woraus wir schließen, wie relevant die Wahl von τ für das Erhalten korrekter Ergebnisse ist.

τ	0.1	0.01	0.001	0.0001	0.00001	0.000001	0.0000001
ε	> 1	> 1	> 1	> 1	0.50625	0.050625	0.0050625

Tabelle 5.4.: Fehlerbetrachtung der Diskretisierung für verschiedene τ bei time = 1.

Betrachtung von Wahrscheinlichkeit und Fehler bezüglich τ

Wir wollen nun auswerten, wie genau sich der Einfluss von τ auf die Wahrscheinlichkeit auswirkt und mithilfe der Fehlerabschätzung der Diskretisierung berechnen, ab welchen Werten für τ wir sicher sein können, dass die Abweichung von der tatsächlichen Wahrscheinlichkeit unter einem gewissen Signifikanzniveau liegt.
Mithilfe der Formel aus Kapitel 3.3 können wir berechnen, wie groß der Fehler bei der Wahl von τ ist und anhand dessen entscheiden, welche τ klein genug sind, um sinnvolle Ergebnisse zu erhalten. In Tabelle 5.4 wird klar, dass erst ab $\tau = 10^{-6}$ von exakten Ergebnissen gesprochen werden kann. Diese Ergebnisse weichen maximal um $\varepsilon = 0.050625$ von den Werten ab, die nach der MDP-Abstraktion für unser Modell gelten. Wir wollen also herausfinden, wie sich für verschiedene feste Werte von *maxlevel* der Verlauf von τ auf die Wahrscheinlichkeiten auswirkt. Dafür wählen wir als maximale Levelanzahl Werte, bei denen die Abstraktion noch einen Einfluss auf das Ergebnis hat. Die nächsten Berechnungen führen wir demnach für $maxlevel \in \{10, 20, 30, 40\}$ aus.
Da wir in Abbildung 5.1 erkennen konnten, dass die Wahrscheinlichkeiten für ein kleiner werdendes τ ansteigen, wählen wir nun signifikant kleinere τ.
Für diese Berechnungen ändern wir daher die Skala der Granularität auf eine logarithmische, um das Verhalten bei einem exponentiell kleiner werdenden τ betrachten zu können. Dadurch werden wir einen deutlich höhere Genauigkeit für kleinere τ erhalten und können sehen, ob die Wahrscheinlichkeiten für sehr kleine τ-Werte noch große Veränderungen aufzeigen. In Abbildung 5.4 sind p_{max} und p_{min} bis $\tau = 0.0001$ dargestellt. Hier kann man sehen, dass die Wahrscheinlichkeiten zu konvergieren scheinen. Approximativ ergeben sich $p_{min} \approx 0.3$ und $p_{max} \approx 0.58$.
In Abbildung 5.5 kann man die Ergebnisse für ein logarithmisch kleiner werdendes τ für die Abstraktionslevel 20, 30 und 40 sehen. Erneut ist gut erkennbar, dass die Spanne zwischen der maximalen und minimalen Wahrscheinlichkeit für größer werdende Abstraktionslevel schrumpft. Außerdem steige in allen drei Betrachtungen p_{min} und p_{max} bis zu $\tau = 0.001$ sehr stark an. Zwischen 0.001 und 0.0001 verändern sich die Werte bei allen vier Tests nur noch um maximal 0.04, für noch kleiner werdende τ liegt die Veränderung in einem Bereich von 10^{-3}. Diese Berechnungen veranschaulichen, dass, entgegen unserer Erwartungen, die Ergebnisse auch für $\tau = 0.0001$ nicht stark von der tatsächlichen Wahrscheinlichkeit abweichen (vgl. Tabelle **??**). Wir haben berechnet, dass bereits für $\tau = 10^{-6}$ der maximale Fehler ε bei 0.050625 liegt. Da die Varianz zwischen $\tau = 10^{-6}$ und $\tau = 10^{-4}$ in allen Berechnungen im Bereich von 10^{-3} liegt, können wir festhalten, dass der maximale Fehler für $\tau = 0.0001$ nur bei 0.0150625 liegt. Für ein so großes τ übersteigt dies, aufgrund der ursprünglichen Fehlerabschätzung, unsere Erwartungen. Wir können also mit einer deutlich geringeren Granularität arbeiten, was die Laufzeiten vermutlich erheblich verbessert.

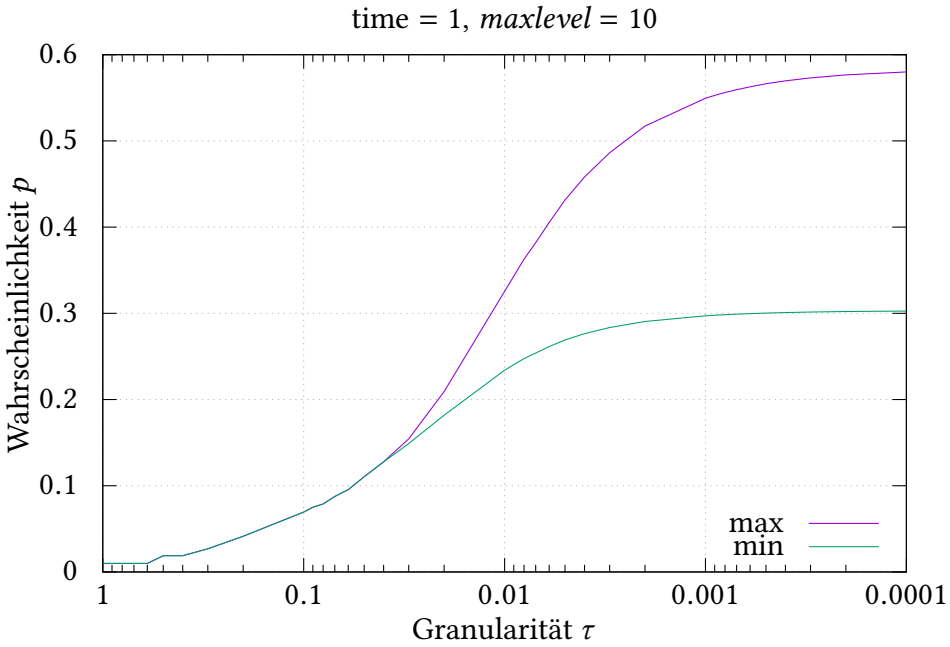


Abbildung 5.4.: Wahrscheinlichkeit p_{max} und p_{min} für logarithmisch schrumpfendes τ bis 0.0001 bei einer Abstraktion nach 10 Leveln.

maxlevel	p_{min}	p_{max}
10	≈ 0.30	≈ 0.56
20	≈ 0.49	≈ 0.57
30	≈ 0.54	≈ 0.56
40	≈ 0.56	≈ 0.58

(a) $\tau = 0.0001$

maxlevel	p_{min}	p_{max}
10	≈ 0.30	≈ 0.58
20	≈ 0.49	≈ 0.57
30	≈ 0.55	≈ 0.57
40	≈ 0.56	≈ 0.56

(b) $\tau = 0.0000001$

Tabelle 5.5.: Approximierte Wahrscheinlichkeiten p_{min}, p_{max} für *maxlevel* = 10 bis *maxlevel* = 40.

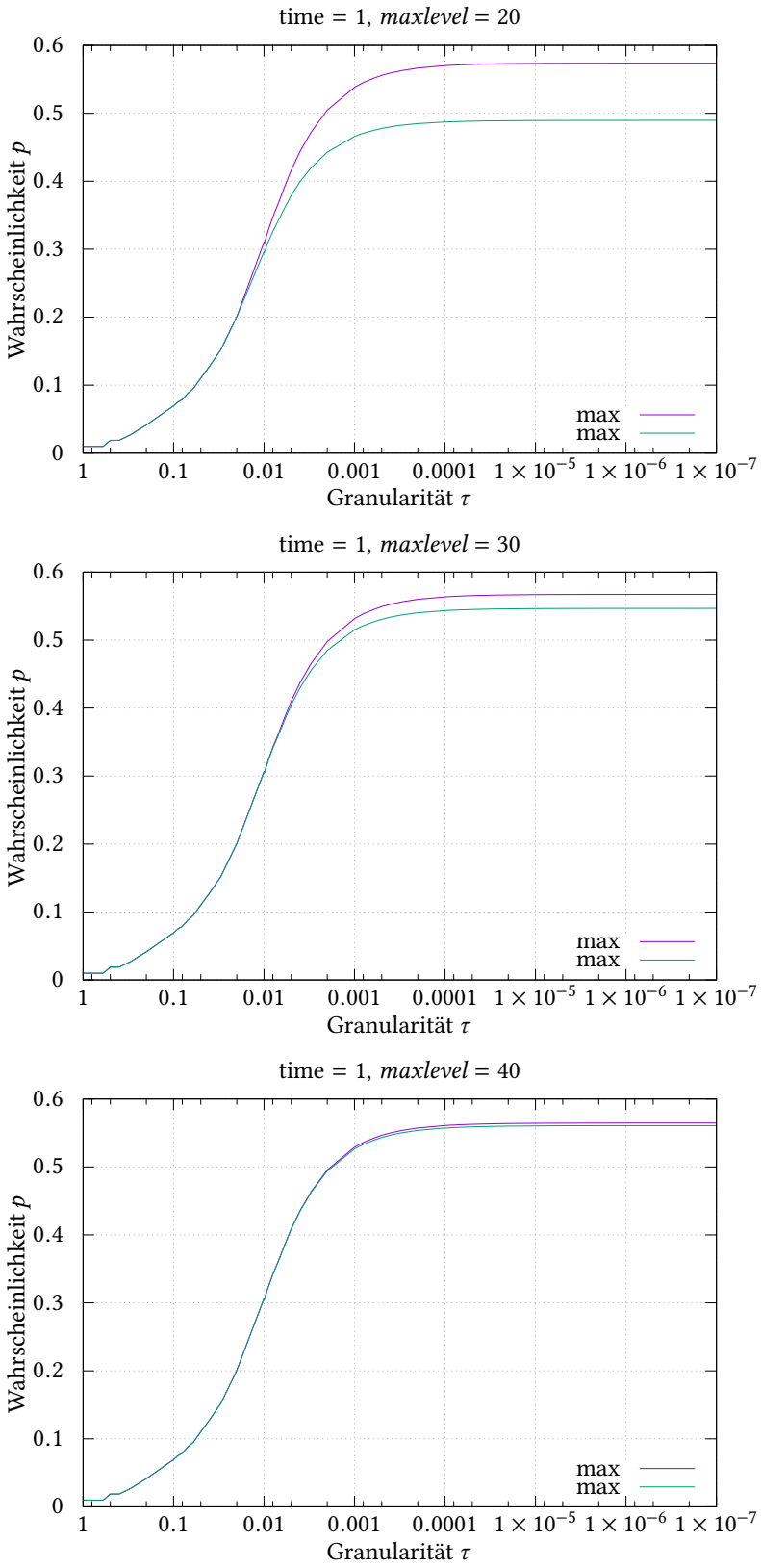


Abbildung 5.5.: Wahrscheinlichkeit p_{max} und p_{min} für logarithmisch schrumpfendes τ bis 0.0000001 bei einer Abstraktion nach 20, 30 und 40 Leveln.

Betrachtung für time = 2

Die verschiedenen Tests mit Variation von *maxlevel* und τ führen wir nun nocheinmal für time = 2 aus. Wir testen also die Formeln `Pmax=? [true U`$^{[0,2/\tau]}$ `(released ∧ ¬burst)]` und `Pmin=? [true U`$^{[0,2/\tau]}$ `(released ∧ ¬burst)]`. Wir wollen versuchen zu erkennen, ob die Schlüsse, die wir bezüglich der Variation der Parameter *maxlevel* und τ für time = 1 geschlossen haben, so auch auf eine andere Grenzzeit übertragbar sind.

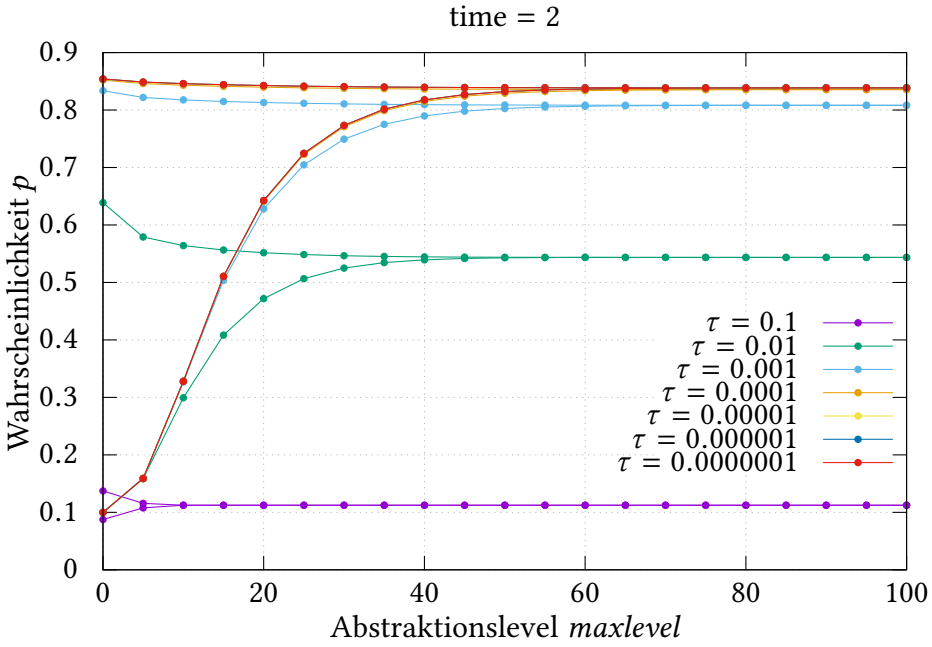


Abbildung 5.6.: Wahrscheinlichkeiten p_{max} und p_{min} für *maxlevel* zwischen 0 und 100 und τ von 0.1 bis 0.0000001.

Variation von *maxlevel* Zunächst betrachten wir wieder für fest gewählte τ zwischen 0.1 und 0.0000001 den Verlauf der Wahrscheinlichkeiten bezüglich *maxlevel*. Diese sind in Abbildung 5.6 dargestellt. Wir beschreiben kurz den Verlauf für jedes τ einzeln.

$\tau = 0.1$: p_{min} und p_{max} unterscheiden sich für dieses τ zu Beginn - allerdings nur bis zu einem Abstraktionslevel von 10 und nur zu einem sehr geringen Wert von $p_{max} - p_{min} \approx 0.05$. Ab dort ist die Wahrscheinlichkeit konstant ≈ 0.112.

$\tau = 0.01$: Hier ist die Differenz zwischen der maximalen und minimalen Wahrscheinlichkeit bereits viel ausgeprägter. Die Werte unterscheiden sich bis zu $maxlevel = 40$ und der Unterschied beträgt für ein kleines *maxlevel* bereits 0.53. Die Wahrscheinlichkeit konvergiert zu 0.543.

$\tau = 0.001$: Bei diesem τ ist die Spanne nun deutlich größer. Die Wahrscheinlichkeiten unterscheiden sich für $maxlevel \leqslant 10$ um über 0.733. Ab $maxlevel = 75$ ist die Wahrscheinlichkeit $p_{max} \approx p_{min} \approx 0.808$.

$\tau = 0.0001$: Hier ist das Level, ab dem sich p_{min} und p_{max} nicht mehr unterscheiden, wie für alle weiteren getesteten τ bei 70. Der Unterschied von p_{min} und p_{max} beträgt für $maxlevel = 0$ etwa 0.75. Die Wahrscheinlichkeit für diese τ konvergiert zu 0.835.

Im Vergleich zu time = 1 ist der Verlauf insgesamt sehr ähnlich. Das Level, ab dem sich die Wahrscheinlichkeiten nicht mehr unterscheiden, ist allerdings für jedes τ etwas größer als bei den vorherigen Berechnungen. Eine Übersicht diesbezüglich kann in Tabelle 5.7 eingesehen werden. Diese Abhängigkeit zeigt klar auf, dass in die Wahl von *maxlevel* auch die zu prüfende Zeit einbezogen werden muss. Es ist nicht möglich, allgemeingültig einen ausreichenden Wert für *maxlevel* zu finden, da die Qualität der Abstraktion deutlich von der entsprechenden Formel abhängt.

τ	$time = 1$	$time = 2$
0.1	5	10
0.01	30	55
0.001	50	75
0.0001	60	70
0.00001	50	80
0.000001	50	75
0.0000001	50	70

Tabelle 5.7.: Vergleich von time = 1 und time = 2 für das Level, ab dem $p_{min} \approxeq p_{max}$ gilt für $\tau = 0.1$ bis $\tau = 0.0000001$.

Man erkennt auch, dass sich die Wahrscheinlichkeiten ab $\tau = 0.0001$ kaum verändern. Dies entspricht den Beobachtungen der vorherigen Tests. Wir betrachten im nächsten Abschnitt die Wahl von τ noch einmal genauer, um diese Aussage zu verifizieren.

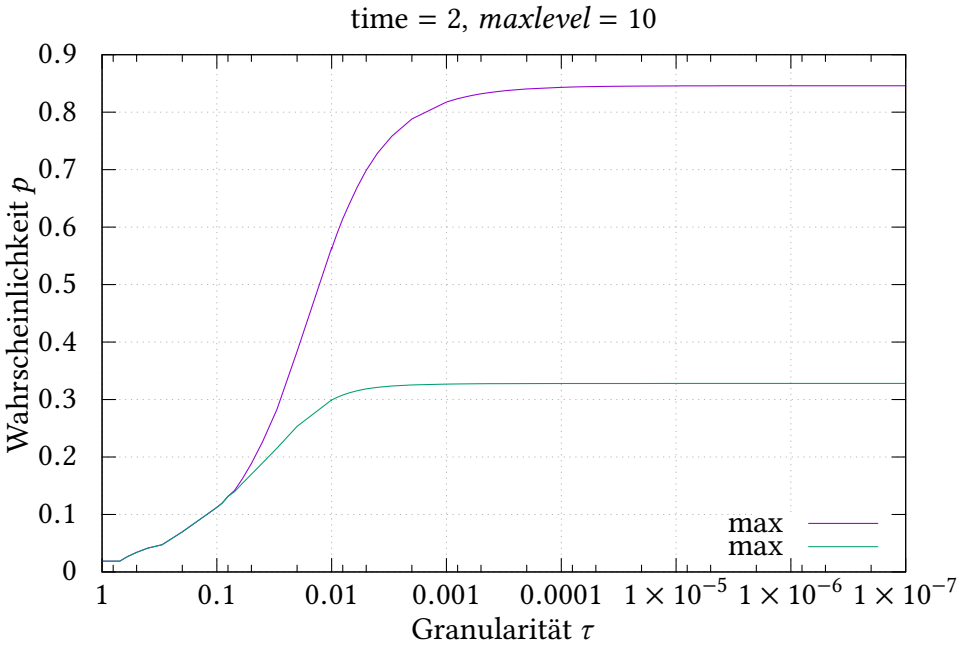


Abbildung 5.7.: Verlauf der Wahrscheinlichkeiten p_{min} und p_{max} für logarithmisch kleiner werdende τ bis $\tau = 10^{-7}$ und $maxlevel = 10$.

Betrachtung für signifikant kleiner werdende τ Als nächstes berechnen wir auch für time $= 2$ den Verlauf der Wahrscheinlichkeiten für feste Werte $maxlevel \in \{10, 20, 30, 40\}$ und signifikant kleiner werdende τ. Die Resultate dieser Tests sind in den Abbildungen 5.7, 5.8, 5.9 und 5.10 dargestellt.
Für $maxlevel = 10$ steigen die Werte zunächst an, entwickeln dann ab $\tau = 0.08$ einen Unterschied zwischen p_{min} und p_{max} und pendeln sich ab $\tau = 0.0001$ auf $p_{max} \approx 0.82$ und $p_{min} \approx 0.33$ ein.
Das Verhalten für die anderen *maxlevel* ist ähnlich, wie wir allerdings in Abbildung 5.6 bereits erkennen konnten, ist die Varianz zwischen p_{min} und p_{max} für höhere *maxlevel* deutlich geringer.
Bei $maxlevel = 20$ (vlg. Abbildung 5.8) unterscheiden sich die Wahrscheinlichkeiten ab $\tau = 0.04$. Sie steigen beide weiterhin stark an und entwickeln nur einen Unterschied von 0.2: $p_{min} \approx 0.64, p_{max} \approx 0.84$.
Für $maxlevel = 30$ (vlg. Abbildung 5.9) ist der erreichte Unterschied geringer als 0.1. p_{max} konvergiert zu 0.84, p_{min} zu 0.77. Ab $\tau = 0.0001$ kann man diese Werte erkennen.
Bei einem Abstraktionslevel von 40 erkennt man erst bei $\tau = 0.01$ einen sichtbaren Unterschied zwischen den beiden Werten. p_{min} erreicht hier 0.817, p_{max} 0.839. Ab $\tau = 0.0001$ erkennt man keine Änderung mehr im Verlauf. Diese Ergebnisse sind in Abbildung 5.10 dargestellt.

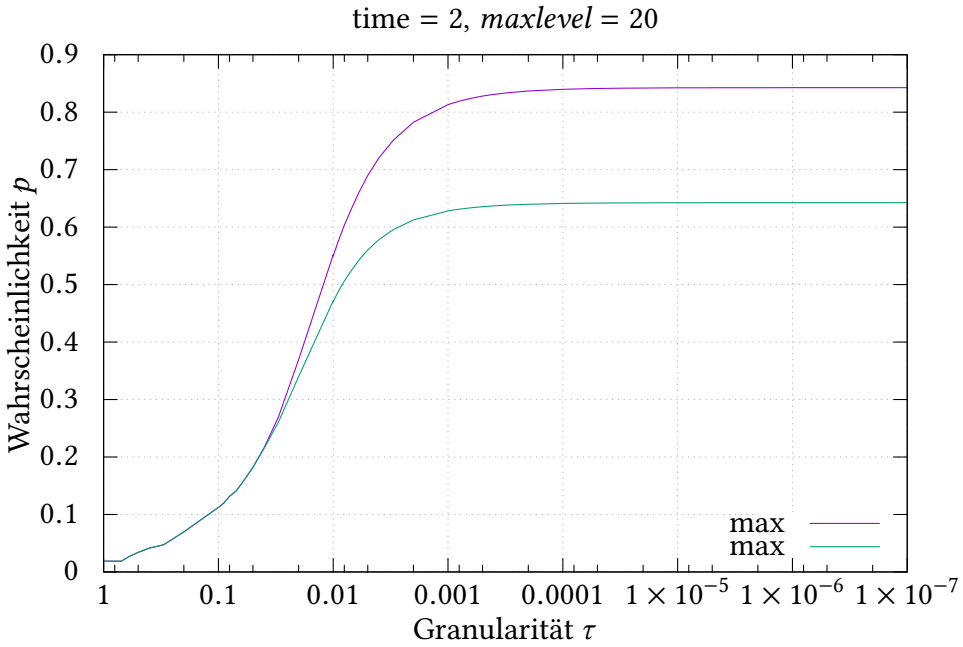


Abbildung 5.8.: Verlauf der Wahrscheinlichkeiten p_{min} und p_{max} für logarithmisch kleiner werdende τ bis $\tau = 10^{-7}$ und $maxlevel = 20$.

Wir können erneut deutlich erkennen, dass für $\tau = 0.0001$ bereits aussagekräftige Werte berechnet werden können. Dies vermittelt den Eindruck, dass der Wert von *time* keine Bedeutung für die Wahl von τ hat. Da τ die Granularität unserer diskreten Zeit darstellt, scheint diese Unabhängigkeit plausibel. Um sie zu statistisch zu belegen müssten weitere Berechnungen durchgeführt werden.

Fazit In den Berechnungen der Erreichbarkeitsformel mit Zeitbeschränkung time = 2 konnten wir erkennen, dass der Wert von time unbedingt für die Wahl des Abstraktionslevels berücksichtigt werden muss. Für eine größere Zeit erreichen die Wahrscheinlichkeiten p_{max} und p_{min} erst später eine Einigung und die Differenz der beiden Werte ist für kleine *maxlevel* deutlich größer. Außerdem haben wir eine vermeintliche Unabhängigkeit zwischen der Zeit und τ erkannt. Diese könnte in einer weiteren Studie untersucht werden. Dafür sollte eine deutlich größere Varianz von Werten für time betrachtet werden, insbesondere Zeitwerte die um einige Größenordnungen größer sind, als die von uns analysierten, könnten aussagekräftige Ergebnisse liefern.

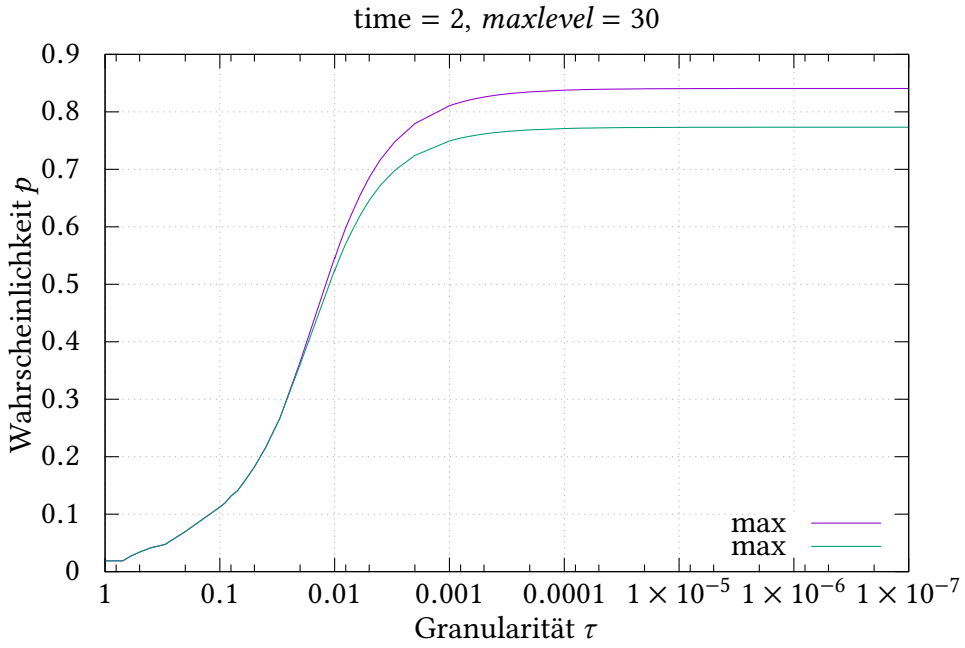


Abbildung 5.9.: Verlauf der Wahrscheinlichkeiten p_{min} und p_{max} für logarithmisch kleiner werdende τ bis $\tau = 10^{-7}$ und $maxlevel = 30$.

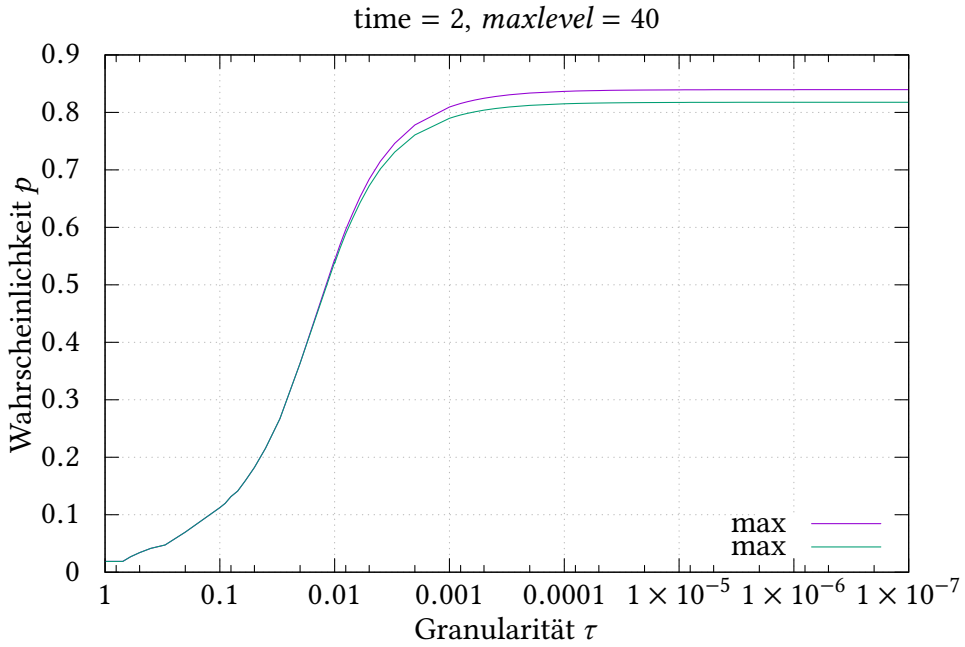


Abbildung 5.10.: Verlauf der Wahrscheinlichkeiten p_{min} und p_{max} für logarithmisch kleiner werdende τ bis $\tau = 10^{-7}$ und $maxlevel = 40$.

5.4. Laufzeitanalyse

In diesem Abschnitt wollen wir untersuchen, ob und inwiefern die verschiedenen Lösungsverfahren unterschiedliche Laufzeiten benötigen. Dafür gehen wir erneut kurz auf die verschiedenen Engines und Lösungsverfahren ein, bevor für zwei verschiedene Werte für τ die Laufzeiten der Verfahren bei der Berechnung von Wahrscheinlichkeiten für maximale Level zwischen 0 und 100 vergleichen.

Engines: [9]	
Hybrid Engine	Die Hybride Engine verwendet eine Kombination von expliziten und symbolischen Datenstrukturen. Für die Berechnungen wird eine Kombination der Methoden der Sparse und der MTBDD Engine genutzt. Diese Engine kann mit gut großen Zustandsmodellen umgehen ist für gewöhnlich am performantesten.
Sparse Engine	Die Sparse Engine kombiniert auch explizite und symbolische Datenstrukturen. Sie nutzt sogenannte dünnbesetzte Matrizen für die Berechnungen. Diese Engine kann im Vergleich zur Hybrid Engine nur kleinere Modelle verarbeiten, für diese aber unter Umständen schneller sein.
MTBDD Engine	Diese Engine verwendet sogenannte *multi-terminal binary decision diagrams* (MTBDD). Auch sie verwendet eine explizite und symbolische Datenstrukturen. Zur Berechnung verwendet sie nur Binäre Entscheidungsdiagramme (BDDs) und MTBDDs. Die Performanz dieses Verfahrens ist nicht einheitlich. Für Modelle mit einer hohen Regelmäßigkeit in der Struktur *kann* diese Engine gute Ergebnisse liefern.
Explicit Engine	Diese Engine verwendet keine symbolischen Datenstrukturen. Für kleinere Modelle kann diese Engine performant sein.

Lösungsverfahren:	
Value Iteration	Die Methode der Value Iteration iteriert über den Lösungsvektor des Gleichungssystems. In jedem Iterationsschritt werden die Wahrscheinlichkeiten eines Zustandes aktualisiert, indem die Übergangswahrscheinlichkeiten zu anderen Zuständen mit den Werten dieser Zustände aus dem letzten Iterationsschritt multipliziert werden. Hier wird immer das Maximum bzw. Minimum der Ergebnisse für verschiedene Aktionen gewählt. [5]
Gauss-Seidel	Dieses Verfahren ist eine Optimierung von Value Iteration: für das Berechnen neuer Wahrscheinlichkeiten werden die aktuellsten gespeicherten Werte jeden Zustandes genutzt. Es wird dadurch auch weniger Speicher benötigt. [5]
Policy Iteration	Diese Methode iteriert über die verschiedenen Scheduler. Es wird zunächst ein beliebiger gedächtnisloser Scheduler ausgewählt. In jedem Iterationssschritt wird verglichen, ob es einen besseren Scheduler geben kann. [5]
Modified Policy Iteration	Die Methode Modified Policy Iteration ist eine Variante der Policy Iteration. Die allgemeine Policy Iteration ist naheliegenderweise ein recht umfangreiches Verfahren, da es sehr viele Scheduler geben kann. Die Modified Policy Iteration reduziert die Zahl der zu testenden Scheduler und erzielt dadurch eine Reduktion in der Komplexität. [3]

Die folgenden vier Tests zeigen je die Laufzeit der vier verschieden numerischen Methoden auf. Da Gauss-Seidel, Policy Iteration und Modified Policy Iteration die Explicit Engine nutzen, Value Iteration auf der Hybrid Engine jedoch der Standard ist, haben wir für Value Iteration beide Engines getestet.
In den Tests berechnen wir p_{min} und p_{max} der Formel $\texttt{true}\ \mathsf{U}^{[0,\text{time}]}(\texttt{released} \wedge \neg\texttt{burst})$ für time = 1. Als Granularität wählen wir $\tau \in \{0.000001, 0.0000001\}$. Dies sind Werte für τ, für die mit Sicherheit feststeht, dass der Fehler sehr gering ist (vlg. Tabelle 5.4). Obwohl wir in späteren Tests erkannt haben, dass auch eine weniger feine Granularität akzeptable Ergebnisse liefert, werden wir die Laufzeitanalyse mit diesen sehr kleinen Werten für τ durchführen. So erwarten wir, Unterschiede in der Laufzeit deutlicher hervorzuheben. Die Ergebnisse für $\tau = 0.0000001$ sind dabei besser als die des größeren Wertes, allerdings vermuten wir hier eine deutlich längere Laufzeit.

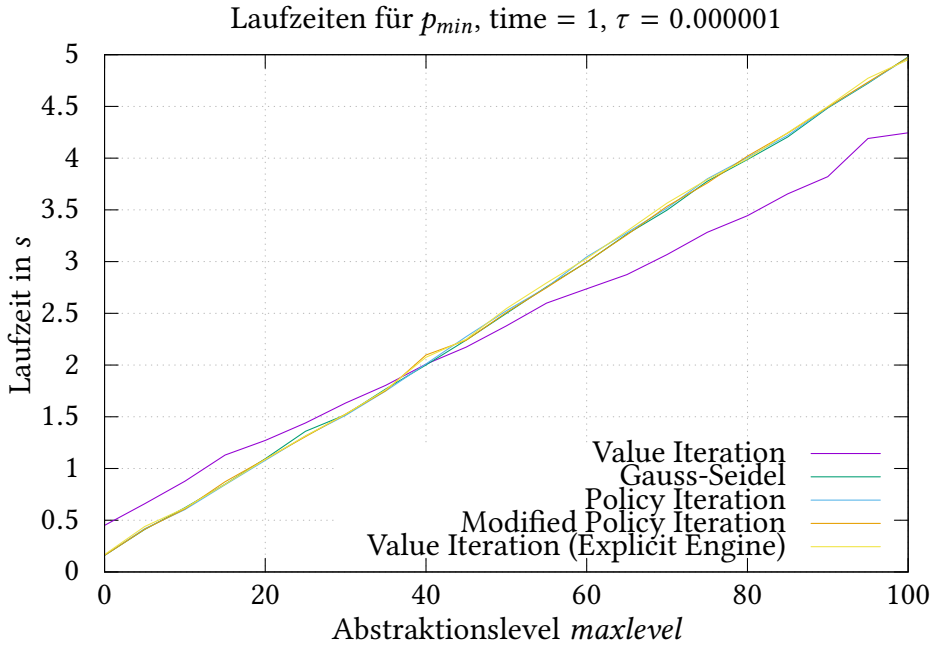


Abbildung 5.11.: Laufzeit in Sekunden für p_{min}, time = 1, *maxlevel* = 0 bis *maxlevel* = 100, τ = 0.000001 für verschiedene Lösungsverfahren.

p_{min} für τ = 0.000001: In Abbildung 5.11 sieht man, dass alle vier Verfahren für die beiden getesteten Engines eine sehr ähnliche Laufzeit haben. Die Varianz zwischen den Verfahren liegt selbst für 100 Level nur bei 0.734 Sekunden. In der Abbildung kann man erkennen, dass Value Iteration auf der Hybrid Engine in diesem Umfeld für eine geringe Levelanzahl das langsamste Verfahren ist. Alle Verfahren der Explicit Engine haben eine sehr ähnliche Laufzeit, die sich in unserem Test maximal um 0.096 Sekunden unterscheidet. Ab der Berechnung von 45 Leveln ändert sich etwas: ab hier ist Value Iteration das schnellste Verfahren. Eine Auflistung einer Auswahl von Laufzeiten, inklusive des Umbruchs zwischen 40 und 45 Ebenen, ist in Tabelle 5.9 dargestellt.

Level \Verfahren	VI	GS	PI	MPI	VI (Ex.)
10	0.877	0.621	0.603	0.609	0.618
40	2.009	2.001	2.011	2.097	2.075
45	2.173	2.241	2.274	2.240	2.249
100	4.245	4.979	4.969	4.973	4.952

Tabelle 5.9.: Laufzeit in Sekunden für die Verfahren Value Iteration (Hybrid Engine), Gauss-Seidel, Policy Iteration, Modified Policy Iteration, Value Iteration (Explicit Engine) (vlnr.) der Berechnung von p_{min} mit time = 1 und τ = 0.000001 bei verschiedenen Abstraktionsleveln.

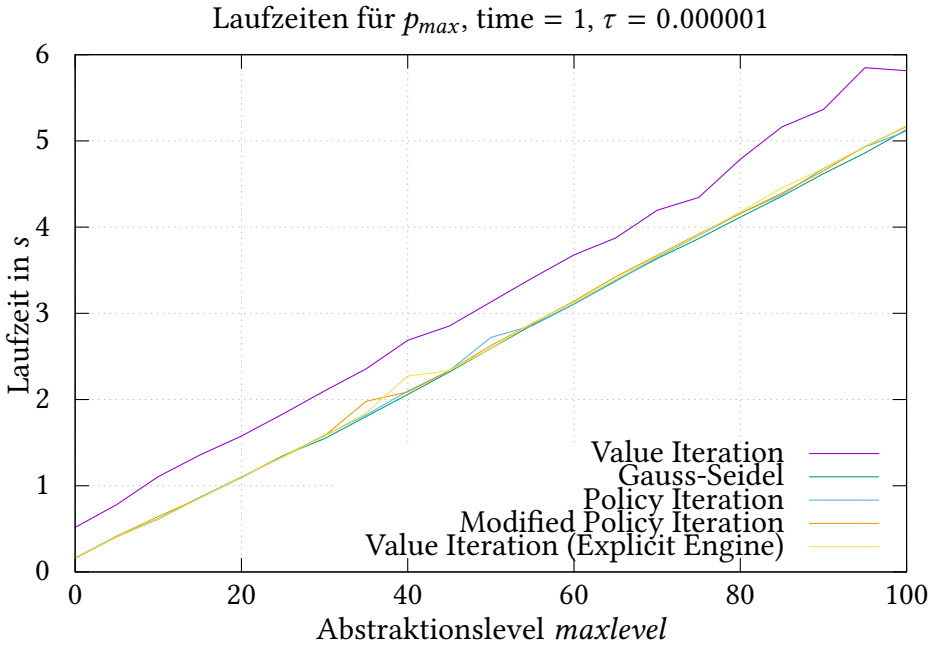


Abbildung 5.12.: Laufzeit in Sekunden für p_{max}, time = 1, *maxlevel* = 0 bis *maxlevel* = 100, $\tau = 0.000001$ für verschiedene Lösungsverfahren.

p_{max} für $\tau = 0.000001$: In Abbildung 5.12 werden die Laufzeiten für $\tau = 0.000001$ für die Berechnung von p_{max} dargestellt. Hier erkennt man auch, dass die verglichenen Verfahren sehr ähnliche Laufzeiten benötigen. Insbesondere die Verfahren auf der Explicit Engine unterscheiden sich kaum, die maximale Varianz liegt hier bei 0.214 Sekunden. In diesem Vergleich erkennt man, dass Value Iteration spürbar länger (im Schnitt 0.575 Sekunden) für das Berechnen der Ergebnisse braucht, als die anderen Verfahren, dies gilt allerdings nur für die Hybrid Engine.

p_{min} für $\tau = 0.0000001$: Abbildung 5.13 zeigt die Laufzeiten bei einer Diskretisierung mit $\tau = 10^{-7}$. Die Dauer ist gegenüber $\tau = 10^{-6}$ etwa verzehnfacht, wie intuitiv zu erwarten war. Je nach Level beträgt die Laufzeit zwischen 0 und 50 Sekunden. Die maximale Differenz der vier expliziten Verfahren beträgt 1.107 Sekunden. Die höchste Varianz inklusive Value Iteration beträgt 6.887 Sekunden bei 100 Leveln. Der Verlauf der Graphen unterscheidet sich in Bezug auf das relative Verhalten der Verfahren zueinander nicht zu dem des größeren τ. Auch hier ist Value Iteration auf der Hybrid Engine für wenige Level das schnellste Verfahren, für viele Level jedoch mit Abstand das langsamste.

p_{max} für $\tau = 0.0000001$: Der Test für p_{max} mit $\tau = 0.0000001$ ergibt ähnliche Ergebnisse wie für $\tau = 0.000001$, die Laufzeiten sind allerdings entsprechend verzehnfacht (vgl. Abbildung 5.14) Value Iteration auf der Hybrid Engine ist hier durchgängig das langsamste Verfahren, für 100 Level benötigt diese Methode $\approx$ 56 Sekunden.

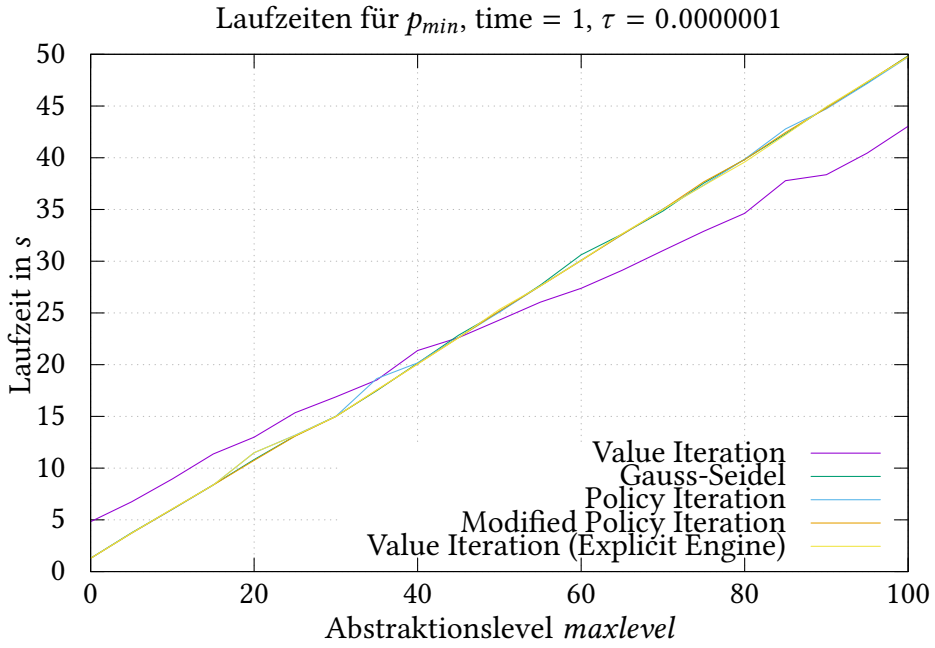


Abbildung 5.13.: Laufzeit in Sekunden für p_{min}, time = 1, *maxlevel* = 0 bis *maxlevel* = 100, τ = 0.0000001 für verschiedene Lösungsverfahren.

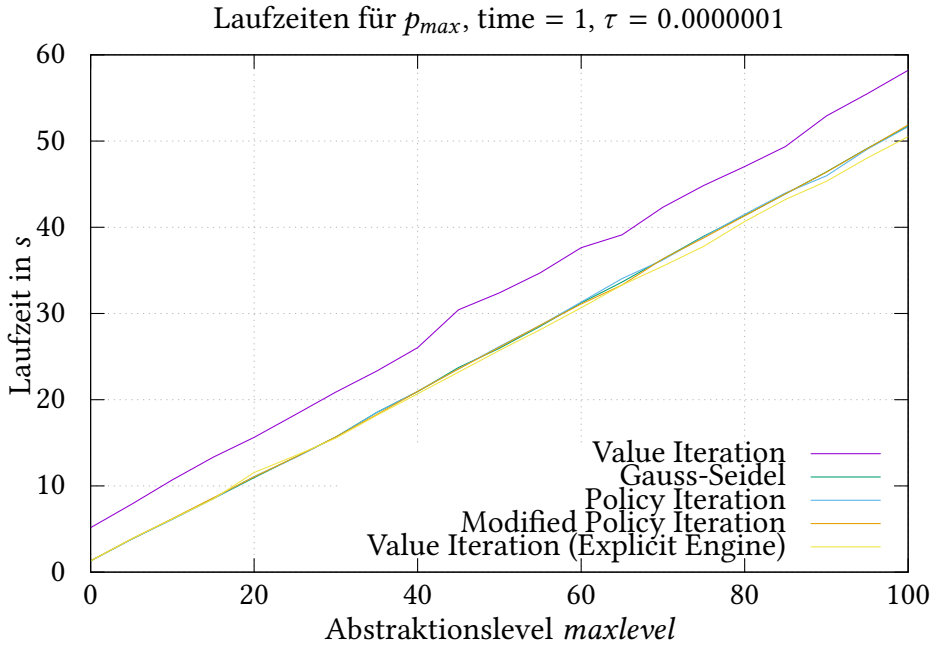


Abbildung 5.14.: Laufzeit in Sekunden für p_{max}, time = 1, *maxlevel* = 0 bis *maxlevel* = 100, τ = 0.0000001 für verschiedene Lösungsverfahren.

Fazit Zusammenfassend stellen wir fest, dass die Laufzeiten für Modelle dieser Größe und Granularität nicht allzu stark variieren. Generell kann man jedoch sagen, dass hier Value Iteration auf der Hybrid Engine für die Berechnung der minimalen Wahrscheinlichkeit zumindest für große *maxlevel* eine gute Wahl ist. Für kleinere *maxlevel* oder die Berechnung der minimalen Wahrscheinlichkeit scheint die Explicit Engine schnellere Berechnungen durchzuführen.

5.5. Vergleich zu Ergebnissen eines anderen Lösungsansatzes

In diesem Abschnitt wollen wir einen Vergleich zu einem anderen Lösungsansatz ziehen. In [4] wurde für das von uns betrachtete Problem eine andere Methode eingeführt. Insbesondere wird hier auch auf das Modell aus [11] Bezug genommen, für das auch wir unser Verfahren untersucht haben.

Im Rahmen der zitierten Arbeit werden auch für die in 5.2.2 eingeführte Erreichbarkeitsformeln Wahrscheinlichkeiten berechnet. Dafür wird das Modell bis zu einem maximalen Level m betrachtet, alle darauffolgenden Level werden abgeschnitten. Im Vergleich: Wir schneiden unser Modell gewissermaßen auch nach einem maximalen Level ab, ergänzen aber noch ein abstraktes Level, das nachfolgende Ebenen simuliert. Dadurch bleiben insbesondere im m-ten Level die Transitionen erhalten. Schneidet man das Modell ab, so verändert sich naheliegenderweise in dieser Ebene die Wahrscheinlichkeit für die verbliebenen Transitionen.

In Abbildung 5.15 vergleichen wir die entsprechenden Resultate für time = 1 mit unseren Ergebnissen für $\tau \in \{0.0001, 0.00001, 0.000001\}$.

Hier kann man erkennen, dass unsere Berechnungen immer kleinere Werte liefern, als die in [4] durchgeführten Berechnungen. Um die Unterschiede besser zu erkennen betrachten wir nähere Darstellungen davon in Abbildung 5.16. Hier kann man sehen, dass die feingranulare Berechnung mit $\tau = 10^{-6}$ sehr nah am berechneten Wert des alternativen Verfahrens liegt, allerdings gilt dies nur für p_{max}. Für den Wert $\tau = 0.0001$ haben wir in der Analyse ausreichende Korrektheit validiert. In der Nahansicht in Abbildung 5.16 jedoch kann man erkennen, dass er deutlich vom Ergebnis aus [4] abweicht.

Wir vermuten, dass der Grund für unsere minimal niedrigeren Wahrscheinlichkeiten die maximale Schrittanzahl ist. Da wir die beiden Modelle mit maximalem Level *maxlevel* betrachten, hat der MDP ein Level mehr als die CTMC, namentlich das abstrakte Level. Es ist somit im abgeschnittenen Modell etwas leichter innerhalb einer bestimmten Zeit die letzte Ebene zu erreichen. Dies löst eine minimale Verschiebung auf der *maxlevel*-Achse aus.

Interessant hingegen ist, dass das Abschneiden des QBDs scheinbar nahezu identische Ergebnisse liefert, wie die Berechnung der maximalen Wahrscheinlichkeit. Aufgrund der MDP-Abstraktion erwarten wir intuitiv, dass die tatsächliche Wahrscheinlichkeit zwischen unserem berechneten p_{max} und p_{min} liegt. Die in [4] berechnete Wahrscheinlichkeit jedoch

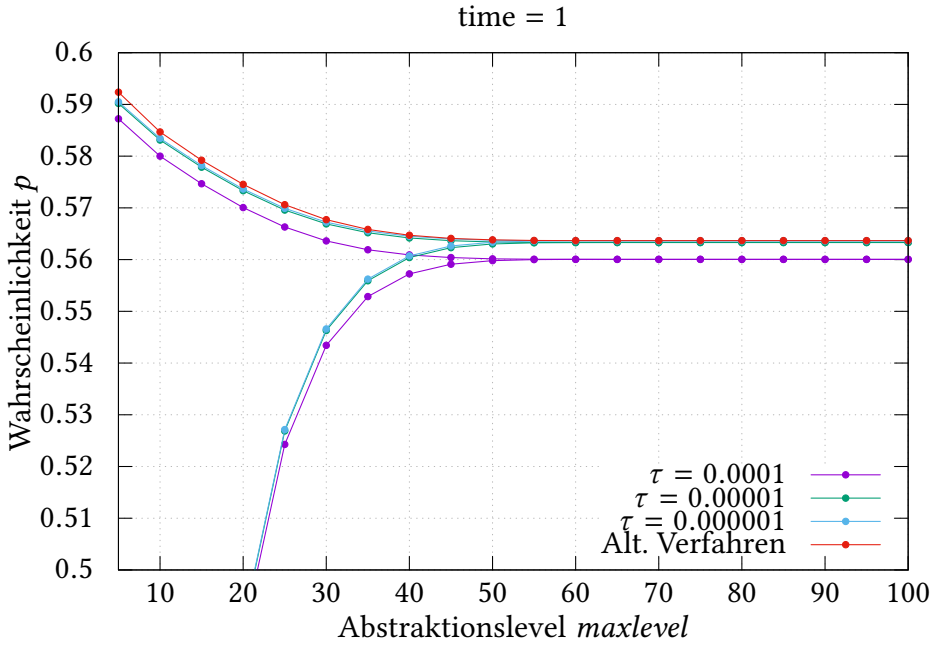


Abbildung 5.15.: Wahrscheinlichkeiten p_{min} und p_{max} für time = 1, *maxlevel* = 0 bis *maxlevel* = 100, $\tau = 0.0001$, $\tau = 0.00001$ und $\tau = 0.000001$ im Vergleich zu den in [4] berechneten Resultaten durch Abschneiden des QBD.

liegt nicht innerhalb dieser Spanne, sondern nahe der Grenzwahrscheinlichkeit p_{max}. Diesbezüglich kann also keine Verbesserung gegenüber dem Abschneiden erkannt werden.

Fazit Im Vergleich zu den Ergebnissen aus [4] haben wir erkannt, dass in der gewählten Fallstudie die Abstraktion keine Verbesserung gegenüber dem Abschneiden des QBDs zur Folge hat. In weiteren Studien wäre zu untersuchen, ob dieses Verhalten eine Regelmäßigkeit zeigt. Ob MDP-Abstraktion für andere Formeln oder QBDs eine Verbesserung gegenüber einfacher berechneten Werten bleibt offen, kann aber im Rahmen dieser Arbeit nicht untersucht werden.
Es bliebe außerdem zu untersuchen, inwiefern die beiden Verfahren sich in der Laufzeit unterscheiden. Mit geeigneten Tests zu aussagekräftigen Werten für time zu verschiedenen Granularitäten τ könnte man die beiden Verfahren mit Abschneiden bzw. Abstrahieren des QBDs nach dem Level *maxlevel* vergleichen und erkennen, ob die Abstraktion gegebenenfalls trotz dem Nachteil der Wahrscheinlichkeitsspanne praktikabler ist als das Abschneiden des QBDs.

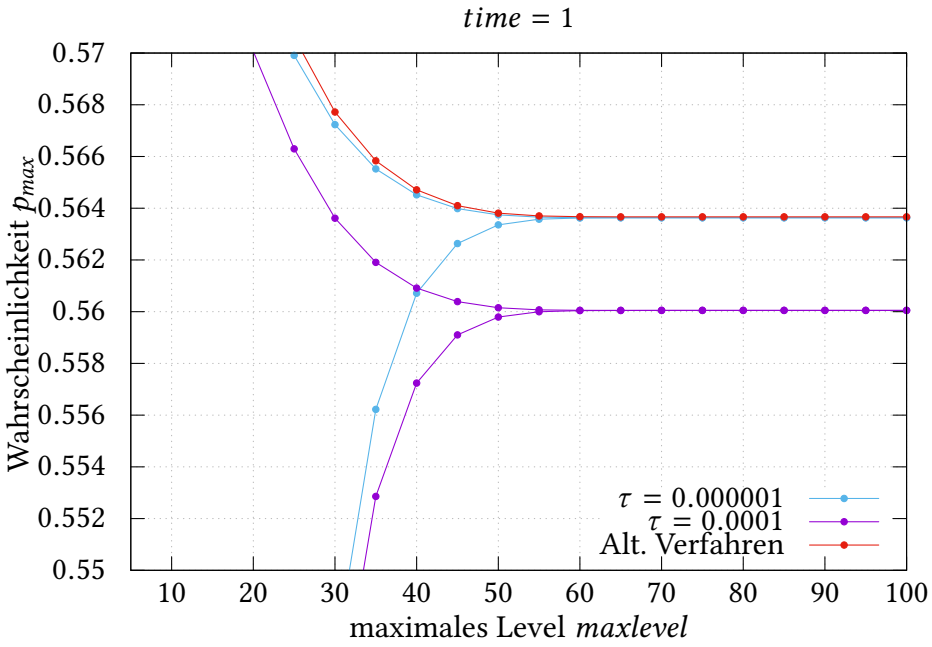


Abbildung 5.16.: Wahrscheinlichkeiten p_{min} und p_{max} für time = 1, *maxlevel* = 0 bis *maxlevel* = 100 im Vergleich zu den in [4] berechneten Resultaten durch Abschneiden des QBD in detailreicheren Darstellungen für zwei verschiedene τ.

6. Fazit und Ausblick

Im Laufe dieser Arbeit haben wir unendliche Zustandsübergangsmodelle mit einer starken Struktur, QBDs, vorgestellt. Wir haben zwei Verfahren kennengelernt, mit denen solche Modelle vereinfacht werden können: Zum einen die MDP-Abstraktion, welche Zustände in abstrakten Zuständen zusammenfasst und so für eine deutliche Reduktion des Modells sorgen kann. Dadurch können wir für ein unendliches Modell eine endliche Repräsentation erhalten. Zum anderen können mit der Diskretisierung kontinuierliche Modelle vereinfacht werden, indem der kontinuierliche Zeitfluss in einzelne Zeitschritte unterteilt wird. Das so erhaltene Modell verhält sich demnach diskret bezüglich dieses Zeitschrittes. Insbesondere ist es erheblich einfacher, Wahrscheinlichkeiten auf Basis des Modells zu berechnen – allerdings entsteht hierbei ein Fehler, welcher berechnet und in der Analyse berücksichtigt werden muss.
In der zweiten Hälfte der Arbeit haben wir uns mit einer Fallstudie beschäftigt, um die Qualität unserer Modelländerungen betrachten zu können. Wir haben die Modellierung des Kommunikationsprotokolls TCP vorgestellt und Abstraktion und Diskretisierung auf die CTMC angewendet. Auf dem erhaltenen MDP haben wir mit dem PRISM Model Checker Berechnungen für die Wahrscheinlichkeiten p_{min} und p_{max} einer Erreichbarkeitsformel durchgeführt. Dabei haben wir die verschiedenen Parameter time, für die Zeit, innerhalb der die Erreichbarkeitsformel erfüllt werden soll, *maxlevel*, für das Abstraktionslevel unseres QBD und τ für die Granularität der Diskretisierung betrachtet.
Wie erwartet verhalten sich die Ergebnisse für besonders feine Werte von τ deutlich besser, als bei einer groben Granularität. Ein zu großes τ liefert stark abweichende. Für time $= 1$ haben wir schließen können, dass eine Zeitschrittdauer $\tau = 0.0001$ ausreichend ist, um aussagekräftige Ergebnisse zu erhalten. Dieses Verhalten haben wir auch für time $= 2$ beobachten können. In weiterführenden Studien sollte untersucht werden, wie sich τ zu deutlich längeren Zeiten verhält, um herauszufinden, ob die Exaktheit der Ergebnisse von der Zeit time abhängt.
Aus den vielfältigen Tests zur Wahl des Abstraktionslevels haben wir mitgenommen, dass ein höheres Abstraktionslevel präzisere Resultate liefert. Die Wahrscheinlichkeiten p_{min} und p_{max} konvergieren für große Abstraktionslevel, sodass sich p_{min} und p_{max} in der Genauigkeit von Gleitkommazahlen nicht mehr unterscheiden. Durch den Vergleich zu den für time $= 2$ erhaltenen Ergebnissen haben wir festgestellt, dass dieser Punkt der Gleichheit für eine längere Zeit später eintritt. Daraus schließen wir, dass die Wahl des Abstraktionslevels von time abhängen muss. Es kann demnach keine allgemeingültige ausreichende Abstraktion gefunden werden. Um optimale Ergebnisse zu erzielen sollte das Abstraktionslevel dynamisch oder in Abhängigkeit der zu betrachtenden PCTL-Formeln gewählt werden.

Im Vergleich unserer Ergebnisse zu Resultaten einer Methode, in welcher der QBD nach einer bestimmten Ebene simpel abgeschnitten wird, haben wir außerdem keine sichtbare Verbesserung durch die Abstraktion erkannt. In weiteren Studien wäre es aufschlussreich, die Laufzeiten dieser beiden Verfahren miteinander zu vergleichen. Außerdem könnten eine Reihe von diverseren Tests veranschaulichen, ob die durch die Abstraktion erhaltenen Daten gegebenenfalls doch in einer leicht veränderten Konfiguration einen sichtbaren Vorteil mit sich bringen.

Literatur

[1] Christel Baier, Joost-Pieter Katoen u. a. *Principles of model checking*. Bd. 26202649. MIT press Cambridge, 2008 (siehe S. 1, 5, 12, 15–19, 22).

[2] Adrian Beer. „Quantitative Analysis of Concurrent System Architectures“. Masterarbeit. Universität Konstanz, 2012 (siehe S. 12).

[3] Sanaa Chafik und Cherki Daoui. „A Modified Policy Iteration Algorithm for Discounted Reward Markov Decision Processes“. In: *International Journal of Computer Applications* 133.10 (2016), S. 28–33 (siehe S. 54).

[4] Stefanie Eva Drerup. „CSL Model Checking von QBDs in PRISM“. Bachelorarbeit. Universität Münster, 2016 (siehe S. 3, 58–60).

[5] Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman und David Parker. „Automated verification techniques for probabilistic systems“. In: *Formal Methods for Eternal Networked Software Systems*. Springer, 2011, S. 53–113 (siehe S. 20, 22–24, 36, 54).

[6] E Moritz Hahn, Holger Hermanns, Björn Wachter und Lijun Zhang. „Time-bounded model checking of infinite-state continuous-time Markov chains“. In: *Fundamenta Informaticae* 95.1 (2009), S. 129–155 (siehe S. 3, 6).

[7] Daniel Klink, Anne Remke, Boudewijn R Haverkort und Joost-Pieter Katoen. „Time-bounded reachability in tree-structured QBDs by abstraction“. In: *Performance Evaluation* 68.2 (2011), S. 105–125 (siehe S. 1–3, 6, 9, 11).

[8] Martin R Neuhausser und Lijun Zhang. „Time-bounded reachability probabilities in continuous-time Markov decision processes“. In: *Quantitative Evaluation of Systems (QEST), 2010 Seventh International Conference on the*. IEEE. 2010, S. 209–218 (siehe S. 2, 3, 8, 12, 24).

[9] *PRISM Manual Version 4.3*. 14. Juli 2015. URL: http://www.prismmodelchecker.org/manual/Main/AllOnOnePage (siehe S. 33, 53).

[10] Anne Remke. „Model checking structured infinite Markov chains“. Diss. University of Twente, 2008 (siehe S. 1).

[11] Anne Remke, Boudewijn R Haverkort und Lucia Cloth. „CSL model checking algorithms for QBDs“. In: *Theoretical Computer Science* 382.1 (2007), S. 24–41 (siehe S. 3, 6, 7, 25, 31, 36, 38, 39, 41, 58).

[12] *The PRISM Language*. 3. Dez. 2010. URL: http://www.prismmodelchecker.org/manual/ThePRISMLanguage/AllOnOnePage (besucht am 17. 04. 2016) (siehe S. 34).

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

A. PRISM Quellcode

```
mdp

const double e = 2.71828182845904523536028747135266249775724709369995957496 6;
const double tau;
const double time;
formula ex = pow(e,-tau);
const int steps = floor(time / tau);

const double alpha = 1;
const double beta = 0.04;
const double r = 10;
const double c = 10;
const int maxlevel;
const int a = maxlevel+1;
const double lambda = 100;
const double mu = 125;

module tcp
i: [0..a] init 0; // Warteschlange
j: [0..1] init 0; // connection management
k: [0..1] init 1; // packet generator


// Grundlevel
[] i=0 & j=0 & k=0
-> (1-pow(ex,beta)) : (k'=1) // von off zu burst
+  (pow(ex,beta)): true;

[] i=0 & j=0 & k=1
-> (1-pow(ex,alpha+lambda))*alpha/(alpha+lambda): (k'=0) // von burst zu off
+  (1-pow(ex,alpha+lambda))*(lambda/(alpha+lambda)): (i'=1) // ins naechste Level
+  (pow(ex,alpha+lambda)): true;

[] i=0 & j=1 & k=0
-> (1-pow(ex,beta+r))*(beta/(beta+r)): (k'=1) // von off zu burst
+  (1-pow(ex,beta+r))*(r/(beta+r)): (j'=0) // von active zu released
+  pow(ex,beta+r): true;

[] i=0 & j=1 & k=1
-> (1-pow(ex,alpha+r+lambda))*alpha/(alpha+r+lambda): (k'=0) // von burst zu off
+  (1-pow(ex,alpha+lambda+r))*(r/(alpha+lambda+r)): (j'=0) // von active zu released
+  (1-pow(ex,alpha+lambda+r))*(lambda/(alpha+lambda+r)): (i'=1) // ins naechste Level
+  pow(ex,alpha+r+lambda): true;

// Wiederholende Level
[] i>0 & i<a & j=0 & k=0
-> (1-pow(ex,beta+c))*(beta/(beta+c)) : (k'=1) // von off zu burst
+ (1-pow(ex,beta+c))*(c/(beta+c)): (j'=1) // von released zu active
+ pow(ex,beta+c): true;

[] i>0 & i<a & j=0 & k=1
-> (1-pow(ex,alpha+c+lambda))*alpha/(alpha+c+lambda): (k'=0) // von burst zu off
```

```
53  +  (1-pow(ex,alpha+lambda+c))*(c/(alpha+lambda+c)): (j'=1) // von released zu active
54  +  (1-pow(ex,alpha+lambda+c))*(lambda/(alpha+lambda+c)): (i'=i+1) // ins naechste Level
55  +  pow(ex,alpha+c+lambda): true;
56
57  [] i>0 & i<a & j=1 & k=0
58  -> (1-pow(ex,beta+mu))*(beta/(beta+mu)) : (k'=1) // von off zu burst
59  +  (1-pow(ex,mu+beta))*(mu/(mu+beta)): (i'=i-1) // ins vorherige Level
60  +  pow(ex,mu+beta): true;
61
62  [] i>0 & i<a & j=1 & k=1
63  -> (1-pow(ex,alpha+lambda+mu))*(alpha/(alpha+lambda+mu)): (k'=0) // von burst zu off
64  +  (1-pow(ex,alpha+lambda+mu))*(lambda/(alpha+lambda+mu)): (i'=i+1) // ins naechste Level
65  +  (1-pow(ex,alpha+lambda+mu))*(mu/(alpha+lambda+mu)): (i'=i-1) // ins vorherige Level
66  +  pow(ex,alpha+lambda+mu): true;
67
68  // Abstraktes Level
69  // default-Transitionen
70  [] i=a & j=0 & k=0
71  -> (1-pow(ex,beta+c))*(beta/(beta+c)): (k'=1) // von off zu burst
72  +  (1-pow(ex,beta+c))*(c/(beta+c)): (j'=1) // von released zu active
73  +  pow(ex,beta+c): true;
74
75  [] i=a & j=0 & k=1
76  -> (1-pow(ex,alpha+lambda+c))*(alpha/(alpha+lambda+c)): (k'=0) // von burst zu off
77  +  (1-pow(ex,alpha+lambda+c))*(c/(alpha+lambda+c)): (j'=1) // von released zu active
78  +  (1-pow(ex,alpha+lambda+c))*(lambda/(alpha+lambda+c)): true // im Level bleiben
79  +  pow(ex,alpha+lambda+c): true;
80
81  // close-Transitionen
82  [close] i=a & j=1 & k=0
83  -> (1-pow(ex,beta+mu))*(beta/(beta+mu)): (k'=1) // von off zu burst
84  +  (1-pow(ex,beta+mu))*(mu/(beta+mu)): (i'=maxlevel) // ins "vorherige" Level (maxlevel)
85  +  pow(ex,beta+mu): true;
86
87  [close] i=a & j=1 & k=1
88  -> (1-pow(ex,lambda+alpha+mu))*(alpha/(lambda+alpha+mu)): (k'=0) // von  burst zu off
89  +  (1-pow(ex,lambda+alpha+mu))*(mu/(lambda+alpha+mu)): (i'=maxlevel) // ins "vorherige"
       Level (maxlevel)
90  +  (1-pow(ex,lambda+alpha+mu))*(lambda/(lambda+alpha+mu)): true // im abstrakten Level
       bleiben
91  +  pow(ex,lambda+alpha+mu): true;
92
93  // far-Transitionen
94  [far] i=a & j=1 & k=0
95  -> (1-pow(ex,beta+mu))*(beta/(beta+mu)): (k'=1) // von off zu burst
96  +  (1-pow(ex,beta+mu))*(mu/(beta+mu)): true // im abstrakten Level bleiben
97  +  pow(ex,beta+mu): true;
98
99  [far] i=a & j=1 & k=1
100 -> (1-pow(ex,lambda+alpha+mu))*(alpha/(lambda+alpha+mu)): (k'=1) // von burst zu off
101 +  (1-pow(ex,lambda+alpha+mu))*((lambda+mu)/(lambda+alpha+mu)): true // im abstrakten
       Level bleiben
102 +  pow(ex,lambda+alpha+mu): true;
103
104 endmodule
105
106 // labels
107 label "burst" = k=1;
108 label "off" = k=0;
109 label "active" = j=1;
110 label "released" = j=0;
```

Listing A.1: PRISM Quellcode des TCP-Modells.

```
1 Pmax=? [ true U[0,steps] ("released")&(!("burst")) ]
```

Listing A.2: `pmax.props`: PCTL-Eigenschaft p_{max} als PRISM-Eingabe

```
1 Pmin=? [ true U[0,steps] ("released")&(!("burst")) ]
```

Listing A.3: `pmin.props`: PCTL-Eigenschaft p_{min} als PRISM-Eingabe

B. Hilfs-Quellcode

Die Programme `run_varMaxlevel.sh` und `run_varTau.sh` führen PRISM-Befehle mit Variation von *maxlevel* oder τ aus.

```
1  #!/bin/sh
2  export PATH="${PATH}:/home/janic/programs/prism/bin/"
3
4  #Fuehrt PRISM Kommandozeilenaufrufe aus mit den angegebenen Parametern time, maxlevel von
       m_start bis m_end in Schritten der Groesse m_step, fuer verschiedene Werte von tau
       zwischen 10-smallest_exp und 10^-biggest_exp
5
6  #Parametereingabe:
7  #time
8  time="2"
9  #maxlevel
10 m_start="0"
11 m_step="5"
12 m_end="100"
13 #tau
14 smallest_exp=-1
15 biggest_exp=-7
16
17 #Ab hier nichts aendern.
18 mkdir results
19 mkdir output
20
21 for t in `seq ${smallest_exp} -1 ${biggest_exp}`
22 do
23   tau=1E${t}
24   for opt in "max" "min"
25   do
26     filename="${opt}_time${time}_maxlevel${m_start}-${m_step}-${m_end}_tau${tau}"
27     prism ./ctmdp.prism ./p${opt}.props -const tau=${tau},time=${time},maxlevel=${m_start
           }:${m_step}:${m_end} -exportresults ./results/${filename}.txt > ./output/${
           filename}.txt
28   done
29   echo "fuer ${tau} berechnet."
30 done
31 echo "fertig."
```

Listing B.1: Bash-Code: Programm `run_varMaxlevel.sh` zur Berechnung von p_{max} und p_{min} mit einem schrittweise angegebenen *maxlevel* für verschiedene τ.

In `run_varMaxlevel.sh` (vgl. B.1) werden PRISM-Aufrufe ausgeführt, denen ein festes τ mitgegeben wird und eine schrittweise Angabe für *maxlevel*. Im Skript können aber auch verschieden Werte für τ über die Änderung der Parameter `smallest_exp` und `biggest_exp` eingegeben werden. Für diese Exponenten t werden die PRISM-Aufrufe mit dem schrittweise veränderten *maxlevel* für alle $\tau = 10^t$, also mit der aktuellen Eingabe für $\tau = 10^{-1}, 10^{-2}, \ldots, 10^{-7}$ berechnet.

```
#!/bin/sh
export PATH="${PATH}:/home/janic/programs/prism/bin/"

#Fuehrt PRISM Kommandozeilenaufrufe aus mit den angegebenen Parametern time, logarithmisch kleiner werdende Werte von tau bis zu 10^-biggest_exp, fuer ggf verschiedene Werte fuer maxlevel von m_start bis m_end in Schritten der Groesse m_step

#Parametereingabe:
#time
time="2"
#maxlevel
m_start=10
m_step=10
m_end=40
#tau
biggest_exp=-7

#Ab hier nichts aendern.
mkdir results
mkdir results/helper
mkdir output

for maxlevel in `seq ${m_start} ${m_step} ${m_end}`
do
  for t in `seq -1 -1 ${biggest_exp}`
  do
    tb=`expr ${t} + 1`
    tau_start=1E${t}
    tau_step=1E${t}
    tau_end=1E${tb}
    for opt in "max" "min"
    do
      filename="${opt}_time${time}_maxlevel${maxlevel}_tau${tau_start}-${tau_step}-${tau_end}"
      prism ./ctmdp.prism ./p${opt}.props -const tau=${tau_start}:${tau_step}:${tau_end},time=${time},maxlevel=${maxlevel} -exportresults ./results/helper/${filename}.txt > ./output/${filename}.txt
    done
    echo "bis ${tau_start} berechnet."
  done
  for opt in "max" "min"
  do
    cat ./results/helper/${opt}_time${time}_maxlevel${maxlevel}_* > ./results/helper/${opt}_time${time}_maxlevel${maxlevel}.txt
    grep -v [[:alpha:]] ./results/helper/${opt}_time${time}_maxlevel${maxlevel}.txt > ./results/${opt}_time${time}_maxlevel${maxlevel}_log.txt
    echo "fuer maxlevel=${maxlevel} berechnet und zum plotten zusammengefuegt."
  done
done
echo "fertig."
```

Listing B.2: Bash-Code: Programm `run_varTau.sh` zur Berechnung von p_{max} und p_{min} mit einem variierenden τ für verschiedene *maxlevel*.

In `run_varTau.sh` (vgl. B.2) wird den PRISM-Aufrufen ein fester Wert für *maxlevel* mitgegeben. Es können jedoch auch hier verschieden Werte für *maxlevel* mitgegeben werden. Die PRISM-Aufrufe werden dann für jeden dieser Werte ausgeführt. Außerdem wird in der Eingabe ein kleinster und ein größter Exponent für τ erwartet. Das Skript führt dann die entsprechenden Berechnungen zwischen diesen beiden Werten durch und fügt anschließend die Resultate aller τ für ein *maxlevel* zusammen. Dadurch können mit einem

geeigneten Tool die logarithmischen Betrachtungen wie in Abbildung 5.4 generiert werden. Das Programm `runPrism.sh` (vgl. B.3) wurde verwendet, um für verschiedene Engines und Lösungsmethoden bei der Berechnung eine Laufzeitausgabe zu speichern. Die Auswahl der Engine bzw. Methode, sowie Werte für time, *maxlevel* und τ kann beeinflusst werden, indem an der jeweiligen Stelle im Code der gewünschte Wert eingegeben wird. Das Programm führt automatisch für alle angegebenen Methoden, Engines und Werte für τ Laufzeitberechnungen durch, time und *maxlevel* müssen zu Beginn fest gewählt werden. Mit den Skripten `cutMaxlevel.sh` und `cutTime.sh`, die in Listing B.4 und B.5 aufgeführt sind, werden aus der von PRISM gelieferten Ausgabe die Parameter für *maxlevel* sowie die benötigte Sekundenzahl ausgeschnitten und anschließend mit dem `pr`-Befehl in eine Datei geschrieben.

```
#!/bin/sh
export PATH="${PATH}:/home/janic/programs/prism/bin/"

#Fuehrt in PRISM die Berechnungen zur Laufzeitanalyse und schneidet mit den passenden Skripten cutTime.sh und cutMaxlevel.sh die Ausgabedateien auf die Laufzeitangaben zu.

#Hier ggf. die Werte fuer time und maxlevel anpassen.
time="1"
maxlevel="0:5:100"
#Hier ggf. die gewuenschten Loesungsverfahren angeben. "valiter" Value Iteration, "gs" Gauss-Seidel, "politer" Policy Iteration, "modpoliter" Modified Policy Iteration
for method in "valiter" "gs" "politer" "modpoliter"
do
  #Hier ggf. die gewuenschten Engines anpassen. "" ist der Standard fuer die jeweilige Methode, "ex" Explicit, "s" Sparse, "m" MTBDD
  for engine in "" "ex" "s" "m"
  do
    if [ "${engine}" = "" ]
    then
      engine_used=""
      engine_name=""
    elif [ "${method}" = "valiter" ]
    then
      engine_used="-${engine}"
      engine_name="_${engine}"
    else
      continue
    fi

    #Hier die gewuenschten Werte fuer tau eingeben.
    for tau in "1E-2" "1E-3"
    do
        for opt in "max" "min"
        do
          filename="${method}${engine_name}_${opt}_time${time}_maxlevel0-5-100_tau${tau}"
          prism ./ctmdp.prism ./p${opt}.props -${method} ${engine_used} -const tau=${tau},time=${time},maxlevel=${maxlevel} -exportresults ./logs/${filename}.txt > ./output/${filename}.txt
          ./cutTime.sh    ./output/${filename}.txt ./output/${filename}_cutTime.txt
          ./cutMaxlevel.sh  ./output/${filename}.txt ./output/${filename}_cutMaxlevel.txt
          pr -m -t ./output/${filename}_cutMaxlevel.txt ./output/${filename}_cutTime.txt > ./output/${filename}_plot.txt
        done
    echo "${method} ${engine}: tau=${tau} berechnet."
    done
  done
```

```
done
echo "fertig."
```

Listing B.3: Bash-Code: Programm `runPrism.sh` zur Berechnung von p_{max} und p_{min} für verschiedene Parameter sowie für verschiedene Engines und Lösungsmethoden, inklusive Ausgabe der Laufzeiten.

```
#!/bin/sh

cat ${1} | grep "Model constants: " | cut -d "," -f3 | grep -o -P "[0-9]{1,3}" | awk 'NR %
     2 == 0' > ${2}
```

Listing B.4: Quellcode des `cutMaxlevel.sh`-Skripts, welches aus der Ausgabe die *maxlevel* filtert.

```
#!/bin/sh

cat ${1} | grep "Time for model checking: " | grep -o -P '[0-9]{1,5}.[0-9]{1,3}' > ${2}
```

Listing B.5: Quellcode des `cutTime.sh`-Skripts, welches aus der Ausgabe die Laufzeiten in Sekunden ausgibt.

Eidesstattliche Erklärung

Hiermit versichere ich, *Joanna Georgia Delicaris*, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Gedanklich, inhaltlich oder wörtlich Übernommenes habe ich durch Angabe von Herkunft und Text oder Anmerkung belegt bzw. kenntlich gemacht. Alle Bilder und Tabellen sind von mir selbst erstellt. Inhaltlich oder gedanklich Übernommenes habe ich auch hier entsprechend gekennzeichnet.

Alle auf der CD beigefügten Dateien sind von mir selbst erstellt oder durch Angabe von Herkunft kenntlich gemacht worden.

Münster, der 12. Mai 2016

Joanna Georgia Delicaris