

Seminar

Eine Logik für das Schlussfolgern über Zeit und Zuverlässigkeit

Theodor Teslia

Informatik 11 – Embedded Software
RWTH Aachen University
Aachen, Germany
teslia@embedded.rwth-aachen.de

Betreuer
Robin Mross

Abstract: Es wird eine Logik für die Verifikation von Soft-Realtime Systemen vorgestellt. Diese ermöglicht es, Aussagen über Zeit und Wahrscheinlichkeiten zu treffen. Weiter werden äußerst performante Model-Checking Algorithmen besprochen, welche ein algorithmisches Auswerten der Formeln ermöglichen. Durch den Vergleich mit weiteren, sehr Ausdrucksstarken Logiken, kommen wir schließlich zu dem Ergebnis, dass die Logik sehr viel über Wahrscheinlichkeiten aussagen kann, für komplexe zeitliche Zusammenhänge aber eher ungeeignet ist.

1 Einführung

In vielen Systemen sind zeitliche Bedingungen für das auftreten bestimmter Ereignisse wichtig. Betrachtet man bspw. einen Getränkeautomaten, so ist ein wichtiges Kriterium, dass nach dem Auswählen des Getränks möglichst wenig Zeit vergeht, bis dieses entnommen werden kann. Wird die dafür definierte Deadline aber verfehlt, so folgen keine katastrophalen Kosten. Das Einhalten einer Deadline sollte also erstrebt werden, ein Verfehlen aber nicht sofort zum Systemabsturz führen. Solche Systeme nennt man Soft-Realtime Systeme. Im Gegensatz dazu nennt man Hard-Realtime Systeme die, dessen Deadlines unter keinen Umständen verfehlt werden dürfen. Möchte man die Korrektheit solcher Systeme überprüfen, so ist es leicht einsehbar, dass das Verwerfen einer Formel bei nicht-Einhaltung einer Deadline sinnvoll ist, da es bei dieser Art an System vermieden werden muss Deadlines zu verfehlen. Schwieriger wird solch eine Überlegung bei Soft-Realtime Systemen. Das Einhalten von Deadlines sollte angestrebt werden, das Verpassen dieser ist aber in den meisten Fällen nicht sonderlich kritisch. Ein möglicher Weg dafür ist es, nicht nur über Zeit, sondern auch über Wahrscheinlichkeiten aussagen treffen zu können. Dadurch ist es dann möglich Aussagen der Form „Eine Deadline wird in 95% der Fälle ein-

gehalten“ zu treffen. Die in dieser Arbeit diskutierte Logik ermöglicht genau solche Aussagen. Genauer geht es um *Probabilistic Computation Tree Logic* (PCTL) von Hansson und Jonsson ([HJ94]). In dem Sinne soll die Logik untersucht werden, Model-Checking Algorithmen besprochen und dann mit weiteren Logiken verglichen werden.

Der Aufbau der Arbeit sieht wie folgt aus: In Kapitel 2 sollen wichtige Grundlagen bzgl. CTL und Transitionssystemen besprochen werden, dadurch setzt die Arbeit keine Kenntnisse über temporale Logiken voraus. In Kapitel 3 wird dann PCTL besprochen. Dafür wird zuerst die Syntax, die dazugehörigen Strukturen und Semantik erklärt. Darauf werden die wichtigsten Model-Checking Algorithmen vorgestellt und schließlich in einem Beispiel zur Verifikation angewendet. In Kapitel 4 werden zwei weitere Logiken vorgestellt, welche Aussagen über Zeit bzw. Wahrscheinlichkeiten treffen können. Beide sind bzgl. ihrer Domäne sehr mächtig und aus dem anschließenden Vergleich mit PCTL können wir wichtige Erkenntnisse ziehen.

2 Grundlagen

Um Erweiterungen einer Logik zu verstehen ist es ratsam, die Grundlegende auch zu kennen. Aus diesem Grund soll in diesem Kapitel in die Logik *Computation Tree Logic* (CTL) und naheliegende, wichtige Themen eingeleitet werden.

CTL ist eine temporale Logik um Aussagen in nicht-deterministischen Systemen zu treffen. Dafür betrachtet man Kripkestrukturen, welche eine Art Graph darstellen, da diese viele Eigenschaften liefern, die für CTL-Anwendungsfälle interessant sind.

Definition 2.1 (Syntax von CTL)

Die Menge der CTL-Formeln lässt sich induktiv mithilfe folgender Regeln definieren [CE82, BK08]:

1. Wenn AP eine Menge an atomaren Aussagen ist, dann ist jedes $a \in AP$ eine CTL-Formel. Außerdem sind \top und \perp CTL-Formeln.
2. Wenn φ_1 und φ_2 CTL-Formeln sind, dann sind $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$ und $\varphi_1 \vee \varphi_2$ ebenfalls CTL-Formeln.
3. Wenn φ_1 eine CTL-Formel ist, dann sind auch $EX \varphi_1$ und $AX \varphi_1$ CTL-Formeln.
4. Wenn φ_1 und φ_2 CTL-Formeln sind, dann sind $E[\varphi_1 U \varphi_2]$, $A[\varphi_1 U \varphi_2]$, $E[\varphi_1 W \varphi_2]$ und $A[\varphi_1 W \varphi_2]$ auch CTL-Formeln.

Bevor die Semantik erläutert wird, sollen zuerst die Strukturen gezeigt werden, die man typischerweise mit CTL zusammen verwendet.

Definition 2.2 (Kripkestrukturen)

Eine *Kripkestruktur* oder *Transitionssystem* ist ein Graph von der Form $\mathcal{K} = (V, E, \mathcal{L})$, wobei E eine zweistellige Kantenrelation über V ist und \mathcal{L} eine Funktion $\mathcal{L} : V \rightarrow 2^{AP}$ ist, die jedem Zustand eine Menge an atomaren Aussagen zuweist [CE82, CES86].

Weiter bezeichnen wir einen Pfad als $\sigma = s_0 s_1 \dots$ mit $s_i \in V$ für alle $i \in \{0, 1, \dots\}$ und $\sigma[i] = s_i$ für $i \in \mathbb{N}$, bzw. für endliche Pfade $\sigma = s_0 \dots s_n$ und $\sigma[i] = s_i$ für $0 \leq i \leq n$ [BK08].

Definition 2.3 (Semantik von CTL)

Für ein Transitionssystem $\mathcal{K} = (V, E, \mathcal{L})$ können wir damit induktiv die Modellbeziehung \models zwischen einem Knoten $v \in V$ und CTL-Formeln definieren [BK08]:

1. $v \models \top \Leftrightarrow \mathcal{K} \models \forall x(x = x)$ und $v \models \perp \Leftrightarrow \mathcal{K} \models \exists x(x \neq x)$.
2. Für $a \in \text{AP}$ gilt $v \models a \Leftrightarrow a \in \mathcal{L}(v)$.
3. Es gilt $v \models \neg\varphi \Leftrightarrow v \not\models \varphi$,
 $v \models \varphi_1 \wedge \varphi_2 \Leftrightarrow v \models \varphi_1$ und $v \models \varphi_2$,
 $v \models \varphi_1 \vee \varphi_2 \Leftrightarrow v \models \varphi_1$ oder $v \models \varphi_2$.
4. Es gilt $v \models \text{EX } \varphi \Leftrightarrow$ es ex. ein $w \in V$ mit $(v, w) \in E$ und $w \models \varphi$ und analog:
 $v \models \text{AX } \varphi \Leftrightarrow$ für alle $w \in V$ mit $(v, w) \in E$ gilt $w \models \varphi$.
5. Es gilt $v \models \text{E}[\varphi_1 \text{ U } \varphi_2] \Leftrightarrow$ es existiert ein Pfad σ in \mathcal{K} , der in v beginnt und ein $i \in \mathbb{N}$ so, dass $\sigma[i] \models \varphi_2$ und für alle $0 \leq j < i$ gilt $\sigma[j] \models \varphi_1$. Analog ist die Definition für $\text{A}[\varphi_1 \text{ U } \varphi_2]$, es muss dann aber für jeden in v beginnenden Pfad ein $i \in \mathbb{N}$ geben so, dass der i -te Zustand φ_2 und alle Zustände davor φ_1 erfüllen.
6. Es gilt $v \models \text{E}[\varphi_1 \text{ W } \varphi_2] \Leftrightarrow$ es existiert ein (unendlicher) Pfad σ so, dass entweder ein $i \in \mathbb{N}$ existiert mit $\sigma[i] \models \varphi_2$ und für alle $0 \leq j < i$ gilt $\sigma[j] \models \varphi_1$, oder es gilt $\sigma[i] \models \varphi_1$ für alle $i \in \mathbb{N}$. Für $\text{A}[\varphi_1 \text{ W } \varphi_2]$ ist die Definition analog, bloß mit einem All- anstatt Existenzquantor.

Intuitiv bedeutet $\text{EX } \varphi$ also, dass φ in einem beliebigen Nachfolgezustand gelten muss und $\text{AX } \varphi$, dass φ von allen Nachfolgezuständen erfüllt wird.

$\text{E}[\varphi_1 \text{ U } \varphi_2]$ bzw. $\text{A}[\varphi_1 \text{ U } \varphi_2]$ sagt aus, dass es einen Pfad σ gibt bzw. auf allen Pfaden σ gilt, dass zuerst φ_1 gilt, bis von einem Zustand φ_2 erfüllt wird. Die anderen Operatoren haben die bekannte Bedeutung.

Um einige Eigenschaften einfacher auszudrücken werden zusätzlich noch weitere Operatoren definiert [CE82]:

- $\text{EF } \varphi \equiv \text{E}[\top \text{ U } \varphi]$ und $\text{AF } \varphi \equiv \text{A}[\top \text{ U } \varphi]$ bedeuten intuitiv, dass es einen Pfad gibt bzw. für alle Pfade irgendwann φ gilt.
- $\text{EG } \varphi \equiv \neg \text{EF } \neg \varphi$ und $\text{AG } \varphi \equiv \neg \text{AF } \neg \varphi$ bedeuten, dass es einen Pfad gibt bzw. für alle Pfade gilt, dass in jedem Zustand φ gilt.

Mithilfe dieser Syntax und zugehöriger Semantik auf Transitionssystemen wollen wir nun eine Beispielhafte CTL-Formel betrachten:

Beispiel 1 (Beispiele für CTL-Formeln und deren Bedeutung)

Sei die Menge der atomaren Aussagen $AP = \{\text{idle}, \text{error}\}$. Dann besagt die Formel $EF\ AG\ \text{error}$, dass es einen Pfad gibt, auf dem ab irgendeinem Zustand alle Zustände die atomare Aussage error erfüllen. Das heißt, es gibt einen Pfad mit einem irreversiblen Fehler.

Zusätzlich soll noch angemerkt werden, dass $0 \in \mathbb{N}$ gilt.

3 Eine Logik für Zeit und Zuverlässigkeit

Wie in Kapitel 2 gezeigt, lassen sich mithilfe von CTL viele interessante Eigenschaften von nicht-deterministischen Systemen beschreiben. Jedoch gibt es auch Anwendungsfälle, in denen mehr Ausdruckskraft benötigt wird, als ein All- und Existenzquantor liefern können. Ein Gebiet, in dem dies stark auffällt, sind Soft-Realtime Systeme, wie sie in der Einführung definiert wurden. Um eben solche Systeme gut beschreiben zu können, benötigt man zwei weitere Aspekte:

1. Um die Zeitschranken zu formulieren, wird ein Konzept von Zeit benötigt. Dieses soll aussagen können, dass zwei Ereignisse eine bestimmte Zeitspanne t voneinander entfernt sind.
2. Da aber das Verfehlen einer Zeitschranke nicht unbedingt zum Verwerfen einer Formel führen soll, werden zusätzlich Wahrscheinlichkeiten benötigt. Da in Soft-Realtime Systemen das Überschreiten einer Deadline zwar nicht direkt verboten ist, aber im Allgemeinen vermieden werden sollte, ist es sinnvoll über die Wahrscheinlichkeit eines Ereignisses Aussagen zu treffen.

Kombiniert man diese Aspekte, lassen sich Eigenschaften wie „Nach Ereignis X passiert innerhalb von 15 Zeiteinheiten mit Wahrscheinlichkeit 90% Ereignis Y“ oder „Ereignis A tritt mit einer Wahrscheinlichkeit von 90% in 10 und mit 95% in 20 Zeitschritten auf“. Eine Logik, die eben diese Erweiterungen von CTL sinnvoll implementiert, ist die Logik Probabilistic Computation Tree Logic (PCTL). Sinnvoll bedeutet hier, dass es einen Model Checking Algorithmus mit polynomieller Laufzeit gibt. In diesem Kapitel soll zuerst die Syntax von PCTL erläutert werden, danach die dazugehörige Semantik aufgezeigt werden, um dann zwei verschiedene Ansätze für das Model Checking von PCTL mit Transitionssystemen zu zeigen. Im Anschluss sollen die kennengelernten Konzepte der Logik sowie des Model Checkings an einem Beispiel erläutert werden.

3.1 Syntax und Semantik von PCTL

Um *Probabilistic Computation Tree Logic* (PCTL) besser zu verstehen, soll hier die Syntax, die Modelle, welche wir zum Auswerten verwenden, sowie die Semantik der Logik erläutert werden.

Wie auch für CTL können wir die Syntax von PCTL mithilfe folgender rekursiver Regeln definieren:

Definition 3.1 (Syntax von PCTL)

Die Menge der PCTL-Formeln lässt sich induktiv wie folgt definieren [HJ94]:

1. Es gilt $\top \in \text{PCTL}$ und $\perp \in \text{PCTL}$.
2. Wenn AP die Menge atomarer Aussagen ist, dann ist jedes $a \in \text{AP}$ eine PCTL Formel.
3. Wenn φ_1 und φ_2 PCTL-Formeln sind, dann sind $\neg\varphi_1$ und $(\varphi_1 \wedge \varphi_2)$ auch PCTL-Formeln.
4. Für zwei PCTL-Formeln φ_1 und φ_2 , $t \in \mathbb{N} \cup \{\infty\}$ und $p \in [0, 1] \subseteq \mathbb{R}$, sind $\varphi_1 \text{ U}_{\geq p}^{\leq t} \varphi_2$ und $\varphi_1 \text{ U}_{> p}^{\leq t} \varphi_2$ auch PCTL-Formeln.

Mit diesen Regeln können wir einige PCTL-Formeln aufstellen.

Beispiel 2 (Korrekte und inkorrekte PCTL-Formeln)

Sei $\text{AP} = \{A, B, X, Y\}$. Dann wären

$$\neg(X \wedge \neg(\top \text{ U}_{\geq 90\%}^{\leq 15} Y)) \text{ und } (A \text{ U}_{\geq 90\%}^{\leq 10} B) \wedge (A \text{ U}_{> 95\%}^{\leq 20} B)$$

korrekte PCTL-Formeln.

Inkorrekt gebildete Formeln wären zum Beispiel

$$\neg(X \wedge \neg(\top \text{ U}_{\geq 90\%}^{\leq 15})) \text{ und } (A \text{ U}_{\geq 90\%}^{\leq 10} B)(A \text{ U}_{> 95\%}^{\leq 20} B).$$

Ähnlich, wie wir Transitionssysteme definiert haben, um diese als Modelle von CTL-Formeln zu verwenden, wollen wir nun sogenannte Markov-Ketten definieren, um Eigenschaften von diesen mithilfe von PCTL zu formulieren.

Definition 3.2 (Markov-Ketten)

Sei S eine endliche Menge, $s_i \in S$ und $\mathcal{L} : S \rightarrow 2^{\text{AP}}$ sowie $\mathcal{T} : S \times S \rightarrow [0, 1]$ Funktionen so, dass für alle $s \in S$ gilt: $\sum_{s' \in S} \mathcal{T}(s, s') = 1$.

Dann nennen wir $\mathfrak{S} = (S, s_i, \mathcal{T}, \mathcal{L})$ eine Markov-Kette, wobei S eine Menge an Zuständen ist, s_i der Anfangszustand, \mathcal{T} die Transitions-Wahrscheinlichkeits-Funktion und \mathcal{L} die Bezeichnungsfunktion, die jedem Zustand eine Menge an atomaren Aussagen zuweist.

Weiter bezeichnen wir mit $\text{paths}_{s_0}^{\mathfrak{S}}$ die Menge der Pfade in \mathfrak{S} , die in s_0 beginnen. Ein $\sigma \in \text{paths}_{s_0}^{\mathfrak{S}}$ ist dann von der Form $\sigma = s_0 s_1 s_2 \dots$ und wir definieren $\sigma[n] := s_n$ als den n -ten Zustand des Pfads und $\sigma \upharpoonright n := s_0 \dots s_n$ als den $n + 1$ langen Präfix des Pfads. [HJ94]

Zur Einfachheit sagen wir, dass zwischen vom Knoten s zum Knoten s' genau dann eine Kante existiert, wenn $\mathcal{T}(s, s') \neq 0$. Man erkennt, dass im Allgemeinen ein Pfad $\sigma \in \text{paths}_{s_0}^{\mathfrak{S}}$ unendlich lang ist. Dies ist wohldefiniert, da jeder Zustand eine ausgehende Kante

haben muss. Andernfalls gibt es ein $\hat{s} \in S$ mit $\{s' \in S : \mathcal{T}(\hat{s}, s') = 0\} = S$. Aber dann ist $\sum_{s' \in S} \mathcal{T}(\hat{s}, s') = 0$. Widerspruch! Es gibt für jeden Zustand also mindestens einen Nachfolgezustand, es gibt also immer unendliche Pfade.

Eine Markov Kette ist also ein gerichteter, gewichteter Graph, wobei die Gewichtung der Kanten angibt, wie wahrscheinlich es ist, eine bestimmte Kante auszuwählen. Zusätzlich soll die Summe aller Gewichte der ausgehenden Kanten eines Knotens immer gleich eins sein. Damit ist auch gewährleistet, dass es keine isolierten Knoten gibt.

Im Kontext von Systemen sollen die Zustände des Graphens Zustände des Systems beschreiben. Die Kanten stellen die Möglichen Folgezustände des Systems dar, wobei der Wechsel in einen Folgezustand mit der Wahrscheinlichkeit durchgeführt wird, mit der die Kante annotiert ist. In jedem Zustand gelten atomare Aussagen, welche die Zustände beschreiben, diese werden von der Funktion \mathcal{L} zugewiesen.

Betrachten wir eine Markov-Kette als Beispiel:

Beispiel 3 (Beispiel einer Markov-Kette)

Sei $S = \{s_0, s_1, s_2\}$, $\mathcal{L} = \{s_0 \mapsto \{A\}, s_1 \mapsto \{A, B\}, s_2 \mapsto \{C\}\}$ und \mathcal{T} durch Tabelle 1 definiert. Dann ist $\mathfrak{S} = (S, s_0, \mathcal{T}, \mathcal{L})$ die in Abbildung 1 graphisch dargestellte Markov-Kette, wobei Transitionen mit einer Wahrscheinlichkeit von 0 nicht eingezeichnet werden.

\mathcal{T}	s_0	s_1	s_2
s_0	0	0.7	0.3
s_1	0.8	0	0.2
s_2	0	0	1

Tabelle 1: Tabelle zur Definition der Funktion $\mathcal{T} : S \times S \rightarrow [0, 1]$

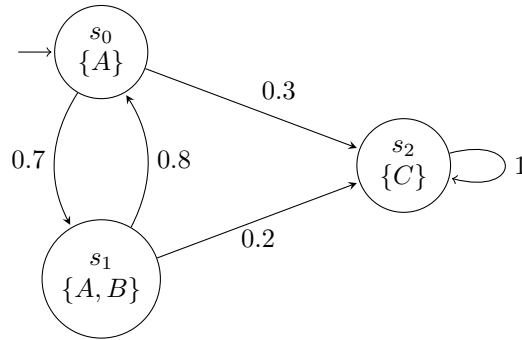


Abbildung 1: Graph für die Markov-Kette \mathfrak{S}

Bevor wir die Semantik von PCTL-Formeln für Markov-Ketten formal definieren können benötigen wir noch den Begriff des Wahrscheinlichkeitsmaßes.

Definition 3.3 (Wahrscheinlichkeitsmaß)

Sei $\pi = s_0 \dots s_n$ eine Folge an Zuständen einer Markov-Kette \mathfrak{S} und $X = \{\sigma \in \text{paths}_{s_0}^{\mathfrak{S}} : \sigma \uparrow n = \pi\}$ die Menge aller (unendlichen) Pfade, die mit dieser Folge beginnen. Wir definieren dann das Wahrscheinlichkeitsmaß

$$\mu_{s_0}^{\mathfrak{S}}(X) := \mathcal{T}(s_0, s_1) \cdot \dots \cdot \mathcal{T}(s_{n-1}, s_n),$$

als das Produkt der Kanten zwischen den Zuständen der Folge. [HJ94]

Weiter ergeben sich einige Gleichheiten [HJ94]:

- Wählt man eine leere Folge mit Länge ergibt sich

$$\mu_{s_0}^{\mathfrak{S}}(\{\sigma \in \text{paths}_{s_0}^{\mathfrak{S}} : \sigma \uparrow 0 = s_0\}) = 1. \quad (1)$$

- Für eine abzählbare Menge $(X_i)_{i \in I}$ mit paarweise disjunkten Mengen an Pfaden gilt

$$\mu_{s_0}^{\mathfrak{S}}\left(\bigcup_{i \in I} X_i\right) = \sum_{i \in I} \mu_{s_0}^{\mathfrak{S}}(X_i). \quad (2)$$

- Sei $X \subseteq \text{paths}_{s_0}^{\mathfrak{S}}$ dann folgt

$$\mu_{s_0}^{\mathfrak{S}}(\text{paths}_{s_0}^{\mathfrak{S}} \setminus X) = 1 - \mu_{s_0}^{\mathfrak{S}}(X). \quad (3)$$

Betrachten wir zum verdeutlichen ein paar Beispiele.

Beispiel 4 (Wahrscheinlichkeitsmaße)

Der Einfachheit halber verwenden wir wieder die Markov-Kette aus Abbildung 1. Sei $\pi = s_0 s_1 s_0 s_2$. Dann ist $X = \{\sigma \in \text{paths}_{s_1}^{\mathfrak{S}} : \sigma \uparrow 3 = \pi\}$ die Menge der Pfade, die durch folgenden Ausdruck dargestellt werden: $s_0 s_1 s_0 (s_2)^\omega$ und es folgt

$$\mu_{s_0}^{\mathfrak{S}}(X) = \mathcal{T}(s_0, s_1) \cdot \mathcal{T}(s_1, s_0) \cdot \mathcal{T}(s_0, s_2) = 0.7 \cdot 0.8 \cdot 0.3 = 0.168$$

Als zweites Beispiel wollen wir alle Pfade betrachten, die dem Ausdruck $s_0 (s_1 s_0)^* s_2$ entsprechen. Also alle Pfade, die endlich oft zwischen s_0 nach s_1 und wieder zurück wechseln und dann in s_2 enden. Es gilt:

$$\begin{aligned} & \mu_{s_0}^{\mathfrak{S}}(\{\sigma \in \text{paths}_{s_0}^{\mathfrak{S}} : \exists i \in \mathbb{N} (\sigma \uparrow 2i + 1 = s_0 (s_1 s_0)^i s_2)\}) \\ &= \mu_{s_0}^{\mathfrak{S}}\left(\bigcup_{i \in \mathbb{N}} \{\sigma \in \text{paths}_{s_0}^{\mathfrak{S}} : \sigma \uparrow 2i + 1 = s_0 (s_1 s_0)^i s_2\}\right) \\ &= \sum_{i \in \mathbb{N}} \mu_{s_0}^{\mathfrak{S}}(\{\sigma \in \text{paths}_{s_0}^{\mathfrak{S}} : \sigma \uparrow 2i + 1 = s_0 (s_1 s_0)^i s_2\}) \\ &= \sum_{i \in \mathbb{N}} (\mathcal{T}(s_0, s_1) \cdot \mathcal{T}(s_1, s_0))^i \cdot \mathcal{T}(s_0, s_2) \\ &= \sum_{i \in \mathbb{N}} 0.56^i \cdot 0.3 = 0.3 \cdot \sum_{i \in \mathbb{N}} 0.56^i \\ &= 0.3 \cdot \frac{1}{1 - 0.56} \approx 0,6818 \end{aligned}$$

Zuletzt soll ausgerechnet werden wie hoch die Wahrscheinlichkeit ist, niemals s_2 zu erreichen. Dies sind die genau die Pfade, die nur zwischen s_0 und s_1 hin-und-her wechseln. Ein alternativer Weg ist es, das Wahrscheinlichkeitsmaß der Pfade zu berechnen, die endlich von zwischen s_0 und s_1 wechseln und dann von s_1 aus in s_2 enden, also dem Ausdruck $s_0 s_1 (s_0 s_1)^* s_2$ entsprechen. Die Berechnung verläuft analog zu dem eben vorgeführten Beispiel. Das Ergebnis ist dann $0.14 \cdot \frac{1}{1-0.56} \approx 0.3182$. Es fällt auf, dass die Summe der letzten beiden Werte 1 ist. Fasst man beide Berechnungen zusammen erhält man demnach, dass die Wahrscheinlichkeit endlich oft zwischen s_0 und s_1 zu wechseln und irgendwie in s_2 zu enden gleich 1 ist. Mit Gleichung 3 ist die Wahrscheinlichkeit nie in s_2 zu enden also 0.

Nun können wir die Semantik von PCTL definieren. Genauer definieren wir eine Relation $\models_{\mathfrak{S}}$ für eine Markov-Kette $\mathfrak{S} = (S, s_i, \mathcal{T}, \mathcal{L})$, ein $s \in S$ und eine PCTL-Formel φ . Zur Vereinfachung definieren wir noch die Relation $\models_{\mathfrak{S}} \subseteq \text{paths}_{s_0}^{\mathfrak{S}} \times X$, wobei X die Menge der Ausdrücke von der Form $\varphi_1 U^{\leq t} \varphi_2$ mit $\varphi_1, \varphi_2 \in \text{PCTL}$, $t \in \mathbb{N}$ ist.

Definition 3.4 (Semantik von PCTL auf Markov-Ketten)

Sei $\mathfrak{S} = (S, s_i, \mathcal{T}, \mathcal{L})$ eine Markov-Kette und $s_0 \in S$. Dann lässt sich die Semantik für PCTL-Formeln induktiv definieren [HJ94]:

1. $s_0 \models_{\mathfrak{S}} \top \Leftrightarrow \mathfrak{S} \models \forall x(x = x)$ und $s_0 \models_{\mathfrak{S}} \perp \Leftrightarrow \mathfrak{S} \models \exists x(x \neq x)$
2. Für $a \in \text{AP}$ gilt $s_0 \models_{\mathfrak{S}} a \Leftrightarrow a \in \mathcal{L}(s_0)$
3. $s_0 \models_{\mathfrak{S}} \neg \varphi \Leftrightarrow \text{nicht } s_0 \models_{\mathfrak{S}} \varphi$
4. $s \models_{\mathfrak{S}} \varphi_1 \wedge \varphi_2 \Leftrightarrow s \models_{\mathfrak{S}} \varphi_1$ und $s \models_{\mathfrak{S}} \varphi_2$
5. $\sigma \models_{\mathfrak{S}} \varphi_1 U^{\leq t} \varphi_2 \Leftrightarrow \exists i \leq t (\sigma[i] \models_{\mathfrak{S}} \varphi_2 \text{ und } \sigma[j] \models_{\mathfrak{S}} \varphi_1 \text{ für alle } 0 \leq j < i)$
6. $s_0 \models_{\mathfrak{S}} \varphi_1 U_{\leq p}^{\leq t} \varphi_2 \Leftrightarrow \mu_{s_0}^{\mathfrak{S}}(\{\sigma \in \text{paths}_{s_0}^{\mathfrak{S}} : \sigma \models_{\mathfrak{S}} \varphi_1 U^{\leq t} \varphi_2\}) \geq p$
7. $s_0 \models_{\mathfrak{S}} \varphi_1 U_{> p}^{\leq t} \varphi_2 \Leftrightarrow \mu_{s_0}^{\mathfrak{S}}(\{\sigma \in \text{paths}_{s_0}^{\mathfrak{S}} : \sigma \models_{\mathfrak{S}} \varphi_1 U^{\leq t} \varphi_2\}) > p$

Es fällt auf, dass im Operator $U_{\leq p}^{\leq t}$, der Parameter t Zeiteinheiten beschreibt, wobei eine Zeiteinheit genau eine Transition in der Markov-Kette ist. Analog beschreibt p die Wahrscheinlichkeit, dass ein gültiger Pfad „genommen“ wird.

Zusätzlich können wir weitere Operatoren definieren, um Formeln zu vereinfachen. Wie bekannt sind \neg und \wedge funktional vollständig, weshalb wir die bekannten booleschen Operatoren wie folgt definieren können:

- $\varphi_1 \vee \varphi_2 \equiv \neg(\neg \varphi_1 \wedge \neg \varphi_2)$
- $\varphi_1 \rightarrow \varphi_2 \equiv \neg \varphi_1 \vee \varphi_2$.

Zusätzlich definieren wir den *außer* Operator \mathcal{U} [HJ94]:

- $\varphi_1 \mathcal{U}_{\geq p}^{\leq t} \varphi_2 \equiv \neg(\neg\varphi_1 \mathcal{U}_{\geq 1-p}^{\leq t} \neg(\varphi_1 \vee \varphi_2))$
- $\varphi_1 \mathcal{U}_{> p}^{\leq t} \varphi_2 \equiv \neg(\neg\varphi_1 \mathcal{U}_{\geq 1-p}^{\leq t} \neg(\varphi_1 \vee \varphi_2)).$

Intuitiv bedeutet $\varphi_1 \mathcal{U}_{\geq p}^{\leq t} \varphi_2$, dass mit einer Wahrscheinlichkeit von mindestens p entweder φ_1 für t Zeiteinheiten gilt, oder innerhalb von t Zeiteinheiten φ_2 erfüllt wird und bis dahin φ_1 gilt. Die Bedeutung von $\varphi_1 \mathcal{U}_{> p}^{\leq t} \varphi_2$ ist analog. [HJ94]

Bevor wir nun Beispiele betrachten wollen, soll noch ein Vergleich mit CTL stattfinden, welcher uns zusätzliche, sehr nützliche Operatoren liefern wird. Wie in Kapitel 2 erläutert wurde, besitzt CTL weder Aussagen über Zeit, noch Wahrscheinlichkeiten. Demnach können wir a priori einschränken, dass $t = \infty$ gelten muss und $p \in \{0, 1\}$, abhängig davon, ob ein A oder E verwendet wird. Man kann folgende Äquivalenzen finden [HJ94]:

- $A[\varphi_1 \mathcal{U} \varphi_2] \equiv \varphi_1 \mathcal{U}_{\geq 1}^{\leq \infty} \varphi_2$
- $E[\varphi_1 \mathcal{U} \varphi_2] \equiv \varphi_1 \mathcal{U}_{> 0}^{\leq \infty} \varphi_2$
- $AF \varphi \equiv \top \mathcal{U}_{\geq 1}^{\leq \infty} \varphi$
- $EF \varphi \equiv \top \mathcal{U}_{> 0}^{\leq \infty} \varphi$
- $AG \varphi \equiv \varphi \mathcal{U}_{\geq 1}^{\leq \infty} \perp$
- $EG \varphi \equiv \varphi \mathcal{U}_{> 0}^{\leq \infty} \perp$

Intuitiv haben die Operatoren die aus Kapitel 2 bekannten Bedeutungen. Durch die Fähigkeiten von PCTL lassen sich die vier unären Operatoren verallgemeinern. Betrachtet man die Formeln $G_{\geq p}^{\leq t} \varphi \equiv \varphi \mathcal{U}_{\geq p}^{\leq t} \perp$ und $F_{\geq p}^{\leq t} \varphi \equiv \top \mathcal{U}_{\geq p}^{\leq t} \varphi$ fällt einem auf, dass diese eben die Bedeutung von EG / AG und EF / AF für arbiträre $t \in \mathbb{N} \cup \{\infty\}$ und $p \in [0, 1]$ ausweiten. $G_{\geq p}^{\leq t} \varphi$ gilt also genau dann, wenn die Wahrscheinlichkeit, dass φ für t Zeiteinheiten mindestens p ist und $F_{\geq p}^{\leq t} \varphi$ bedeutet, dass mit einer Wahrscheinlichkeit von p , φ innerhalb von t Zeiteinheiten gelten wird. [HJ94]

In der Beschreibung von Systemen sind bedingte Aussagen wie zum Beispiel „Wenn A passiert, wird B passieren“ sehr hilfreich. Daher soll ein Operator, der solch ein Verhalten, verbunden mit Zeit- und Wahrscheinlichkeitsparametern, für PCTL definiert werden:

$$\varphi_1 \rightsquigarrow_{\geq p}^{\leq t} \varphi_2 \equiv AG(\varphi_1 \rightarrow F_{\geq p}^{\leq t} \varphi_2),$$

mit der intuitiven Bedeutung, dass wenn φ_1 gilt, dann wird mit einer Wahrscheinlichkeit von mindestens p , in t Schritten φ_2 erfüllt. [HJ94]

Nun können wir Ausdrücke formulieren, die interessante Eigenschaften von Systemen beschreiben. Beginnen wir zuerst mit einigen abstrakten Beispielen.

Beispiel 5 (Beispiele für PCTL-Formeln und deren Syntax)

Betrachte die Formeln aus Beispiel 2. Die erste ist $\varphi = \neg(X \wedge \neg(\top \mathcal{U}_{\geq 90\%}^{\leq 15} Y))$. Es fällt

schnell auf, dass $\neg(X \wedge \neg(\top U_{\geq 90\%}^{\leq 15} Y)) \equiv X \rightarrow (\top U_{\geq 90\%}^{\leq 15} Y) \equiv X \rightarrow F_{\geq 90\%}^{\leq 15} Y$. Intuitiv bedeutet $s \models_{\mathfrak{S}} \varphi$ also, dass wenn $s \models_{\mathfrak{S}} X$, dann folgt, dass innerhalb von 15 Zeiteinheiten mit einer Wahrscheinlichkeit von 90%, Y gelten wird.

Die zweite Formel aus Beispiel 2 ist $\psi = (A U_{\geq 90\%}^{\leq 10} B) \wedge (A U_{> 95\%}^{\leq 20} B)$. Die Bedeutung ist intuitiv hier, dass mit einer Wahrscheinlichkeit von mindestens 90% A für 10 Schritte und dann B gilt und mit einer Wahrscheinlichkeit von mehr als 95% gilt A für 20 Schritte und danach B .

Für ein paar weitere Beispiele und Erkenntnisse soll die Markov-Kette $\mathfrak{S} = (\{s_0, s_1, s_2\}, s_0, \mathcal{T}, \mathcal{L})$ aus Abbildung 1 betrachtet werden. Um Wahrscheinlichkeitsmaße zu erläutern, wurden zwei interessante Maße berechnet. Zum Einen die Wahrscheinlichkeit, dass von s_0 irgendwann zu s_2 gewechselt wird, und die analoge Wahrscheinlichkeit für den Wechsel zu s_2 von s_1 aus. Es fällt auf, dass die Summe der Wahrscheinlichkeiten 1, die Wahrscheinlichkeit, nie s_2 zu erreichen, ist also 0. Es folgt also $s_0 \models_{\mathfrak{S}} \neg EG A$ bzw. $s_0 \models_{\mathfrak{S}} A \rightsquigarrow_1^{\leq \infty} C$.

Da wir nun vertraut mit der Funktionsweise von PCTL sind, wollen wir das algorithmische Lösen des Model-Checking Spiels für PCTL-Formeln und Markov-Ketten betrachten.

3.2 Das Model-Checking Spiel für PCTL

Ein wichtiges Anwendungsgebiet für temporale Logiken wie CTL und PCTL ist die Verifikation von Systemen. Bei diesem erhält man eine Struktur \mathfrak{S} (im Fall von PCTL eine Markov-Kette) und eine PCTL-Formel φ , welche das gewünschte Verhalten eines Systems beschreibt. Nun überprüft man ob $\mathfrak{S} \models \varphi$ gilt und falls dies der Fall ist, wurde somit gezeigt, dass das durch \mathfrak{S} modellierte System sich korrekt verhält. Da komplexe Systeme aber nur durch sehr große Strukturen modelliert werden können und sich kompliziertes Verhalten auch nicht mit kurzen Formeln dargestellt werden kann, ist es sinnvoll dieses Modellierungsproblem zu automatisieren, indem man Model Checking Algorithmen entwickelt. Die Algorithmen aus [HJ94] sollen demnach in diesem Kapitel vorgestellt werden.

Sei $\mathfrak{S} = (S, s_i, \mathcal{T}, \mathcal{L})$ also eine Markov-Kette und φ eine PCTL-Formel. Der Algorithmus wird induktiv über den Formelaufbau definiert, indem für jede Formel zuerst ihre Subformeln ausgewertet werden. Dafür definieren wir eine Funktion $\text{label} : S \rightarrow 2^{\text{PCTL}}$, welche jedem Zustand aus \mathfrak{S} eine Menge an Formeln zuweist, die in diesem Zustand gelten. Am Anfang des Algorithmus wird $\text{label}(s) := \mathcal{L}(s) \cup \{\top\}$ für alle $s \in S$ gesetzt. Damit haben wir den Induktionsanfang bestimmt. Sei φ nun ein PCTL-Formel, die entweder der Regel 3 oder Regel 4 aus Definition 3.1 entspricht.

- Falls $\varphi = \neg\psi$ für eine PCTL-Formel ψ , dann können wir nach Induktionsvoraussetzung annehmen, dass ψ bereits für \mathfrak{S} ausgewertet wurde. Wir legen für alle $s \in S$ fest, dass $\text{label}(s) := \text{label}(s) \cup \{\varphi\}$, wenn $\psi \notin \text{label}(s)$.
- Falls $\varphi = \varphi_1 \wedge \varphi_2$, dann aktualisieren wir $\text{label}(s) := \text{label}(s) \cup \{\varphi\}$, wenn $\varphi_1 \in \text{label}(s)$ und $\varphi_2 \in \text{label}(s)$.

- Die Fälle $\varphi = \varphi_1 U_{\geq p}^{\leq t} \varphi_2$ und $\varphi = \varphi_1 U_{> p}^{\leq t} \varphi_2$ sind sehr viel komplexer und werden daher explizit im nächsten Abschnitt betrachtet.

Da nach Definition 3.1 zu jeder PCTL-Formel eine äquivalente Formel gebildet werden kann, die nur die obigen Operatoren verwendet, fehlt nur noch zu zeigen, wann $\text{label}(s)$ angepasst wird, im Fall der temporalen Operatoren. [HJ94]

Zur Vereinfachung teilen wir den Fall der temporalen Operatoren in drei Fälle auf, abhängig vom Parameter t . Die drei Fälle sind $t = 0$, $0 < t < \infty$ und $t = \infty$. Der Fall $t = 0$ ist trivial, da sich dann $s \models \varphi_1 U_{\geq p}^{\leq 0} \varphi_2$ offensichtlich zu $s \models \varphi_2$ vereinfachen lässt (und analog für $U_{> p}^{\leq 0}$). Die anderen beiden Fälle sollen nun aber weiter betrachtet werden, beginnend mit $0 < t < \infty$.

Sei $\varphi := \varphi_1 U_{\geq p}^{\leq t} \varphi_2$, $0 < t < \infty$ und die Zustände in denen φ_1 und φ_2 gelten wurden bereits bestimmt. Betrachtet man die Definition der Semantik für $U_{\geq p}^{\leq t}$ fällt einem auf, dass eine gesonderte Relation für Pfade und Formeln der Form $\varphi_1 U^{\leq t} \varphi_2$ definiert wurde. Daher definieren wir auch hier eine neue Funktion. Sei $\mathcal{P} : \mathbb{N} \times S \rightarrow [0, 1]$. Dann ist $\mathcal{P}(t, s)$ das Wahrscheinlichkeitsmaß für die Menge der Pfade σ , die in s beginnen und für die $\sigma \models \varphi_1 U^{\leq t} \varphi_2$ gilt. Wie in [HJ94] bewiesen erfüllt \mathcal{P} folgende Rekursionsgleichung:

$$\mathcal{P}(t, s) = \begin{cases} 1 & \text{wenn } \varphi_2 \in \text{label}(s) \\ 0 & \text{wenn } \varphi_1 \notin \text{label}(s) \\ \sum_{s' \in S} \mathcal{T}(s, s') \cdot \mathcal{P}(t-1, s') & \text{sonst} \end{cases} \quad (4)$$

wobei $t > 0$. Für $t = 0$ definieren $\mathcal{P}(0, s)$ wie folgt:

$$\mathcal{P}(0, s) = \begin{cases} 1 & \text{wenn } \varphi_2 \in \text{label}(s) \\ 0 & \text{sonst} \end{cases} \quad (5)$$

Dies liefert unmittelbar Algorithmus 1 zum Berechnen von \mathcal{P} .

Ein anderer Ansatz zum berechnen von \mathcal{P} lässt sich mithilfe von Matrizen finden. Dafür partitionieren wir die Zustandsmenge $S = \{s_1, \dots, s_n\}$ in drei Teilmengen S_s , S_f und S_i auf mit den Bedeutungen, dass in S_s die Erfolgszustände sind, anfangs also die $s \in S$ mit $\varphi_2 \in \text{label}(s)$, S_f sind die abgelehnten Zustände, also die $s \in S$ mit $\varphi_1, \varphi_2 \notin \text{label}(s)$. S_i sind die restlichen Zustände über die noch keine Aussage getroffen werden kann, genauer also die $s \in S$ mit $\varphi_1 \in \text{label}(s)$ und $\varphi_2 \notin \text{label}(s)$. [HJ94]

Nun definieren wir eine $|S| \times |S|$ -Matrix M

$$M[s_k, s_l] = \begin{cases} \mathcal{T}(s_k, s_l) & \text{wenn } s_k \in S_i \\ 1 & \text{wenn } s_k \notin S_i \wedge k = l \\ 0 & \text{sonst} \end{cases}$$

und für jedes $t \in \mathbb{N}$ einen Spaltenvektor $\overline{\mathcal{P}}(t)$ der Größe $|S|$ so, dass für das i -te Element $\overline{\mathcal{P}}(t)_i = \mathcal{P}(t, s_i)$ gilt. Für $t = 0$ gilt insbesondere also, dass $\overline{\mathcal{P}}(0)_i = 1$ wenn $s_i \in S_s$ und 0 sonst. Nach dem Beweis in [HJ94] gilt dann, dass

$$\overline{\mathcal{P}}(t) = \overbrace{M \times M \times \dots \times M}^t \times \overline{\mathcal{P}}(0) = M^t \times \overline{\mathcal{P}}(0). \quad (6)$$

Algorithmus 1 Übersetzung der Rekursionsgleichung 4 in Pseudo-Code [HJ94]

```
for  $i = 0, \dots, t$  do
  for all  $s \in S$  do
    if  $\varphi_2 \in \text{label}(s)$  then
       $\mathcal{P}(i, s) \leftarrow 1$ 
    else if  $\varphi_1 \notin \text{label}(s)$  then
       $\mathcal{P}(i, s) \leftarrow 0$ 
    else
       $\mathcal{P}(i, s) \leftarrow 0$ 
      if  $i > 1$  then
        for all  $s' \in S$  do
           $\mathcal{P}(i, s) \leftarrow \mathcal{P}(i, s) + \mathcal{T}(s, s') \cdot \mathcal{P}(i-1, s')$ 
        end for
      end if
    end if
  end for
end for
```

Diese Gleichung liefert direkt Algorithmus 2 zum Berechnen von \mathcal{P} . [HJ94]

Algorithmus 2 Algorithmus zum Berechnen von \mathcal{P} mithilfe der Gleichung 6 [HJ94]

```
for all  $s_i \in S$  do
  if  $\varphi_2 \in \text{label}(s)$  then
     $\overline{\mathcal{P}}(0)_i \leftarrow 1$ 
  else
     $\overline{\mathcal{P}}(0)_i \leftarrow 0$ 
  end if
end for
 $\overline{\mathcal{P}}(t) = M^t \times \overline{\mathcal{P}}(0)$ 
```

Damit haben wir nun zwei Algorithmen für das Berechnen von Wahrscheinlichkeiten in Bezug auf $U_{\geq p}^{\leq t}$ (bzw. $U_{> p}^{\leq t}$). Betrachten wir arithmetische Operationen fällt auf, dass Algorithmus 1 eine Zeitkomplexität von $\mathcal{O}(t \cdot |S|^2)$ bzw. falls $|E|$ die Anzahl der Transitionen in \mathcal{T} mit einem Wert größer 0 sind von $\mathcal{O}(t \cdot (|S| + |E|))$. Da zum Berechnen von M^t insgesamt $\log t$ Matrixmultiplikationen benötigt werden, welche jeweils in $\mathcal{O}(|S|^3)$ sind, ergibt sich für Algorithmus 2 eine Zeitkomplexität von $\mathcal{O}(\log t \cdot |S|^3)$. Es folgt, dass bei großen Werten für t , Algorithmus 2 sehr viel effizienter ist, während bei kleinen t aber sehr vielen Zuständen 1 schneller ist. [HJ94]

Falls $t = \infty$ ergeben sich für beide bisher vorgestellten Algorithmen einige Probleme, da Algorithmus 1 nicht terminieren würde und M^t im Algorithmus 2 nicht definiert ist. Daher müssen wir die Algorithmen anpassen. Betrachten wir die Partitionierung von S , welche wir zur Definition der Matrix M verwendet haben. Sei nun R die Menge der Erfolgzustände, also die, in denen φ_2 gilt. Um die abgelehnten Zustände zu identifizieren

reicht es nun nicht mehr nur die betrachten, in denen φ_1 nicht gilt, da ein noch unentschiedener Zustand in S_i , von dem kein Erfolgszustand erreichbar ist auch abgelehnt werden muss. Dies liefert uns die Definition der Menge Q als die Knoten, in denen φ_1 nicht gilt oder denen, in denen φ_1 gilt, φ_2 nicht gilt und von denen kein Knoten in R erreichbar ist. Die restlichen Knoten, in denen φ_1 gilt, φ_2 nicht gilt und von denen ein Knoten in R erreichbar ist befinden sich wieder in der Menge S . [HJ94]

Da R einfach zu bestimmen ist und auch S_i direkt aus den Definitionen von R und Q folgt benötigen wir einen Algorithmus zum identifizieren von Q . Algorithmus 3 erreicht genau das, wobei er die Knotenmengen S_s und S_i aus Algorithmus 2 verwendet.

Algorithmus 3 Algorithmus zum Bestimmen von Q [HJ94]

```

ungesehen  $\leftarrow S_i \cup S_s$ 
rand  $\leftarrow S_s$ 
markiert  $\leftarrow \emptyset$ 
for  $i = 0, \dots, |S_i|$  do
    markiert  $\leftarrow$  markiert  $\cup$  rand
    ungesehen  $\leftarrow$  ungesehen  $\setminus$  rand
    rand  $\leftarrow \{s \mid s \in \text{ungesehen} \wedge \exists s' \in \text{rand} : \mathcal{T}(s, s') > 0\}$ 
end for
 $Q := S \setminus \text{markiert}$ 

```

Der Algorithmus markiert alle Knoten, von denen ein Knoten in S_s erreichbar ist, wobei zu Beginn des i -ten Durchlaufs der Schleife in der Menge rand genau die Knoten sind, deren kürzester Pfad zu einem Knoten in S_s die Länge i hat. Die Menge Q ist dann genau das Komplement. Analog lässt sich ein Algorithmus aufstellen um R zu der Menge zu erweitern, die nicht nur die Knoten enthält die φ_2 erfüllen, sondern auch die, dessen Wahrscheinlichkeitsmaß, einen Knoten in S_s zu erreichen ohne über einen Knoten in S_f zu gehen gleich 1 ist. Nun können wir, ähnlich zur Gleichung 4, eine Rekursionsgleichung angeben:

$$\mathcal{P}(\infty, s) = \begin{cases} 1 & \text{wenn } s \in R \\ 0 & \text{wenn } s \in Q \\ \sum_{s' \in S} \mathcal{T}(s, s') \cdot \mathcal{P}(\infty, s') & \text{sonst} \end{cases} \quad (7)$$

Betrachtet man $\mathcal{P}(\infty, s)$ bildet sich ein Gleichungssystem mit $|S|$ unbekannten. Dieses lässt sich mithilfe des Gaußschen Eliminierungsverfahren mit einem Zeitaufwand von $\mathcal{O}((|S| - |Q| - |R|)^2 \cdot 81)$ berechnen. [HJ94]

Somit haben wir für die drei Fälle ein Verfahren, um die Modellbeziehung zu berechnen. Sei $\varphi = \varphi_1 \cup_{\geq p}^{\leq t} \varphi_2$ (bzw. $\varphi = \varphi_1 \cup_{> p}^{\leq t} \varphi_2$) und φ_1 , sowie φ_2 wurden bereits betrachtet. Nun befinden wir uns in einem der drei Fälle. Gehe für jedes $s \in S$ diese durch und aktualisiere $\text{label}(s)$:

- Falls $t = 0$: Setze $\text{label}(s) = \text{label}(s) \cup \{\varphi\}$, wenn $\varphi_2 \in \text{label}(s)$.
- Falls $0 < t < \infty$: Sei $s = s_i$ für ein $i \in \mathbb{N}$. Berechne mithilfe von Algorithmus 1 die Funktion \mathcal{P} oder mithilfe von Algorithmus 2 den Spaltenvektor $\overline{\mathcal{P}}(t)$. Setze

$\text{label}(s) = \text{label}(s) \cup \{\varphi\}$, wenn $\mathcal{P}(t, s) = \overline{\mathcal{P}}(t)_i \geq p$ (bzw. $\mathcal{P}(t, s) = \overline{\mathcal{P}}(t)_i > p$).

- Falls $t = \infty$: Löse das durch $\{\mathcal{P}(\infty, s) : s \in S\}$ definierte Gleichungssystem und setze $\text{label}(s) = \text{label}(s) \cup \{\varphi\}$, wenn $\mathcal{P}(\infty, s) \geq p$ (bzw. $\mathcal{P}(\infty, s) > p$).

Somit haben wir die Induktion vom Anfang des Kapitels beendet und wir sind in der Lage, für beliebige PCTL-Formeln φ , welche die elementaren Operatoren verwenden, in polynomieller Zeit zu überprüfen, ob für ein gegebenes s aus einer Markov-Kette \mathfrak{S} $s \models_{\mathfrak{S}} \varphi$ gilt.

Es lassen sich aber noch einige Verbesserungen finden. Zum Einen haben wir die erweiterten Operatoren wie $\mathcal{U}_{\geq p}^{\leq t}$, AF, EG usw. nicht betrachtet, da sich diese ja auf die behandelten, grundlegenden Operatoren vereinfachen lassen. Vor allem im Fall von $\mathcal{U}_{\geq p}^{\leq t}$ und $\mathcal{U}_{> p}^{\leq t}$ erhöht dies aber die Länge der Formel drastisch, weshalb in [HJ94] für diese Fälle weitere Algorithmen gefunden werden können.

Auch gibt es noch weitere Sonderfälle für die Operatoren $\mathcal{U}_{\geq p}^{\leq t}$ und $\mathcal{U}_{> p}^{\leq t}$. Ist $p \in \{0, 1\}$, dann gibt es deutlich effizientere Algorithmen als die hier vorgestellten. Für endliche t muss entweder ein Pfad mit Länge höchstens t gefunden werden, der eben die *Until*-Eigenschaften erfüllt oder einer, der diese nicht erfüllt, diese lassen sich ebenfalls in [HJ94] finden. Für $t = \infty$ lassen sich normale CTL Model-Checking Algorithmen verwenden. Beispiele dafür gibt es in [BK08] oder [CE82].

3.3 Model-Checking Beispiel

Sei $\text{AP} = \{\text{running}, \text{stopped}, \text{warning}, \text{error}\}$. Weiter definieren wir die Markov-Kette $\mathfrak{S} = (S = \{s_0, s_1, s_2\}, s_0, \mathcal{T}, \mathcal{L})$, wobei \mathcal{T} die durch Tabelle 2 definierte Funktion ist und $\mathcal{L} = \{s_0 \mapsto \{\text{running}\}, s_1 \mapsto \{\text{stopped}, \text{warning}\}, s_2 \mapsto \{\text{stopped}, \text{error}\}\}$ gilt.

\mathcal{T}	s_0	s_1	s_2
s_0	0.95	0	0.05
s_1	0.4	0.5	0.1
s_2	0	0.4	0.6

Tabelle 2: Tabelle zur Definition der Funktion $\mathcal{T} : S \times S \rightarrow [0, 1]$ für das Model-Checking Beispiel

Dadurch ergibt sich dann der Graph aus Abbildung 2.

Die Markov-Kette beschreibt ein beliebiges System. Beim Auftreten eines Fehlers, was in jedem Zeitschritt mit einer Wahrscheinlichkeit von 5% passieren kann, wird in den Zustand s_2 gewechselt, welcher den Fehlerzustand darstellt, und das System stoppt den Betrieb. Mit einer Wahrscheinlichkeit von 40% wird das System repariert, wodurch es in den Zustand s_1 wechselt. In diesem hat das System noch immer nicht seinen Betrieb wiederaufgenommen, dies passiert erst, wenn der Fehler quitiert wurde, was jeden Zeitschritt mit einer Wahrscheinlichkeit von 40% passieren kann.

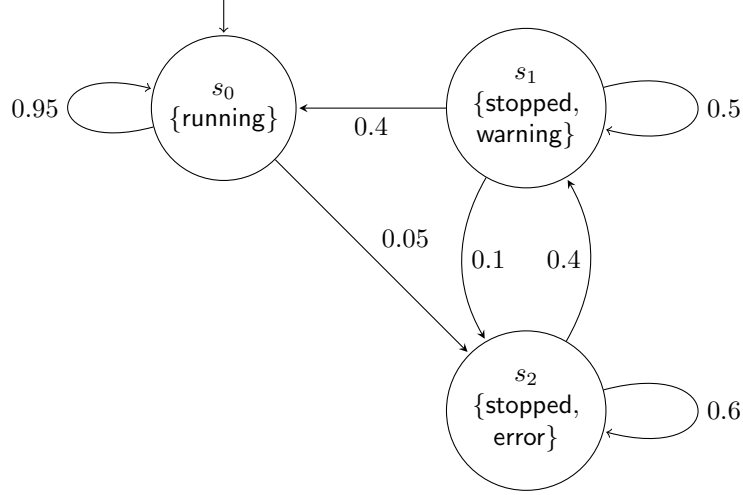


Abbildung 2: Graph für die Markov-Kette \mathfrak{S} des Model-Checking Beispiels

Betrachten wir zu Beginn die Formel $\varphi = F_{\geq 60\%}^{\leq 2} \text{running}$. Mithilfe der Model-Checking Algorithmen aus Kapitel 3.2 wollen wir nun überprüfen, von welchen Zuständen aus \mathfrak{S} φ erfüllt wird. Nach den Definitionen aus Kapitel 3 gilt $\varphi \equiv \top \cup_{\geq 60\%}^{\leq 2} \text{running}$. Offensichtlich gilt $\top \in \text{label}(s)$ für alle $s \in S$ und $\text{running} \in \text{label}(s_0)$. Benutzen wir zuerst Algorithmus 1. Wir erhalten dadurch die Funktion, die in Tabelle 3 gesehen werden kann.

$\mathcal{P}(i, s)$	$s = s_0$	$s = s_1$	$s = s_2$
$i = 0$	1	0	0
$i = 1$	1	0.4	0
$i = 2$	1	$0.4 \cdot 1 + 0.5 \cdot 0.4 + 0.1 \cdot 0 = 0.6$	0.16

Tabelle 3: Tabelle für die induktive Definition der Funktion $\mathcal{P} : \mathbb{N} \times S \rightarrow [0, 1]$

Damit erhalten wir also, dass $s_0 \models_{\mathfrak{S}} \varphi$ und $s_1 \models_{\mathfrak{S}} \varphi$ gelten, da $P(2, s_0) > P(2, s_1) \geq 0.6$.

Selbiges wollen wir nun auch mit Algorithmus 2 berechnen. Es gilt $\overline{\mathcal{P}}(0) = (1, 0, 0)^T$, da $\text{running} \in \text{label}(s_0)$, aber $\text{running} \notin \text{label}(s_1)$ bzw. $\text{running} \notin \text{label}(s_2)$. Nun müssen wir die Matrix M definieren. Es gilt

$$M := \begin{pmatrix} 1 & 0 & 0 \\ 0.4 & 0.5 & 0.1 \\ 0 & 0.4 & 0.6 \end{pmatrix}$$

und dann $M^2 \cdot \overline{\mathcal{P}}(0) = \overline{\mathcal{P}}(2) = (1, 0.6, 0.16)^T$. Wir erhalten also die gleichen Werte wie mit Algorithmus 1.

Als zweites Beispiel wollen wir die Formel $\psi = \text{EG running}$ betrachten. Wieder formulieren wir eine äquivalente Formel mit den Operatoren, für die wir Model-Checking Algorithmen definiert haben. Es gilt $\psi \equiv \text{running } \mathcal{U}_{>0}^{\leq \infty} \perp \equiv \neg(\neg\text{running } \mathcal{U}_{\geq 1}^{\leq \infty} \neg\text{running})$. Definiere θ so, dass $\neg(\neg\text{running } \mathcal{U}_{\geq 1}^{\leq \infty} \neg\text{running}) = \neg(\theta)$ und werte zuerst θ für alle $s \in S$ aus. Die einzige Subformel in θ ist $\neg\text{running}$ und es gilt offensichtlich $\neg\text{running} \in \text{label}(s)$ genau dann, wenn $s \neq s_0$. Offensichtlich gilt $R = \{s_1, s_2\}$ und $s_0 \notin Q$. Damit erhalten wir das Gleichungssystem

$$\begin{aligned}\mathcal{P}(\infty, s_0) &= \mathcal{T}(s_0, s_0) \cdot \mathcal{P}(\infty, s_0) + \mathcal{T}(s_0, s_1) \cdot \mathcal{P}(\infty, s_1) + \mathcal{T}(s_0, s_2) \cdot \mathcal{P}(\infty, s_2) \\ &= 0.95 \cdot \mathcal{P}(\infty, s_0) + 0 \cdot \mathcal{P}(\infty, s_1) + 0.05 \cdot \mathcal{P}(\infty, s_2) \\ \mathcal{P}(\infty, s_1) &= 1 \\ \mathcal{P}(\infty, s_2) &= 1\end{aligned}$$

bzw. in Matrixschreibweise

$$\begin{aligned}&\left(\begin{array}{ccc|c} 0.95 \cdot \mathcal{P}(\infty, s_0) & 0 & 0.05 \cdot \mathcal{P}(\infty, s_2) & \mathcal{P}(\infty, s_0) \\ 0 & 1 \cdot \mathcal{P}(\infty, s_1) & 0 & 1 \\ 0 & 0 & 1 \cdot \mathcal{P}(\infty, s_2) & 1 \end{array} \right) \\ &\rightsquigarrow \left(\begin{array}{ccc|c} -0.05 \cdot \mathcal{P}(\infty, s_0) & 0 & 0.05 \cdot \mathcal{P}(\infty, s_2) & 0 \\ 0 & 1 \cdot \mathcal{P}(\infty, s_1) & 0 & 1 \\ 0 & 0 & 1 \cdot \mathcal{P}(\infty, s_2) & 1 \end{array} \right) \\ &\rightsquigarrow \begin{pmatrix} -0.05 & 0 & 0.05 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathcal{P}(\infty, s_0) \\ \mathcal{P}(\infty, s_1) \\ \mathcal{P}(\infty, s_2) \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}.\end{aligned}$$

Löst man dieses Gleichungssystem erhält man $\mathcal{P}(\infty, s_0) = \mathcal{P}(\infty, s_1) = \mathcal{P}(\infty, s_2) = 1$. Und es ergibt sich, dass $s \models_{\mathcal{S}} \theta$ für jedes $s \in S$ gilt. Nach Definition von θ folgt also, dass $s \models \psi$ für alle $s \in S$, insbesondere $s_0 \models \psi$. Damit haben wir gezeigt, dass es keinen Pfad gibt, der immer nur in s_0 bleibt. Über eine unbeschränkte Laufzeit ist es also unmöglich, dass das System immer läuft.

Mit diesen beiden Beispielen wurden die drei Algorithmen aus Kapitel 3.2 genauer erklärt und besprochen. Insgesamt ist nun also bekannt was PCTL ist, wofür es verwendet werden kann, die PCTL-Formeln gebildet werden und was diese bedeuten und wie man algorithmisch auswertet, ob eine gegebene Markov-Kette ein Modell einer PCTL-Formel ist. Im Rest dieser Arbeit soll PCTL mit anderen Logiken verglichen werden, um zu betrachten, wie sinnvoll und zielführend der in [HJ94] verwendete Ansatz ist.

4 Vergleich mit anderen Logiken

Da nun die Logik PCTL bekannt ist sollen einige alternative Ansätze bzw. Logiken diskutiert werden, die entweder zeitliche Aspekte oder Wahrscheinlichkeiten hinzufügen. Dafür sollen zwei Logiken genauer betrachtet werden. Eine erweitert CTL durch ein komplexes System welches reelle Zeitwerte erlaubt und so sehr viel mehr über Zeit aussagen kann als

PCTL. Die andere Logik stellt eine Erweiterung durch Wahrscheinlichkeiten dar. Genauer wird eine Erweiterung des modalen μ -Kalküls betrachtet, welche es ermöglicht, ähnlich wie in PCTL auszusagen, dass eine Formel für mindestens bzw. mehr als $p\%$ der Fälle gilt.

4.1 Timed Computation Tree Logic

In diesem Kapitel wird die in [ACD90] definierte Logik *Timed Computation Tree Logic* (PCTL) besprochen. Dafür werden die dazugehörigen Strukturen, ihre Syntax sowie ihre Semantik erörtert und durch Beispiele dargestellt.

Die Logik TCTL stellt, ähnlich wie PCTL, eine Möglichkeit zur Verfügung, um zeitliche Zusammenhänge auszudrücken. Während dies in PCTL aber nur mit diskreten Werten möglich ist, verwendet TCTL reelle Werte, wodurch sich Systeme potentiell besser beschreiben lassen. Da dies aber offensichtlich komplexer, als bloß bei jeder verwendeten Transition einen Zähler zu erhöhen, wird eine neue Struktur benötigt, welche Systeme modellieren soll und über welchen TCTL-Formeln ausgewertet werden. Dafür definieren wir eine feste Menge \mathcal{N} , welche uns Werte für zeitliche Vergleiche in Formeln gibt. Der Einfachheit halber definieren wir hier $\mathcal{N} := \{0, 1, \dots\} = \mathbb{N}$. Da es aber offensichtlich eine Bijektion zwischen \mathbb{N} und \mathbb{Q} gibt, lässt sich auch $\mathcal{N} := \mathbb{Q}$ festlegen, wodurch sich die Vorteile einer dichten Ordnung¹ im Vergleich zu PCTL ausnutzen lassen. [ACD90]

Definition 4.1 (Zeitliche Graphen)

Ein *zeitlicher Graph* ist ein Tupel $\mathfrak{G} = (S, s_0, E, C, \pi, \tau, \mu)$ mit folgenden Definitionen [ACD90]:

- S ist eine endliche Menge an Knoten.
- $s_0 \in S$ ist der Startzustand.
- $E \subseteq S \times S$ ist die Kantenrelation.
- C ist eine endliche Menge an Uhren.
- $\pi : E \rightarrow 2^C$ ist eine Abbildung, die jeder Kante eine Menge an Uhren zuweist.
- τ ist eine Abbildung, die jeder Kante eine Formel zuweist, die aus Booleschen Junktoren über den atomaren Formeln $x \leq c$ und $c \leq x$ besteht, wobei $c \in C$ und $x \in \mathcal{N}$ gilt.
- Für eine Menge an atomaren Aussagen AP ist $\mu : S \rightarrow 2^{AP}$ eine Abbildung, die jedem Zustand eine Menge an atomaren Aussagen zuweist.

Ein zeitlicher Graph besitzt also eine Menge an sogenannten Uhren C . Diese zählen unabhängig voneinander hoch so, dass jede Uhr zu jedem „Zeitpunkt“ einen reellen Wert

¹Eine dichte Ordnung ist eine lineare Ordnung so, dass zwischen je zwei Elementen ein drittes liegt.

$x \in \mathbb{R}_{\geq 0}$ speichert. Mithilfe von π lassen sich Uhren zurücksetzen, für jede Transition lässt sich also eine Menge an Uhren auswählen, welche mit dem Wechsel über die Transition zurückgesetzt werden. Zusätzlich existiert für jede Kante eine Formel, welche bestimmte Voraussetzungen an die Transition stellt.

Semantisch soll ein zeitlicher Graph so interpretiert werden, dass eine Kante $e \in E$ nur dann genommen werden kann, wenn $\tau(e)$ zum aktuellen Zeitpunkt erfüllt wird. Alle Uhren zählen gleich schnell hoch, können aber natürlich unterschiedliche Werte besitzen, da das Zurücksetzen unabhängig voneinander passieren kann.

Um diese Definition besser zu verstehen, soll ein Beispiel angeführt werden.

Beispiel 6 (Beispiel für einen zeitlichen Graphen)

Sei $AP = \{\text{running}, \text{stopped}, \text{warning}, \text{error}\}$ eine Menge an atomaren Aussagen und $\mathfrak{G} = (S, s_0, E, C, \pi, \tau, \mu)$ ein zeitlicher Graph mit

- $S := \{s_0, s_1, s_2\}$
- $E := \{(s_0, s_2), (s_1, s_0), (s_1, s_1), (s_1, s_2), (s_2, s_1)\}$
- $C := \{x, y\}$
- $\pi(u, v) = \begin{cases} \{x\} & \text{falls } v = s_2 \\ \{y\} & \text{falls } (u, v) = (s_1, s_0) \\ \emptyset & \text{sonst} \end{cases}$
- $\tau(u, v) = \begin{cases} \top & \text{falls } v = s_2 \\ x \geq 2 & \text{falls } u = s_2 \\ x \geq 3 & \text{sonst} \end{cases}$
- $\mu(s) = \begin{cases} \{\text{running}\} & \text{falls } s = s_0 \\ \{\text{stopped}, \text{warning}\} & \text{falls } s = s_1 \\ \{\text{stopped}, \text{error}\} & \text{falls } s = s_2 \end{cases}$

Dann lässt sich die graphische Darstellung von \mathfrak{G} in Abbildung 3 sehen².

Es lässt sich also sehen, dass die Uhr x die Zeit stoppt, die seit dem Letzten Fehler passiert ist, während die Uhr y die Zeit misst, die seit der Letzten Wiederaufnahme des Betriebs des Systems vergangen ist. Weiter gibt τ uns die Information, dass wenn ein Fehler auftritt, für zwei Zeiteinheiten der Fehler bestehen bleiben muss und die darauf folgende Fehlermeldung für mindestens drei Zeiteinheiten seit dem auftreten des Fehlers beibehalten wird.

Um nun TCTL zu definieren benötigen wir noch einige Begriffe und Strukturen. Für einen zeitlichen Graphen \mathfrak{G} definiert $\Gamma(\mathfrak{G})$ die Menge aller möglichen Zuweisungen von Werten zu den Uhren, formal also $\Gamma(\mathfrak{G}) := \{\nu \mid \nu : C \rightarrow \mathbb{R}_{\geq 0}\}$. Weiter schreiben wir für ein

²Die Formel \top wird als Platzhalter für eine beliebige Tautologie wie zum Beispiel $x \geq 2 \vee \neg(x \geq 2)$.

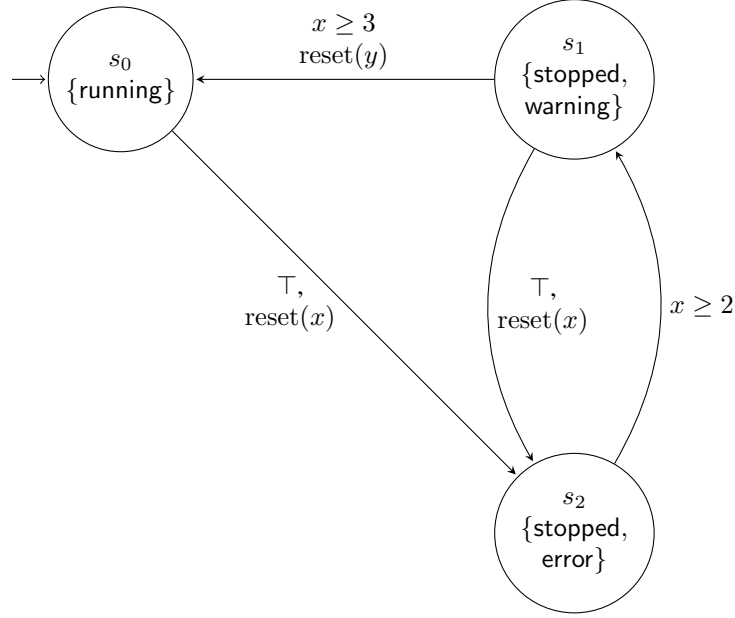


Abbildung 3: Die graphische Darstellung des zeitlichen Graphen \mathfrak{G}

$\nu \in \Gamma(\mathfrak{G})$ und ein $t \in \mathbb{R}_{\geq 0}$, den Ausdruck $\nu + t$ für eine neue Zuweisung definiert als $(\nu + t)(x) := \nu(x) + t$ und für ein $x \in C$ definieren wir die Schreibweise $[x \mapsto t]\nu$ als

$$([x \mapsto t]\nu)(y) := \begin{cases} t & \text{falls } x = y \\ \nu(y) & \text{sonst} \end{cases}$$

Eine interessante Beobachtung ist die, dass die eine Konfiguration eines zeitlichen Graphens ein Paar aus Zustand und Uhr-Belegungen sind. Der aktuelle Stand eines zeitlichen Graphens \mathfrak{G} lässt sich also eindeutig durch das Paar $\langle s, \nu \rangle$ charakterisieren, wobei s ein Zustand ist und $\nu \in \Gamma(\mathfrak{G})$ gilt. [ACD90]

Damit können wir Läufe durch zeitliche Graphen definieren, welche stetige Zustandswechsel erlauben. Es ist also möglich nicht nur zu diskreten Zeitpunkten einen Zustand zu wechseln, wodurch die in der Einleitung benannten Vorteile von TCTL verwendet werden können.

Definition 4.2 ($\langle s, \nu \rangle$ -Lauf)

Sei $\langle s, \nu \rangle$ eine Konfiguration des zeitlichen Graphen $\mathfrak{G} = (S, s_0, E, C, \pi, \tau, \mu)$. Dann ist ein $\langle s, \nu \rangle$ -Lauf eine unendliche Folge an Tripeln der Form

$$(\langle s, \nu, 0 \rangle = \langle s_1, \nu_1, t_1 \rangle, \langle s_2, \nu_2, t_2 \rangle, \dots) \in (S \times \Gamma(\mathfrak{G}) \times \mathbb{R}_{\geq 0})^\omega$$

so, dass folgende Regeln eingehalten werden [ACD90]:

1. Für alle $i \in \mathbb{N} \setminus \{0\}$ gilt $t_{i+1} > t_i$. Die Zeit-Werte steigen also streng monoton.
2. Für die Tripel $\langle s_i, \nu_i, t_i \rangle$ und $\langle s_{i+1}, \nu_{i+1}, t_{i+1} \rangle$ definieren wir $e_i = (s_i, s_{i+1})$ und es muss gelten $e_i \in E$.
3. Für alle $i \in \mathbb{N} \setminus \{0\}$ gilt $\nu_{i+1} = [\pi(e_i) \mapsto 0](\nu_i + t_{i+1} - t_i)$.
4. Für die boolesche Voraussetzung $\tau(e_i)$ ist $(\nu_i + t_{i+1} - t_i)$ eine erfüllende Belegung der Uhr-Werte.
5. Für jedes $t \in \mathbb{R}_{\geq 0}$ gibt es ein j so, dass $t_j \geq t$. Es gibt also keine obere Schranke für die Zeit-Werte.

Nun können wir TCTL definieren.

Definition 4.3 (Syntax von TCTL)

Sei AP eine Menge an atomaren Aussagen, $p \in \text{AP}$ und $c \in \mathcal{N}$. Dann definieren die beiden folgenden Grammatiken die Logik TCTL:

$$\begin{aligned} \varphi &::= p \mid \varphi \wedge \varphi \mid \neg \varphi \mid \exists \varphi \text{U}_{\kappa c} \varphi \mid \forall \varphi \text{U}_{\kappa c} \varphi \\ \kappa &::= < \mid \leq \mid = \mid \geq \mid > \end{aligned}$$

Die Menge der TCTL-Formeln sind die ausdrücke, die von φ erzeugt werden. [ACD90]

Informell soll $\exists \varphi_1 \text{U}_{\sim c} \varphi_2$ bedeuten, dass es einen Präfix gibt, der κc Zeiteinheiten lang ist und in dem φ_1 gilt, wobei im Zustand nach dem Präfix φ_2 gelten muss. Die Bedeutung von $\forall \varphi_1 \text{U}_{\kappa c} \varphi_2$ ist analog.

Schließlich können wir die Semantik von TCTL definieren.

Definition 4.4 (Semantik von TCTL)

Sei $\mathfrak{G} = (S, s_0, E, C, \pi, \tau, \mu)$ ein zeitlicher Graph. Dann können wir die Modellrelation \models zwischen Konfigurationen von zeitlichen Graphen und TCTL-Formeln induktiv definieren. Sei $s \in S$, $nu \in \Gamma(\mathfrak{G})$, $p \in \text{AP}$, $c \in \mathcal{N}$ und $\sim \in \{<, \leq, =, \geq, >\}$. [ACD90]

- $\langle s, \nu \rangle \models p$ gdw. $p \in \mu(s)$
- $\langle s, \nu \rangle \models \varphi_1 \wedge \varphi_2$ gdw. $s \models \varphi_1$ und $s \models \varphi_2$
- $\langle s, \nu \rangle \models \neg \varphi$ gdw. $s \not\models \varphi$
- $\langle s, \nu \rangle \models \exists \varphi_1 \text{U}_{\sim c} \varphi_2$ gdw. es einen $\langle s, \nu \rangle$ -Lauf gibt, mit $t_i \sim c$, $s_i \models \varphi_2$ und $s_j \models \varphi_1$ für $1 \leq j < i$.
- $\langle s, \nu \rangle \models \forall \varphi_1 \text{U}_{\kappa c} \varphi_2$ gdw. für alle $\langle s, \nu \rangle$ -Läufe gilt, dass es ein $i \in \mathbb{N}$ gibt, mit $t_i \sim c$, $s_i \models \varphi_2$ und für alle $1 \leq j < i$ gilt $s_j \models \varphi_1$.

Damit können wir bestimmen, wann eine Konfiguration eine TCTL-Formel erfüllt.

Beispiel 7 (Beispiel für das Auswerten von TCTL-Formeln)

Betrachte bspw. den zeitlichen Graphen aus Abbildung 3 und die Formel

$$\varphi := \exists \text{stopped } U_{<3} \text{ running},$$

die besagt, dass das Systems stoppt und in weniger als drei Zeiteinheiten in einen laufenden Zustand wechselt. Sei ν eine zeitliche Belegung mit $\nu(x) = 0$ und $\nu(y) \in \mathbb{R}_{\geq 0}$ beliebig. Dann stellt die Konfiguration $\alpha = \langle s_2, \nu \rangle$ dar, dass gerade in den Zustand s_2 gewechselt wurde, wodurch die Uhr x zurückgesetzt wurde. Wenn wir nun $\alpha \models \varphi$ betrachten, dann fällt auf, dass der einzige Pfad zu einem Zustand mit running der Pfad s_2, s_1, s_0 ist. Weiter ist die Transition (s_1, s_0) erst möglich, wenn $x \geq 3$ gilt. Da wir unsere Auswertung mit $\nu_1(x) = 0 = t_1$ beginnen, ist $\nu_i(x) = t_i$ für alle $i \geq 1$. Daraus ergibt sich aber auch, dass für alle $\langle s, \nu \rangle$ -Läufe gilt, dass wenn $s_i = s_0$ gilt, $\nu_i(x) = t_i \geq 3$ gelten muss. Es folgt also $\alpha \not\models \varphi$.

Wie man an diesem Beispiel erkennen konnte, ist das Auswerten von TCTL-Formeln nicht trivial. Aus diesem Grund ist die algorithmische Betrachtung des Model-Checking Spiels für TCTL äußerst interessant und wird in [ACD90] sehr konkret besprochen. Da der vorgestellte Algorithmus aber um einiges komplexer ist, also die in Kapitel 3.2 diskutierten, soll hier bloß die grobe Idee angemerkt werden. Es lässt sich feststellen, dass im Kontext des Model-Checkings zwei Zeitbelegungen ν und ν' äquivalent sind ($\nu \cong \nu'$), wenn die abgerundeten Ganzzahlwerte und die Reihenfolge der Nachkomma-Werte gleich sind.

Formal verwenden wir $\lfloor x \rfloor$ für die größte ganze Zahl y mit $y \leq x$ und $\text{frac}(x)$ für $x - \lfloor x \rfloor$. Weiter sei für $x \in C$ der Wert $c_x \in \mathcal{N}$ definiert als $c_x := \max\{c : c \leq x \text{ oder } x \leq c \text{ kommt in } \tau(e) \text{ vor, } e \in E \text{ bel.}\}$, also als größte Konstante, mit der x verglichen wird. Dann gilt für $\nu, \nu' \in \Gamma(\mathfrak{S})$, dass $\nu \cong \nu'$ genau dann, wenn

1. für alle $x \in C$ gilt, dass entweder $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ oder dann $\nu(x)$ und $\nu'(x)$ größer als c_x sind und
2. für alle $x, y \in C$ mit $\nu(x) \leq c_x$ und $\nu(y) \leq c_y$ gilt, dass $\text{frac}(\nu(x)) \leq \text{frac}(\nu(y))$ gdw. $\text{frac}(\nu'(x)) \leq \text{frac}(\nu'(y))$. [ACD90]

Damit lässt sich dann erkennen, dass für eine beliebige TCTL-Formel φ und zwei äquivalente $\nu, \nu' \in \Gamma(\mathfrak{S})$ gilt, dass $\langle s, \nu \rangle \models \varphi$ gdw. $\langle s, \nu' \rangle \models \varphi$. [ACD90]

Betrachtet man nun die Äquivalenzklassen $\{[\nu] : \nu \in \Gamma(\mathfrak{S})\}$ der soeben definierten Äquivalenzrelation erhält man Mengen an Konfigurationen $\langle s, [\nu] \rangle$, welche wir Regionen nennen. Schlussendlich lässt sich der Model-Checking Algorithmus dann über Pfade dieser Regionen definieren. [ACD90]

Der daraus resultierende Model-Checking Algorithmus hat eine Komplexität von

$$\mathcal{O} \left[c(\varphi) \cdot |\varphi| \cdot (|S| + |E|) \cdot |C|! \cdot \prod_{x \in C} c_x \right],$$

wobei $c(\varphi)$ die größte in φ vorkommende Konstante ist. Der Algorithmus wächst also linear in Formellänge als auch Graphengröße, aber wächst exponentiell mit der Anzahl an Uhren. [ACD90]

Dies stellt eine sehr viel höhere Komplexität dar, als die in Kapitel 3.2 vorgestellten Algorithmen. Zudem waren die PCTL-Algorithmen nicht optimiert für $p \in \{0, 1\}$, was hier ja aber der Fall ist. Offensichtlich ist die Art, wie PCTL Zeit implementiert sehr viel schwächer als TCTLs Implementation, jedoch ist diese Steigerung an Aussagekraft auch sehr klar in der Model-Checking Komplexität erkennbar. Ein Ausweiten von TCTL bzw. zeitlichen Graphen auf Markov-Ketten würde zusätzliche Komplikationen hinzufügen. So würde es bspw. nicht mehr nur ausreichen, \mathcal{T} von einer Kante abhängig zu machen. Da es in einem zeitlichen Graphen nicht möglich ist, Kanten zu verwenden, deren boolesche Formeln nicht erfüllt werden, würde die das Wahrscheinlichkeitsmaß in diesen Fällen 0 sein müssen. Demnach wäre es nötig, \mathcal{T} von der Kante, dem ausgehenden Knoten und einem zeitlichen Parameter abhängig zu machen.

Die Aussagekraft von TCTL ist im Vergleich aber um einiges höher. So lässt sich zum Beispiel Zeit genau so darstellen wie in PCTL indem eine Uhr x verwendet wird, die bei jedem Zustandswechsel zurückgesetzt wird und indem wir $\tau(e)$ für alle e neu definieren durch $\tau(e) \leftarrow \tau(e) \wedge x \leq 1 \wedge 1 \leq x$. Dadurch würde jede Transition nur bei $x = 1$ genommen werden, wodurch das Verhalten, wie es PCTL bei Markov-Ketten voraussetzt implementiert wird. Offensichtlich lassen sich in TCTL aber auch weitere Verhalten darstellen. So könnte man bspw. bei einem Wechsel in einen Zustand für die ersten fünf Zeiteinheiten eine Menge an Kanten erlauben, nach den ersten fünf aber eine andere. In PCTL wäre solch ein Verhalten nicht möglich, für das modellieren von Systemen ist solch eine Eigenschaft aber äußerst hilfreich, da nach fünf Zeiteinheiten bspw. nur noch in verschiedene Fehlerzustände gewechselt werden könnte.

Es lässt sich also erkennen, dass PCTL trotz der, bzgl. Zeit, geringeren Aussagekraft auch Vorteile gegenüber TCTL bietet. Falls es aber wichtig ist starke Aussagen über das zeitliche Verhalten eines Systems zu formulieren, wäre TCTL vermutlich die bessere Wahl.

4.2 Generalized Probabilistic Logic

Der *modale μ -Kalkül* ($L\text{-}\mu$) stellt eine sehr wichtige Fixpunktlogik für Korrektheitsverifikation dar. Das liegt daran, dass es möglich ist, die meisten für Model-Checking verwendeten Logiken in diese einzubetten. So lässt sich beispielsweise zu jeder CTL- oder CTL*-Formel eine äquivalente $L\text{-}\mu$ -Formel definieren. Zusätzlich lassen sich aber auch noch weitere Aussagen treffen, wie bspw. dass auf einem Pfad in jedem zweiten Zustand eine Bedingung gilt. Dies lässt sich weder in CTL, noch CTL* ausdrücken. [CIN05]

Zu Beginn soll die Syntax und Semantik kurz definiert werden, für mehr Informationen wird dabei aber auf [SW91] und [Koz83] verwiesen.

Definition 4.5 (Syntax und Semantik von $L\text{-}\mu$)

Die Menge der $L\text{-}\mu$ -Formeln wird für Variablen X , atomare Aussagen a und Aktionen α durch die folgende Grammatik definiert:

$$\varphi ::= X \mid a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle \alpha \rangle \varphi \mid [\alpha] \varphi \mid \mu X. \varphi \mid \nu X. \varphi$$

Variablen werden, wie aus FO bekannt, durch die Operatoren ν und μ gebunden und sind

sonst frei. Der Einfachheit halber sind freie Variablen im Weiteren nicht erlaubt.

Für den modalen μ -Kalkül benötigen wir eine etwas abgeänderte Art an Transitionssystem. Sei Act eine Menge an Aktionen, S eine Menge an Zuständen und AP eine Menge an atomaren Aussagen. Dann ist $\mathcal{K} = (S, (E_\alpha)_{\alpha \in \text{Act}}, \mathcal{L} : S \rightarrow 2^{\text{AP}})$ ein wie vorher definiertes Transitionssystem mit dem Unterschied, dass für jedes $\alpha \in \text{Act}$ eine einzelne Kantenrelation existiert. Damit können wir die Semantik von L- μ -Formeln als Modellbeziehung zwischen Zuständen $s \in S$ und Formeln induktiv definieren:

- $s \models a$ gdw. $a \in \mathcal{L}(s)$
- $s \models \neg a$ gdw. $a \notin \mathcal{L}(s)$
- $s \models \varphi_1 \wedge \varphi_2$ gdw. $s \models \varphi_1$ und $s \models \varphi_2$
- $s \models \varphi_1 \vee \varphi_2$ gdw. $s \models \varphi_1$ oder $s \models \varphi_2$
- $s \models \langle \alpha \rangle \varphi$ gdw. es ein $s' \in S$ mit $(s, s') \in E_\alpha$ gibt so, dass $s' \models \varphi$
- $s \models [\alpha] \varphi$ gdw. für alle $s' \in S$ mit $(s, s') \in E_\alpha$, $s' \models \varphi$ gilt
- $s \models \mu X. \varphi$ gdw. $s \in \text{lfp}(F_\varphi)$
- $s \models \nu X. \varphi$ gdw. $s \in \text{gfp}(F_\varphi)$

Es lässt sich also ein ähnlicher Prozess wie bei PCTL erkennen. Für eine qualitative Logik, welche bloß aussagen kann, ob ein Zustand eine Spezifikation erfüllt, wird eine quantitative Erweiterung definiert, die genauere Aussagen über das spezifische Verhalten treffen kann. Bevor dies für L- μ aber passieren kann, werden noch einige Vorschritte benötigt.

Definition 4.6 (Reaktive-probabilistische-Transitionssysteme)

Sei S eine abzählbare Menge an Zuständen, Act eine Menge an Aktionen und AP eine Menge an atomaren Aussagen. Dann bezeichnen wir ein Tupel $\mathfrak{S} = (S, \delta, P, \mathcal{L})$ als ein *reaktives-probabilistisches-Transitionssystem* (RPLTS), wenn $\delta \subseteq S \times \text{Act} \times S$ eine Übergangsrelation, $P : \delta \rightarrow (0, 1]$ eine Transitions-Wahrscheinlichkeitsfunktion und $\mathcal{L} : S \rightarrow 2^{\text{AP}}$ eine Bezeichnungsfunktion ist. Weiter muss für P gelten, dass

1. Für alle $s \in S$ und alle $\alpha \in \text{Act}$ gilt $\sum_{s': (s, \alpha, s') \in \delta} P(s, \alpha, s') \in \{0, 1\}$
2. Für alle $s \in S$ und alle $\alpha \in \text{Act}$ gilt, wenn es ein s' mit $(s, \alpha, s') \in \delta$ gibt, dann ist $\sum_{s': (s, \alpha, s') \in \delta} P(s, \alpha, s') = 1$ [CIN05]

Nun können wir Berechnungen von RPLTS als Abfolge von Zuständen der Form $\sigma = s_0 \xrightarrow{a_1} s_1 \cdots \xrightarrow{a_n} s_n$ definieren mit der Voraussetzung, dass für $0 \leq i < n$, $(s_i, a_{i+1}, s_{i+1}) \in \delta$. Weiter ist ein σ' der Präfix eines $\sigma = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n$, wenn es ein $i \leq n$ gibt, mit $\sigma' = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_i} s_i$ und als Abkürzung schreiben wir $\mathcal{C}_\mathfrak{S}$ für alle Berechnungen im RPLTS \mathfrak{S} und $\mathcal{C}_\mathfrak{S}(s) := \{\sigma \in \mathcal{C}_\mathfrak{S} : s_0 = s\}$. [CIN05]

Damit erhalten wir eine Definition von Bäumen.

Definition 4.7 (d-Bäume)

Sei $\mathfrak{S} = (S, \delta, P, \mathcal{L})$ ein RPLTS und sei $T \subseteq \mathcal{C}_{\mathfrak{S}}$. Wir nennen T einen *d-Baum*, wenn T folgende Bedingungen erfüllt [CIN05]:

- T ist abgeschlossen unter Präfixen, d.h. wenn $\sigma \in T$ und σ' ist ein Präfix von σ , dann ist $\sigma' \in T$
- T ist deterministisch, d.h. wenn $\sigma, \sigma' \in T$ mit $\sigma = s_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_n \xrightarrow{a} s \dots$ und $\sigma' = s_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_n \xrightarrow{a'} s' \dots$, dann ist entweder $a \neq a'$ oder $s = s'$. Intuitiv bedeutet das, dass wenn zwei Berechnungspfade einen gemeinsamen Präfix haben, es entweder eine Stelle geben muss, an denen sich die verwendeten Aktionen unterscheiden oder es die gleichen Pfade sein müssen.
- Es existiert ein $s \in S$ mit $T \subseteq \mathcal{C}_{\mathfrak{S}}(s)$. s bezeichnen wir als Wurzel von T

Des weiteren nennen wir einen d-Baum T *maximal*, wenn es kein T' mit $T \subsetneq T'$ gibt und für ein RPLTS \mathfrak{S} ist $\mathcal{M}_{\mathfrak{S}}$ die Menge aller maximalen d-Bäume, bzw. $\mathcal{M}_{\mathfrak{S}}(s)$ die Menge aller maximalen d-Bäume T mit $T \subseteq \mathcal{C}_{\mathfrak{S}}(s)$. [CIN05]

Für eine Menge B an d-Bäumen mit gemeinsamer Wurzel für den gilt, dass es einen Baum T gibt so, dass für jedes $T' \in B$, $T \subseteq T'$ gilt, lässt sich ein Wahrscheinlichkeitsmaß $\mu_{\mathfrak{S}}$ definieren mit

$$\mu_{\mathfrak{S}}(T) := \prod_{s_0 \xrightarrow{a_1} \dots \xrightarrow{a_i} s_i \xrightarrow{a_{i+1}} s_{i+1} \in T} P(s_i, a_{i+1}, s_{i+1})$$

Wie in [CIN05] gezeigt, verhält sich dies dann wie ein Wahrscheinlichkeitsmaß.

Mit diesem Maß haben wir die Werkzeuge, um die Erweiterung von $L\text{-}\mu$ zu definieren.

Definition 4.8 (Syntax von der generalized probabilistic logic)

Die generalized probabilistic logic (GPL) lässt sich durch die folgenden Grammatiken definieren. Dabei ist X eine Variable, $a \in \text{Act}$ eine Aktion und $p \in [0, 1] \subseteq \mathbb{R}$.

$$\begin{aligned} \Phi &::= a \mid \neg a \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \text{Pr}_{>p} \Psi \mid \text{Pr}_{\geq p} \Psi \\ \Psi &::= \Phi \mid X \mid \Psi \wedge \Psi \mid \Psi \vee \Psi \mid \langle a \rangle \Psi \mid [a] \Psi \mid \mu X. \Psi \mid \nu X. \Psi \end{aligned}$$

Wie auch bei $L\text{-}\mu$ werden keine freien Variablen erlaubt. Auch ist eine Alternation von Fixpunktoperatoren nicht zulässig.

Intuitiv sollen von Φ erzeugte Formeln Zustandsformeln darstellen und sind schlussendlich auch GPL-Formeln. Die von Ψ erzeugten Formeln sind fusselige Formeln. Dementsprechend stellt Φ die Menge aller Zustands- und Ψ die Menge aller fusselige Formelndar.

Um die Formeln nun auszuwerten, müssen wir zwischen Φ und Ψ unterscheiden. Das Ziel ist es, $\psi \in \Psi$ zu einem Wert in $[0, 1]$ auszuwerten, um dann mithilfe des Pr -Operators die Modelleigenschaft zu entscheiden. Daher wollen wir dafür das Wahrscheinlichkeitsmaß

$\mu_{\mathfrak{G}}$ verwenden. Genauer wird mit der Funktion $\Theta_{\mathfrak{G}} : \Psi \rightarrow 2^{\mathcal{M}_{\mathfrak{G}}}$ jeder fusselige Formeln eine Menge an maximalen Bäumen über dem RPLTS zugeordnet, welche dann zu einem Wahrscheinlichkeitswert ausgewertet werden kann. [CIN05]

Diese Idee wollen wir nun formalisieren. Die Funktion $\Theta_{\mathfrak{G}}$ wird induktiv über den Formelaufbau definiert:

- $\Theta_{\mathfrak{G}}(\varphi) = \bigcup_{s \models \varphi} \mathcal{M}_{\mathfrak{G}}$, wobei $\varphi \in \Phi$
- $\Theta_{\mathfrak{G}}(\langle a \rangle \psi) = \{T \in \mathcal{M}_{\mathfrak{G}} : \exists T' : T \xrightarrow{a} T' \wedge T' \in \Theta_{\mathfrak{G}}(\psi)\}$, wobei $T \xrightarrow{a} T'$ bedeutet, dass es eine a -Transition von der Wurzel von T zur Wurzel von T' gibt
- $\Theta_{\mathfrak{G}}([a]\psi) = \{T \in \mathcal{M}_{\mathfrak{G}} : (T \xrightarrow{a} T') \Rightarrow T' \in \Theta_{\mathfrak{G}}(\psi)\}$
- $\Theta_{\mathfrak{G}}(\psi_1 \wedge \psi_2) = \Theta_{\mathfrak{G}}(\psi_1) \cap \Theta_{\mathfrak{G}}(\psi_2)$
- $\Theta_{\mathfrak{G}}(\psi_1 \vee \psi_2) = \Theta_{\mathfrak{G}}(\psi_1) \cup \Theta_{\mathfrak{G}}(\psi_2)$
- $\Theta_{\mathfrak{G}}(\mu X.\psi) = \bigcup_{i=0}^{\infty} M_i$, mit $M_0 = \emptyset$ und $M_{i+1} = \Theta_{\mathfrak{G}}(\psi[X \mapsto M_i])$, wobei $\psi[X \mapsto M_i]$ die Formel ψ ist, wo X durch die monadische Relation M_i ersetzt wird
- $\Theta_{\mathfrak{G}}(\nu X.\psi) = \bigcap_{i=1}^{\infty} N_i$, mit $N_0 = \mathcal{M}_{\mathfrak{G}}$ und $N_{i+1} = \Theta_{\mathfrak{G}}(\psi[X \mapsto N_i])$

Weiter können wir für ein $s \in S$ $\Theta_{\mathfrak{G},s}(\psi) := \Theta_{\mathfrak{G}} \cap \mathcal{M}_{\mathfrak{G}}(s)$ definieren. [CIN05]

Mit dieser Funktion können wir nun die Semantik von GPL definieren.

Definition 4.9 (Semantik von GPL)

Sei $\mathfrak{G} = (S, \delta, P, \mathcal{L})$ ein RPLTS und $a \in \text{AP}$. Dann ist $\models_{\mathfrak{G}} \subseteq S \times \Phi$ induktiv definiert [CIN05]:

- $s \models_{\mathfrak{G}} a$ gdw. $a \in \mathcal{L}(s)$
- $s \models_{\mathfrak{G}} \neg a$ gdw. $a \notin \mathcal{L}(s)$
- $s \models_{\mathfrak{G}} \varphi_1 \wedge \varphi_2$ gdw. $s \models_{\mathfrak{G}} \varphi_1$ und $s \models_{\mathfrak{G}} \varphi_2$
- $s \models_{\mathfrak{G}} \varphi_1 \vee \varphi_2$ gdw. $s \models_{\mathfrak{G}} \varphi_1$ oder $s \models_{\mathfrak{G}} \varphi_2$
- $s \models_{\mathfrak{G}} \text{Pr}_{>p} \psi$ gdw. $\mu_{\mathfrak{G}}(\Theta_{\mathfrak{G},s}(\psi)) > p$
- $s \models_{\mathfrak{G}} \text{Pr}_{\geq p}$ gdw. $\mu_{\mathfrak{G}}(\Theta_{\mathfrak{G},s}(\psi)) \geq p$

Damit haben wir GPL vollständig definiert und können die Logik an einem Beispiel betrachten. Zur Vereinfachung schreiben wir $\langle \cdot \rangle \psi := \bigvee_{a \in \text{Act}} \langle a \rangle \psi$ und $[\cdot] \psi := \bigwedge_{a \in \text{Act}} [a] \psi$.

Beispiel 8 (Beispiel für GPL)

Man betrachte den Graphen aus Abbildung 4, welcher einen RPLTS darstellt. Als erstes Beispiel soll die fusselige Formeln $\psi_1 := \mu X.\text{stop} \vee \langle \cdot \rangle X$ welche aussagt, dass es einen

Pfad zu einem Knoten gibt, welcher mit stop annotiert ist. Da A der Startzustand ist, erhalten wir $\Theta_{\mathfrak{S},A}(\psi_1) = \{A \xrightarrow{a} B \xrightarrow{a} C\}$ und damit dann $\mu_{\mathfrak{S}}(\Theta_{\mathfrak{S},A}(\psi_1)) = \left(\frac{1}{3}\right)^2 = \frac{1}{9}$.

Als zweites wollen wir nun noch die Wahrscheinlichkeit bestimmen, in einem mit error annotierten Zustand zu enden. Nehmen wir dafür die analoge Formel $\psi_2 := \mu X.\text{error} \vee \langle \cdot \rangle X$. Da die maximalen d-Bäume betrachtet werden, erhalten wir $\Theta_{\mathfrak{S},A}(\psi_2) = \{A \xrightarrow{a} D, A \xrightarrow{a} B \xrightarrow{a} D\}$ und daraus $\mu_{\mathfrak{S}}(\Theta_{\mathfrak{S},A}(\psi_2)) = \mu_{\mathfrak{S}}(A \xrightarrow{a} D) + \mu_{\mathfrak{S}}(A \xrightarrow{a} B \xrightarrow{a} D) = \frac{2}{3} + \frac{1}{3} \cdot \frac{2}{3} = \frac{8}{9}$.

Es fällt also auf, dass $\mu_{\mathfrak{S}}(\Theta_{\mathfrak{S},A}(\psi_1)) + \mu_{\mathfrak{S}}(\Theta_{\mathfrak{S},A}(\psi_2)) = 1$. Es gibt demnach nur die Möglichkeiten in einem error- oder einem stop-Zustand zu enden. Bei Betrachtung des RPLTS ist dies offensichtlich leicht einsehbar, wir haben dies nun aber auch mithilfe von GPL bewiesen.

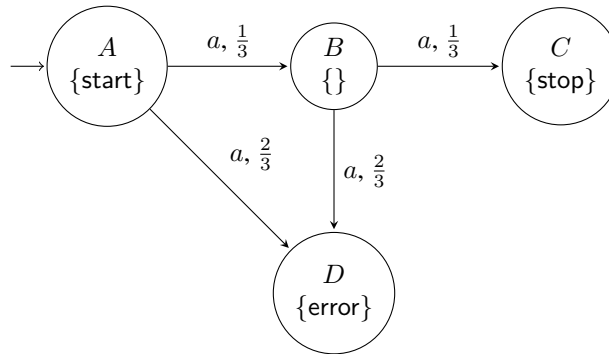


Abbildung 4: Die graphische Darstellung des RPLTS \mathfrak{S}

Wir haben damit eine sehr interessante Logik kennengelernt. Zum Einen stellt die „Basis-Logik“ $L-\mu$ eine sehr viel ausdrucksstärkere Logik dar als CTL, was sich auch in der Einbettbarkeit von einigen probabilistischen Logiken, wie unter anderem pCTL* sehen lässt [CIN05]. Durch Wahl geeigneter Formeln wäre es demnach vermutlich auch möglich PCTL mit Voraussetzung $t = \infty$ in GPL einzubetten. Insgesamt stellt diese Erhöhung der Ausdruckskraft aber auch Nachteile dar. So ist das Auswerten sehr viel aufwendiger und auch der in [CIN05] vorgestellte Model-Checking Algorithmus benötigt einige Schritte mehr als der für PCTL. Wir befinden uns also in einer ähnlichen Situation wie bei TCTL. Jedoch gibt es dennoch den Unterschied, dass GPL mit zeitlichen Aspekten wie PCTL erweitert werden könnte, da bloß das Zählen der verwendeten Kanten nötig ist. Dies ließe sich bspw. durch das Kontrollieren der Baumlänge durchführen. Eine solche Erweiterung um, wie PCTL, sowohl über Zeit als auch Wahrscheinlichkeiten aussagen zu treffen scheint also um einiges realistischer als für TCTL.

Zusätzlich sind die Aussagen, die GPL über Wahrscheinlichkeiten aufstellen kann sehr ähnlich zu denen von PCTL, unter Anbetracht, dass GPL die deutlich mächtigere Logik ist. Dies liegt daran, dass beide Logiken die möglichen Pfade durchsuchen, um die Wahrscheinlichkeit des Erreichens von Regionen zu berechnen. Als Schnittstelle bieten beide

Logiken auch die Möglichkeit eine Schwelle anzugeben, welche von einem Knoten erreicht bzw. überschritten werden muss, um die Formel zu erfüllen. Die beiden Logiken ähneln sich also sehr in dem Aspekt, wie diese Wahrscheinlichkeiten implementieren.

5 Verwandte Arbeiten

PCTL stellt als Beispiel der Logiken, die andere Logiken erweitern, eine sehr interessante Rolle. So sieht man bspw. in Kapitel noch andere Logiken, welche bestimmte Probleme lösen sollen, indem sie bereits etablierte um geeignete Fähigkeiten erweitern. In diesem Kapitel sollen einige solche Erweiterungen kurz vorgestellt werden.

Eine Logik, welche die Zeitaspekte wie in PCTL implementiert ist die *Real Time Computation Tree Logic* aus [EMSS91]. Diese erlaubt Until-Operatoren der Wart $\varphi_1 U^{\leq k} \varphi_2$ mit der selben Bedeutung wie in PCTL für den Fall $p \in \{0, 1\}$. Ein anderer sehr interessanter Ansatz ist aus [JM86]. Die dort definierte Logik *Real Time Logic* stellt nämlich, anders als bisher alle Logiken in dieser Arbeit, keine modale Logik dar, und kann damit über eine Vielzahl an Eigenschaften aussagen treffen. Zum Beispiel ist es in einer nicht-modalen Fixpunktlogik, wie sie *Real Time Logic* ist, möglich eine bestimmte Anzahl an verschiedenen Nachfolgern vorauszusetzen. Aus der Bisimulationsinvarianz von $L-\mu$ ergibt sich, dass dies für PCTL oder GPL nicht möglich ist. Auch äußerst Interessant ist die Logik $\leq \omega-L-\mu$ aus [Eme91], welche $L-\mu$ mit Zeit erweitert. Dies wird dadurch erreicht, indem Fixpunktoperatoren mit natürlichen Zahlen ergänzt werden, welche darstellen, wie viele Iterationen der Fixpunktinduktion durchgeführt werden sollen. Dadurch lassen sich bspw. die Aussagen aus der *Real Time Computation Tree Logic* auch in $\leq \omega-L-\mu$ darstellen.

Auch im Aspekt Wahrscheinlichkeiten gibt es einige weitere Interessante Logiken. In [LS89] werden bspw. drei Logiken mit aufsteigender Aussagekraft definiert. Was diese Interessant macht ist, dass diese Logiken durch ein Übersetzen in eine „Testsprache“ ausgewertet werden. Zusätzlich werden dann mögliche Durchläufe des Tests durch das (nichtdeterministische) Programm definiert. Durch Anpassen der Anzahl an Durchläufen lässt sich dies so ausweiten, dass für einen beliebigen erlaubten Fehler das Programm auf Korrektheit getestet wird. Betrachtet man die Logik aus [CY95] fällt einem auf, dass das Gleichungssystem zum Berechnen des *Until*-Operators das selbe ist, dass auch im Algorithmus für PCTL erzeugt wird. Es lässt sich dort also eine genauere Untersuchung der Eigenschaften von PCTL bzgl. der Wahrscheinlichkeitsaussagen finden.

6 Konklusion

In dieser Arbeit wurde sich eingehend mit der Logik PCTL beschäftigt. Diese stellt eine Erweiterung der Logik CTL dar und ermöglicht es, Eigenschaften von Soft-Realtime Systemen auszudrücken, da sie sowohl über Wahrscheinlichkeiten, als auch Zeit Aussagen treffen kann. Erzielt wird dies durch einen erweiterten *Until*-Operator, welcher einen Zeit und einen Wahrscheinlichkeitsparameter erhält. Insgesamt ergeben sich sehr effiziente

Model-Checking Algorithmen für unterschiedliche Fälle.

Vergleicht man PCTL mit anderen Logiken fällt auf, dass diese Logik eine sehr hohe Ausdruckskraft bzgl. Wahrscheinlichkeiten besitzt. So sind die dafür verwendeten Mechanismen die gleichen wie für GPL, welche eine deutlich stärkere Logik darstellt und eine Vielzahl an anderen Logiken ausdrücken kann. Bzgl. Zeit existieren aber sehr viel mächtigere Implementationen, wie am Beispiel von TCTL gesehen wurde, welche sehr viel mehr zeitliche Aspekte formulieren kann als PCTL.

Literatur

- [ACD90] Rajeev Alur, Costas Courcoubetis und David Dill. Model-checking for real-time systems. In *[1990] Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*, Seiten 414–425. IEEE, 1990.
- [BK08] Christel Baier und Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.
- [CE82] Edmund M Clarke und E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs: Workshop, Yorktown Heights, New York, May 1981*, Seiten 52–71. Springer, 1982.
- [CES86] Edmund M Clarke, E Allen Emerson und A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.
- [CIN05] Rance Cleaveland, S Purushothaman Iyer und Murali Narasimha. Probabilistic temporal logics via the modal μ -calculus. *Theoretical Computer Science*, 342(2-3):316–350, 2005.
- [CY95] Costas Courcoubetis und Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM (JACM)*, 42(4):857–907, 1995.
- [Eme91] E Allen Emerson. Real-time and the μ -calculus (preliminary report). In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*, Seiten 176–194. Springer, 1991.
- [EMSS91] E Allen Emerson, Aloysius K Mok, A Prasad Sistla und Jai Srinivasan. Quantitative temporal reasoning. In *Computer-Aided Verification: 2nd International Conference, CAV'90 New Brunswick, NJ, USA, June 18–21, 1990 Proceedings 2*, Seiten 136–145. Springer, 1991.
- [HJ94] Hans Hansson und Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6:512–535, 1994.
- [JM86] Farnam Jahanian und Aloysius Ka-Lau Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on software engineering*, (9):890–904, 1986.
- [Koz83] Dexter Kozen. Results on the propositional μ -calculus. *Theoretical computer science*, 27(3):333–354, 1983.
- [LS89] Kim G Larsen und Arne Skou. Bisimulation through probabilistic testing (preliminary report). In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, Seiten 344–352, 1989.

- [SW91] Colin Stirling und David Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, 1991.