Petrinetz-Synthese und modale Spezifikationen[1]

Uli Schlachter[2]

Abstract: Bei der Petrinetz-Synthese soll ein gegebenes Verhalten durch ein Petrinetz erzeugt werden, was bedeutet, dass der Erreichbarkeitsgraph des Petrinetzes genau das geforderte Verhalten hat. In dieser Arbeit wird die Synthese von Petrinetz-Teilklassen untersucht, beispielsweise schlichten und schlingen-freien Petrinetzen. Unlösbare LTS werden durch Überapproximation behandelt. Außerdem wird die Synthese von modalen Transitionssystemen (MTS), dem modalem μ-Kalkül und einer Teilmenge des μ-Kalküls, die konjunktiver ν-Kalkül heißt, behandelt. Das Synthese-Problem für MTS und den ν-Kalkül ist unentscheidbar, aber durch eine kleine Einschränkung der betrachteten Petrinetze wird dieses Problem sogar für den ausdrucksmächtigeren μ-Kalküls entscheidbar.

1 Einleitung

In der Informatik wird viel mit Modellen gearbeitet. Ein Modell ist dabei eine abstrakte Beschreibung eines Systems über das Aussagen gemacht werden sollen. Für verteilte Systeme muss hierbei die Nebenläufigkeit abgebildet werden, also das unabhängige, evtl. zeitgleiche, Agieren von verschiedenen, räumlich getrennten Teilen des Systems. Mit *Petrinetzen* können solche Systeme gut beschrieben werden, da es hier einen intuitiven Begriff von Nebenläufigkeit gibt [BCD02; BD11; GGS13]. Daher werden sie beispielsweise in der Geschäftsprozessmodellierung [Aa16] und in der Biologie [PWM03] eingesetzt.

Das Verhalten eines Petrinetzes wird durch seinen *Erreichbarkeitsgraphen* beschrieben. Dieser Graph enthält alle erreichbaren Zustände des Petrinetzes und die möglichen Übergänge zwischen diesen. Hiermit kann z.B. untersucht werden, ob es im Modell Zustände gibt, in denen das System nicht weiter arbeiten kann, sogenannte Verklemmungen. Jedoch ist es aufwendig zu prüfen, ob Verklemmungen existieren.

Eine Möglichkeit um gewünschte Eigenschaften für ein System, z.B. Verklemmungs-Freiheit, sicherzustellen, ist die *Synthese*. Hierbei ist eine Beschreibung des gewünschten Verhaltens gegeben und ein System soll automatisch gefunden werden. In der Petrinetz-Synthese wird beispielsweise ein prototypischer Erreichbarkeitsgraph, ein sogenanntes beschriftetes Transitionssystem (labelled transition system; LTS) gegeben und ein Petrinetz soll gefunden werden, das dieses Transitionssystem erzeugt. Hierbei wird üblicherweise gefordert, dass jede Beschriftung, die in der Eingabe vorkommt, von genau einer einzigen Transition im Petrinetz erzeugt wird.

[1] Englischer Titel der Dissertation: “Petri Net Synthesis and Modal Specifications” [Sc18]

[2] Carl von Ossietzky Universität Oldenburg, uli.schlachter@uol.de

Durch ein LTS ist das Verhalten eines Systems exakt beschrieben. Allerdings ist es wünschenswert flexiblere Spezifikationen zu haben, in denen noch Variationen möglich sind. Beispielsweise sollte es möglich sein, Verklemmungen zu verbieten, ohne konkrete Lösungen vorzugeben. Daher werden in dieser Dissertation modale Spezifikationen eingesetzt. Hierbei gibt es zwei Modi von Verhalten: Es gibt *gefordertes* Verhalten, dass auf jeden Fall im System möglich sein muss, und es gibt *erlaubtes* Verhalten, dass das System haben kann, aber das auch weggelassen werden darf. Eine modale Spezifikation beschreibt dabei eine Familie von LTS. Bei der Petrinetz-Synthese soll ein Petrinetz gefunden werden, dessen Verhalten einem dieser LTS entspricht bzw. dessen Verhalten die modale Spezifikation *implementiert*. Beispiele für modale Spezifikationen sind modale Transitionssysteme [Kř17; La89] und der modale μ-Kalkül [AN01; Ko83]. Die Petrinetz-Synthese von modalen Spezifikationen wurde in [BBD15] als interessantes offenes Problem genannt und wird in dieser Dissertation untersucht.

Der nächste Abschnitt führt Petrinetz-Synthese formal ein. In Abschnitt 3 und 4 werden Inhalte der Dissertation vorgestellt. Nachdem in Abschnitt 5 modale Spezifikationen eingeführt wurden, geht Abschnitt 6 auf den zweiten Teil der Dissertation ein: Petrinetz-Synthese aus modalen Spezifikationen.

2 Petrinetze und Petrinetz-Synthese

Ein Petrinetz ist ein Tupel $N = (P, T, F, M_0)$. P und T sind endliche und disjunkte Mengen an *Stellen* und *Transitionen*. $F\colon ((P \times T) \cup (T \times P)) \to \mathbb{N}$ ist eine *Kantenrelation*, die *Kantengewichte* angibt. Eine *Markierung* ist eine Funktion $P \to \mathbb{N}$, die jeder Stelle eine Anzahl an *Token* zuweist und M_0 ist eine ausgezeichnete Markierung, die *Initialmarkierung*.

Ein Beispiel für ein Petrinetz $N = (P, T, F, M_0)$ zeigt der linke Teil von Abbildung 1. Dieses Netz hat die Stellen $P = \{p_0, p_1, p_2, p_3\}$ und Transitionen $T = \{a, b, c\}$. Es gibt eine Kante mit Gewicht 1 von p_0 nach a, also gilt $F(p_0, a) = 1$. Außerdem gilt $F(a, p_0) = 0$, es gibt also keine Kante von a nach p_0. Die Initialmarkierung wird beschrieben durch $M_0(p_0) = M_0(p_2) = 1$ und $M_0(p_1) = M_0(p_3) = 0$.

Eine Transition $t \in T$ ist in einer Markierung M *aktiviert*, wenn jede Stelle mindestens so viele Token enthält wie durch die ausgehenden Kanten gefordert: $\forall p \in P\colon F(p,t) \leq M(p)$. Dies wird als $M[t\rangle$ geschrieben. Eine aktivierte Transition kann *feuern*, wodurch eine neue Markierung M' erreicht wird. Hierfür wird anhand der Kantengewichte die aktuelle Markierung verändert: $\forall p \in P\colon M'(p) = M(p) - F(p,t) + F(t,p)$. Dies wird als $M[t\rangle M'$ geschrieben und diese Schreibweise wird induktiv auf Sequenzen $\sigma \in T^*$ erweitert.

Beispielsweise ist im Petrinetz aus Abbildung 1 in der aktuellen Markierung Transition a aktiviert, da diese nur ein Token von der Stelle p_0 benötigt, welche aktuell genau ein Token besitzt. Dieses Token wird beim Feuern von a konsumiert und ein Token auf der Stelle p_1 produziert. Die neue Markierung M' erfüllt $M'(p_0) = 0$, $M'(p_1) = 1 = M'(p_2)$ und $M'(p_3) = 0$.

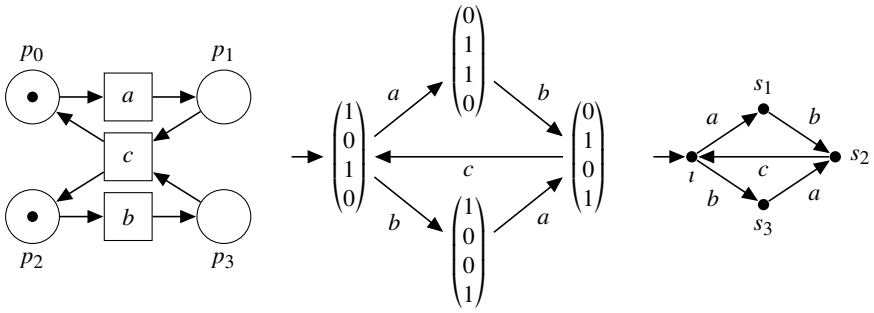

Abb. 1: Beispiel für ein Petrinetz (links), sein Verhalten (Mitte) und ein LTS (rechts)

Das Verhalten eines Prozesses, wie z.B. eines Petrinetzes, kann als LTS dargestellt werden. Ein LTS ist eine Struktur $A = (S, \Sigma, \rightarrow, \iota)$. Hier ist S eine Menge an *Zuständen*, Σ ist ein Alphabet mit *Beschriftungen* oder *Ereignissen*, $\rightarrow \subseteq S \times \Sigma \times S$ ist eine *Kantenrelation* und ι ist der *Initialzustand*.

Ein Beispiel für ein LTS ist im rechten Teil von Abbildung 1 zu sehen. Es hat ι als Initialzustand. Das Alphabet ist nicht explizit gegeben, aber enthält mindestens $\{a, b, c\}$, da dies die Beschriftungen der Kanten sind. Dieses LTS hat unter anderem die Kanten (ι, a, s_1) und (s_1, b, s_2). Somit ist z.B. auch die Sequenz ab möglich, die vom Zustand ι aus zu s_2 führt.

Das Verhalten eines Petrinetzes N kann als LTS dargestellt werden, das dessen *Erreichbarkeitsgraph* $\mathrm{RG}(N)$ genannt wird. Hierfür wird die Menge aller erreichbaren Markierungen $\mathfrak{E}(N) = \{M \mid \exists \sigma \in T^* \colon M_0[\sigma\rangle M\}$ als Zustandsmenge verwendet. Eine Kante mit Beschriftung $t \in T$ existiert zwischen zwei Markierungen $M, M' \in \mathfrak{E}(N)$ genau wenn $M[t\rangle M'$ gilt. Der Erreichbarkeitsgraph des Petrinetzes aus dem linken Teil von Abbildung 1 ist in der Mitte der gleichen Abbildung illustriert. Hierfür wird eine Markierung M als Spaltenvektor $M = (M(p_0)\ M(p_1)\ M(p_2)\ M(p_3))$ geschrieben.

Für ein gegebenes Petrinetz ist es recht einfach seinen Erreichbarkeitsgraphen zu berechnen, solange das Petrinetz nur endlich viele verschiedene erreichbare Markierungen hat. Beispielsweise kann durch Tiefen- oder Breitensuche der gesamte Erreichbarkeitsgraph bestimmt werden. *Petrinetz-Synthese* ist die umgekehrte Operation: Ein endliches LTS ist gegeben und es soll ein Petrinetz bestimmt werden, dass dieses LTS löst. Dies bedeutet, dass der Erreichbarkeitsgraph des Petrinetzes isomorph zum gegebenen LTS sein soll, also die gleiche Form haben muss. Von den konkreten Markierungen wird dabei abstrahiert.

Die Grundlage für die Petrinetz-Synthese sind sogenannte *Regionen*. Eine Region eines LTS entspricht einer möglichen Stelle eines Petrinetzes, dass das LTS lösen soll. Die Regionen-Theorie beschreibt, wann das berechnete Petrinetz das gegebene LTS löst und geht auf Arbeiten von Ehrenfeucht und Rozenberg [ER90] zurück. Hierfür gibt es sogenannte

Separierungsprobleme, die von Regionen gelöst werden müssen. Davon gibt es zwei Arten: Bei einem *Beschriftungs-Zustands-Separierungsproblem (event-state separation problem; ESSP)* muss in einem Zustand $s \in S$ des LTS die Beschriftung $e \in \Sigma$ verhindert werden, da s keine ausgehende Kante mit Beschriftung e hat. Dies bedeutet, das eine Stelle gesucht wird, die Transition e deaktiviert, wenn der Zustand s eingenommen wird. Ein *Zustands-Separierungsproblem (state separation problem; SSP)* besteht aus zwei verschiedenen Zuständen $s, s' \in S$. Diese beiden Zustände sollen in einer Lösung durch verschiedene Markierungen repräsentiert werden.

3 Gezielte Synthese von Petrinetz-Teilklassen

Es existieren schon Algorithmen zur Petrinetz-Synthese. Beispielsweise kann das Problem als lineares und ganzzahliges Ungleichungssystem ausgedrückt werden [BBD15]. In der Dissertation wird dieser Ansatz erweitert, um zusätzliche Eigenschaften für das synthetisierte Petrinetz zu garantieren. Anstatt ein beliebiges Petrinetz zu berechnen, wird auf diese Art auf eine gegebene Teilklasse von Petrinetzen abgezielt. Beispiele für solche Teilklassen sind schlichte Petrinetze, bei denen keine Kantengewichte größer als eins erlaubt sind, schlingenfreie Petrinetze, bei denen zwischen einem Paar aus Transition und Stelle eine Kante in höchstens einer Richtung existieren darf, und k-beschränkte Petrinetze, bei denen jede Stelle höchstens k Token in jeder erreichbaren Markierung enthalten darf. Diese Teilklassen werden durch Ungleichungen charakterisiert, beziehungsweise durch prädikatenlogische Formeln, die über ganzen Zahlen interpretiert werden.

Indem diese Formeln zu den Gleichungssystemen hinzugefügt werden, die mittels des bereits bekannten Ansatz aufgestellt werden, wird garantiert, dass die berechneten Petrinetze zur gewünschten Teilklasse gehören. Außerdem lassen sich Teilklassen kombinieren. Beispielsweise werden schlichte und k-beschränkte Petrinetze synthetisiert, indem die Formeln für beide Teilklassen mit in die Gleichungssysteme aufgenommen werden.

In der Arbeit werden eine Reihe von bekannten Teilklassen von Petrinetzen identifiziert, die auf diese Weise ausgedrückt werden können. Es werden aber auch Beispiele für Teilklassen gegeben, die nicht mit diesem Ansatz umsetzbar sind.

4 Kleinste Lösbare Überapproximation

Nicht jedes LTS kann durch ein Petrinetz gelöst werden. Außerdem sind die meisten Teilklassen ausdrucksschwächer als Petrinetze im Allgemeinen, wodurch noch weniger LTS synthetisiert werden können. Der Algorithmus erkennt diese Situation daran, dass ein Separierungsproblem bzw. das hierfür aufgestellte Gleichungssystem unlösbar ist. In der Dissertation wird ein Algorithmus vorgestellt, der diesen Fall behandelt. Anstatt als Ergebnis „unlösbar“ auszugeben, approximiert dieser Algorithmus das gegebene LTS.

Das Ergebnis des Approximierungs-Algorithmus ist eine kleinste Überapproximation der Eingabe. Hierfür wird eine Partialordnung für LTS namens LTS-Homomorphismus definiert. Diese Partialordnung hat Ähnlichkeit zur bekannten Simulations-Partialordnung. Anhand dieser Ordnung kann das Ergebnis des Algorithmus formal definiert werden: Alle LTS, die größer als das Eingabe-LTS A sind, sind Überapproximationen desselben. Nur manche dieser Überapproximationen sind Petrinetz-lösbar. Der Algorithmus berechnet die kleinste dieser Petrinetz-lösbaren Überapproximationen, welche Approx(A) genannt wird. Die Eindeutigkeit von Approx(A) wird in der Arbeit gezeigt.

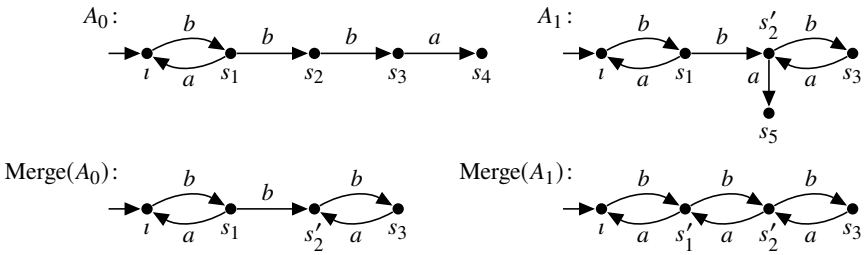

Abb. 2: Ein Beispiel des Approximierungs-Algorithmus

Ein Beispiel für den Algorithmus wird in Abbildung 2 gezeigt. Das LTS A_0 soll approximiert werden. Das LTS A_1 ist ein Zwischenergebnis. Der Algorithmus ist ein Fixpunkt-Algorithmus, bei dem zwei Schritte immer wiederholt werden, bis keine weiteren Änderungen mehr notwendig sind.

Bei Merge(A) werden unlösbare SSP-Instanzen behandelt, also Zustände, denen zwingend dieselbe Markierung zugeordnet werden muss. Hierzu werden die Zustände zu einem einzelnen Zustand zusammengefasst. Im Beispiel wurden im LTS Merge(A_0) die Zustände s_2 und s_4 zu s_2' zusammengefasst, da die zugehörige SSP-Instanz unlösbar ist.

Im zweiten Schritt wird Merge(A_0) zu einem LTS A_1 expandiert, indem unlösbare ESSP-Instanzen eliminiert werden. Im Beispiel kann im Zustand s_2' die Beschriftung a nicht verhindert werden und wird daher zum LTS hinzugefügt. Dies geschieht, indem ein neuer Zustand s_5 eingeführt wird und als Ziel der aus s_2' ausgehenden Kante mit Beschriftung a verwendet wird.

Nun wird der erste Schritt mit A_1 wiederholt und Merge(A_1) bestimmt. Im abgebildeten Beispiel werden hierbei die Zustände s_1 und s_5 zu s_1' vereinigt. Anschließend gibt es keine weiteren unlösbaren Separierungsprobleme mehr. Das resultierende LTS ist Petrinetz-lösbar und erlaubt u.a. jegliches Verhalten, das schon in A_0 möglich war.

Dieser Algorithmus ist auf einige Petrinetz-Teilklassen anwendbar, während für andere Teilklassen die kleinste Überapproximation ein unendlich großes LTS sein kann und dann daher nicht berechnet werden kann.

Die vorgestellte Überapproximation ist minimal nach einer in der Dissertation definierten

Partialordnung. Anhand eines Beispieles wird gezeigt, dass der Algorithmus nicht die kleinste Überapproximation laut Sprachinklusion liefert. Hiermit ist gemeint, dass zu einem gegebenen LTS A die Sprache $L(A)$ approximiert werden soll. Minimalität bedeutet hierbei, dass das berechnete Petrinetz N eine kleinere Sprache hat als jede andere Überapproximation von A, also $\forall N' \colon L(A) \subseteq L(N') \Rightarrow L(N) \subseteq L(N')$ erfüllt. Es wird auch gezeigt, wie die kleinste Überapproximation gemäß Sprachinklusion bestimmt werden kann.

5 Modale Spezifikationen

Ein LTS spezifiziert das exakte Verhalten eines Systems, ohne dass noch Variationsmöglichkeiten bestehen. Modale Spezifikationen erlauben eine weniger exakte Spezifikation. Eine modale Spezifikation beschreibt eine Menge von LTS. Jedes dieser LTS implementiert die Spezifikation.

In der Dissertation werden drei Arten von modalen Spezifikationen betrachtet: Modale Transitionssysteme (MTS) [La89], der modale μ-Kalkül [AN01; Ko83] und eine syntaktische Teilmenge des μ-Kalküls, die konjunktiver ν-Kalkül heißt. Im folgenden wird der μ-Kalkül kurz eingeführt.

Der μ-Kalkül besteht aus Formeln, die wie folgt aufgebaut sind:

$$\beta ::= \texttt{true} \mid \texttt{false} \mid \beta_1 \wedge \beta_2 \mid \beta_1 \vee \beta_2 \mid \neg\beta_1 \mid \langle a \rangle \beta_1 \mid [a]\beta_1 \mid \nu X.\beta_1 \mid \mu X.\beta_1 \mid X$$

Der μ-Kalkül umfasst die Aussagenlogik, also die Konstanten `true` und `false` und die Konnektoren Konjunktion $\wedge$, Disjunktion $\vee$ und Negation $\neg$. Formeln werden in Zuständen eines LTS interpretiert. Es gibt eine existentielle Modalität $\langle a \rangle \beta$ und eine universelle Modalität $[a]\beta$. Die existentielle Modalität $\langle a \rangle \beta$ fordert, dass der aktuelle Zustand eine ausgehende Kante hat, die mit a beschriftet ist und zu einem Zustand führt, der die Formel β erfüllt. Die universelle Modalität macht die gleiche Aussage für alle ausgehenden Kanten mit Beschriftung a. Da Petrinetze deterministisch sind, gibt es in einem Erreichbarkeitsgraphen höchstens eine ausgehende Kante für jede Beschriftung. Somit bedeutet $\langle a \rangle \beta$ in diesem Kontext, dass es eine a-Kante geben muss. Im Gegensatz dazu fordert $[a]\beta$ diese Kante nicht, sondern macht nur darüber Aussagen, was gelten muss, falls diese a-Kante vorhanden ist. Die Formel $[a]\texttt{false}$ sagt z.B. aus, dass es keine ausgehende Kante mit Beschriftung a geben darf, da kein Zustand die Formel `false` erfüllen kann.

Weiterhin gibt es Variablen X, den größten Fixpunkt ν und den kleinsten Fixpunkt μ. Mit diesen kann unendliches Verhalten beschrieben werden. Beispielsweise bedeutet $\langle a \rangle X$, dass nach einer Kante mit Beschriftung a die Variable X erfüllt werden muss. Diese freie Variable kann durch einen Fixpunkt-Operator gebunden werden. So bedeutet $\nu X.\langle a \rangle X$, dass a unendlich oft hintereinander möglich ist. In einem endlichen System fordert dies also eine Schleife.

Der Unterschied zwischen den beiden Fixpunkt-Operatoren ist, dass der größte Fixpunkt ν unendlich oft durchlaufen werden darf, während der kleinste Fixpunkt μ nur endliche

Rekursion erlaubt. Beispielsweise bedeutet $\mu X.\langle a\rangle X \vee \langle b\rangle \texttt{true}$, dass nach endlich vielen a-Kanten, eine ausgehende Kante mit Beschriftung b existieren muss. Im Gegensatz hierzu würde $\nu X.\langle a\rangle X \vee \langle b\rangle \texttt{true}$ auch durch unendlich viele Wiederholungen der Beschriftung a erfüllt werden.

Ein LTS A *erfüllt* bzw. *implementiert* eine Formel β, geschrieben $A \models \beta$, falls der Initialzustand des LTS die Formel β erfüllt, wie gerade skizziert wurde.

Der μ-Kalkül ist ein sehr ausdrucksmächtiger Formalismus. Er umfasst beispielsweise die bekannten Logiken LTL, CTL, and CTL* [CGR11], als auch modale Transitionssysteme. Letzteres wird in der Dissertation gezeigt, indem ein syntaktisches Fragment des μ-Kalküls definiert wird, die konjunktiver ν-Kalkül heißt, und gezeigt wird, dass dieses Fragment im Wesentlichen äquivalent zu modalen Transitionssystemen ist.

6 Petrinetz-Realisierungen Modaler Spezifikationen

Eine modale Spezifikation definiert eine Menge von LTS, die es implementieren. Ein Petrinetz erzeugt einen Erreichbarkeitsgraphen, welches ein LTS ist. Diese beiden bereits existierenden Begriffe werden in der Arbeit zu einem neuen Begriff verbunden: Ein Petrinetz *realisiert* eine modale Spezifikation, falls sein Erreichbarkeitsgraph diese Spezifikation implementiert.

Nun kann das *Realisierungsproblem* formuliert werden: Gegeben eine modale Spezifikation, gibt es eine Petrinetz-Realisierung? Verschiedene Varianten dieses Problems können betrachtet werden. In der Arbeit werden dafür drei Arten von modalen Spezifikationen vorgestellt. Es kann außerdem gefordert werden, dass das Petrinetz zu einer gegebenen Teilklasse gehört, wie beispielsweise schlingen-freie und k-beschränkte Petrinetze.

In der Dissertation wird für das Realisierungsproblem noch gefordert, dass das Petrinetz nur endlich viele erreichbare Markierungen hat. Für den unendlichen Fall mit schlingen-freien Petrinetze wurde bereits die Unentscheidbarkeit gezeigt [Fe05].

Es wird in der Dissertation gezeigt, dass das Realisierungsproblem auch für endliche Erreichbarkeitsgraphen unentscheidbar ist. Dieses Ergebnis kann auch für schlingen-freie Petrinetze und einige andere Teilklassen gezeigt werden. Es gilt auch für sämtliche betrachteten Arten von modalen Spezifikationen, also nicht nur für den sehr ausdrücksmächtigen μ-Kalkül, sondern auch für modale Transitionssysteme und den konjunktiven ν-Kalkül, die nicht einmal Disjunktionen ausdrücken können und somit sehr eingeschränkt sind. Der Beweis basiert auf einer Simulation von Zwei-Zähler-Maschinen.

Um Entscheidbarkeit zu erreichen, werden die betrachteten Petrinetze eingeschränkt, und zwar auf die Klasse der k-beschränkten Petrinetze, wobei k eine weitere Eingabe an den Algorithmus ist, also a priori gegeben ist. Durch diese Einschränkung gibt es nur noch endlich viele mögliche Petrinetze, die in Frage kommen können: Im Petrinetz können nur

Zahlen bis k auftauchen und das Alphabet Σ, also die Transitionsmenge des Petrinetzes, ist durch die Spezifikation bereits festgelegt. Diese Petrinetze könnten theoretisch alle darauf geprüft werden, ob sie die Spezifikation implementieren, jedoch wäre der Aufwand dieses Ansatzes riesig: Eine Abschätzung der Anzahl in Frage kommender Petrinetze ergibt $2^{(k+1)^{1+2|\Sigma|}}$, wobei für jedes dieser Petrinetze noch der Erreichbarkeitsgraph berechnet und geprüft werden muss, ob dieser die Spezifikation implementiert.

Daher wird in der Arbeit ein direkter Ansatz verfolgt. Es soll ein LTS konstruiert werden, das Petrinetz-lösbar ist und die Spezifikation implementiert. Falls ein Zustand eine Formel $\langle a \rangle \beta$ erfüllen muss, aber noch keine ausgehende Kante mit der Beschriftung a hat, dann wird eine solche Kante zu einem neuen Zustand hinzugefügt. Die anderen Operatoren des μ-Kalküls können auf ähnliche Weise behandelt werden. Hierfür wird auf einen Algorithmus von Stirling und Walker [SW89; SW91] zur lokalen Modellprüfung zurückgegriffen. Normalerweise schlägt dieser Algorithmus fehl, wenn ein Zustand $\langle a \rangle \beta$ erfüllen soll, aber keine passende ausgehende Kante hat. Dies ist jedoch genau die Information, die zum Erweitern des LTS benötigt wird.

Ein auf diese Weise konstruiertes LTS ist nicht notwendigerweise Petrinetz-lösbar. Um dies sicherzustellen, wird wieder die kleinste Überapproximation eingesetzt. Der Algorithmus besteht darin, die beiden Schritte, Erweiterung des LTS und Überapproximation, zu wiederholen bis eine Petrinetz-lösbare Implementierung gefunden wurde oder die Erweiterung fehlschlägt. Ein Fehlschlag passiert beispielsweise, wenn ein Zustand eine Kante mit der Beschriftung a hat, aber $[a]\texttt{false}$ erfüllen soll.

Da nur die Überapproximation spezifisch für Petrinetze ist, unterstützt dieser Algorithmus alle Teilklassen von Petrinetzen, für die eine Überapproximation möglich ist.

Der vorgestellte Algorithmus und seine Implementierung werden am Ende der Dissertation für eine Fallstudie eingesetzt. In dieser Fallstudie wird Dijkstras bekanntes Philosophenproblem als LTS und als modale Spezifikation umgesetzt.

7 Fazit

Die hier beschriebene Dissertation besteht aus zwei Teilen. Im ersten Teil wird Petrinetz-Synthese aus LTS untersucht. Nach der Einführung der Grundlagen und existierender Algorithmen wird einer dieser Algorithmen erweitert, um auf Teilklassen von Petrinetzen zielen zu können. Dies bedeutet, dass weitere Anforderungen an das Petrinetz gestellt werden, beispielsweise können Kantengewichte verboten werden. Als nächstes wird ein Ansatz zur Approximation vorgestellt. Nicht jedes LTS kann in ein Petrinetz überführt werden. In einem solchen Fall kann das LTS strukturell verändert werden, um es Petrinetz-lösbar zu machen. Hierzu wird ein Algorithmus zur kleinsten Überapproximation eines gegeben LTS vorgestellt. Dieser Algorithmus funktioniert auch für manche der zuvor untersuchten

Teilklassen, während für andere Teilklassen die kleinste Überapproximation unendlich groß werden kann.

Im zweiten Teil der Dissertation geht es um Petrinetz-Synthese aus modalen Spezifikationen. Hierzu werden MTS, der μ-Kalkül und der ν-Kalkül zunächst eingeführt und eine Beziehung zwischen ihnen hergestellt. Dann wird gezeigt, dass die Petrinetz-Synthese aus einer sehr ausdrucksschwachen Spezifikationssprache, den MTS, unentscheidbar ist. Als nächstes werden k-beschränkte Petrinetze betrachtet, für die das Synthese-Problem wieder entscheidbar ist, und ein Synthese-Algorithmus wird vorgestellt.

Alle Algorithmen, die in der Dissertation entwickelt wurden, wurden in einem Open-Source-Werkzeug implementiert. Diese Implementierung wurde für eine Fallstudie herangezogen, in der das bekannte Philosophenproblem mit modalen Spezifikationen modelliert und anschließend in ein Petrinetz synthetisiert wurde. Hierbei zeigte sich, dass die unabhängigen Teile eines verteilten Systems mit modalen Spezifikationen gut abgebildet werden können.

Eine der offenen Fragen ist die Komplexität der untersuchten Probleme und der vorgestellten Algorithmen. Hierzu gibt es bereits erste Ansätze, aber noch keine abschließenden Antworten. Eine andere Richtung für weitere Forschung wäre es, den Ansatz zur Realisation von μ-Kalkül-Formeln auf andere Spezifikationssprachen zu erweitern. Beispielsweise ist die monadische Logik zweiter Ordnung (monadic second order logic) ausdrucksmächtiger als der μ-Kalkül und kann beispielsweise ausdrücken, dass zwei Pfade zum gleichen Zustand führen müssen. Eine solche Anforderung könnte durch Zusammenfassen von Zuständen behandelt werden. Somit sollte auch die Synthese entsprechender Formeln möglich sein. Allerdings fehlt noch ein Ansatz, um ein gegebenes LTS um benötigtes Verhalten zu erweitern. Die Grundidee für einen Algorithmus existiert jedoch schon.

Literatur

[Aa16] van der Aalst, W. M. P.: Process Mining - Data Science in Action, Second Edition. Springer, 2016.

[AN01] Arnold, A.; Niwiński, D.: Rudiments of μ-calculus. North Holland, 2001.

[BBD15] Badouel, E.; Bernardinello, L.; Darondeau, P.: Petri Net Synthesis. Springer, 2015.

[BCD02] Badouel, E.; Caillaud, B.; Darondeau, P.: Distributing Finite Automata Through Petri Net Synthesis. Formal Aspects of Computing 13/6, S. 447–470, 2002.

[BD11] Best, E.; Darondeau, P.: Petri Net Distributability. In (Clarke, E. M.; Virbitskaite, I.; Voronkov, A., Hrsg.): PSI 2011. Bd. 7162. LNCS, Springer, S. 1–18, 2011.

[CGR11] Cranen, S.; Groote, J. F.; Reniers, M. A.: A linear translation from CTL* to the first-order modal μ-calculus. Theoretical Computer Science 412/28, S. 3129–3139, 2011.

[ER90] Ehrenfeucht, A.; Rozenberg, G.: Partial (Set) 2-Structures. Part I: Basic Notions and the Representation Problem and Part II: State Spaces of Concurrent Systems. Acta Inf. 27/4, S. 315–368, 1990.

[Fe05] Feuillade, G.: Spécification logique de réseaux de Petri, Diss., Université de Rennes I, 2005, URL: http://www.irisa.fr/s4/download/papers/Feuillade-these2005.pdf.

[GGS13] van Glabbeek, R. J.; Goltz, U.; Schicke-Uffmann, J.: On Characterising Distributability. Logical Methods in Computer Science 9/3, 2013.

[Ko83] Kozen, D.: Results on the Propositional mu-Calculus. Theoretical Computer Science 27/3, S. 333–354, 1983.

[Kř17] Křetínský, J.: 30 Years of Modal Transition Systems: Survey of Extensions and Analysis. In (Aceto, L.; Bacci, G.; Bacci, G.; Ingólfsdóttir, A.; Legay, A.; Mardare, R., Hrsg.): Models, Algorithms, Logics and Tools. Bd. 10460. LNCS, Springer, S. 36–74, 2017.

[La89] Larsen, K.: Modal Specifications. In (Sifakis, J., Hrsg.): AVMFSS. Bd. 407. LNCS, Springer, S. 232–246, 1989.

[PWM03] Pinney, J.; Westhead, D.; McConkey, G.: Petri Net representations in systems biology. Biochemical Society Transactions 31/6, S. 1513–1515, 2003, ISSN: 0300-5127.

[Sc18] Schlachter, U.: Petri Net Synthesis and Modal Specifications, Diss., Carl von Ossietzky Universität Oldenburg, 2018, URL: http://nbn-resolving.de/urn:nbn:de:gbv:715-oops-38362.

[SW89] Stirling, C.; Walker, D.: Local Model Checking in the Modal Mu-Calculus. In (Diaz, J.; Orejas, F., Hrsg.): TAPSOFT'89 (CAAP'89). Bd. 351. LNCS, Springer, S. 369–383, 1989.

[SW91] Stirling, C.; Walker, D.: Local Model Checking in the Modal mu-Calculus. Theoretical Computer Science 89/1, S. 161–177, 1991.

Uli Schlachter wurde am 12. Dezember 1989 in Oldenburg geboren. Nach dem Abitur an der Kooperativen Gesamtschule Rastede im Jahr 2009 begann er ein Bachelorstudium der Informatik an der Carl von Ossietzky Universität Oldenburg. An dieses schloss er ein Masterstudium an, welches er 2014 mit Auszeichnung abschloss. Anschließend arbeitete er dort in der theoretischen Informatik als wissenschaftlicher Mitarbeiter in der Abteilung für parallele Systeme bei Prof. Dr. Eike Best, wo er im November 2018 promovierte.