Rekurrenz in Selbststabilisierung: Theorie, Verifikation und Anwendung[1]

Oday Jubran[2]

Zusammenfassung: Selbststabilisierung ist ein Konzept der Fehlertoleranz, welches eine Wiederherstellung eines Systems nach Ausfall durch transiente Fehler garantiert, ohne von sich aus in einen solchen zu geraten. In der Literatur wird Selbststabilisierung meist in Hinblick auf die Wiederherstellung einer sicheren Ausführung untersucht. In dieser Arbeit wird das Konzept der Selbststabilisierung generalisiert, um mehrere Eigenschaftsarten abzudecken, die die Betrachtung der Wiederherstellung eines Systems hin zu einer möglichst hohen Servicequalität ermöglichen. Hierzu wird das Design, die Analyse und das Re-Engineering von Systemen, die bekannte Probleme in verteilten Systemen lösen, angewendet. Die Probleme sind: die Zeitschlitzzuteilung in Zeitmultiplexverfahren sowie der wechselseitige Ausschluss und eine erweiterte Version davon. Zusätzlich wird eine automatische Verifikationsmethode präsentiert, die die Verifikation solcher Eigenschaften in selbststabilisierenden Systemen erleichtert.

Stichwörter: Selbststabilisierung, Zuverlässigkeit, Rekurrenz, verteilter Algorithmus, Verifikation

1 Einführung

"Ich dachte, dass ich in [Di74] drei Lösungen veröffentlicht hätte, jedoch später lernte ich, dass ich auch drei Probleme veröffentlicht hatte, da die Programme ohne einen Beweis ihrer Korrektheit angegeben worden sind." – Dijkstra [Di86]. Diese Aussage beschreibt, wie schwierig es war für die Algorithmen, die Dijkstra in seiner wegweisenden Arbeit [Di74] 12 Jahre zuvor zeigte, einen Korrektheitsbeweis zu finden. Diese Algorithmen – zu diesem Zeitpunkt – adressierten Systeme mit mehreren Prozessen, die keinen gemeinsamen Speicher teilen. Das Ziel der Algorithmen ist es, die Prozesse in die Lage zu versetzen ein globales Übereinkommen oder irgendeine Art von Synchronisation zwischen ihnen zu erreichen, ohne die Notwendigkeit eines gemeinsamen Speichers oder sogar einer Voreinstellung. Deswegen geben beide [Di74, Di86] eine positive Antwort auf die Frage, ob es möglich ist, für teilnehmenden Komponenten in einem verteilten System, auf eine Spezifikation übereinzukommen, ohne die Existenz einer globalen Kontrolle oder einer Voreinstellung.

Diese Arbeit behandelt ein bestimmtes Konzept der Fehlertoleranz in verteilten Systemen, dem von Dijkstra in seinem Werk [Di74] der Weg bereitet wurde. In diesem Konzept, wird von einem System verlangt sich von Ausfällen durch flüchtige, transiente, seltene Fehler oder dynamischen Topologieänderungen wiederherzustellen und in der Lage zu sein schließlich korrekt von einer beliebigen anfänglichen Konfiguration (globaler

[1] Englischer Originaltitel: "Recurrence in Self-Stabilization: Theory, Verification, and Application"

[2] Carl von Ossietzky Universität Oldenburg, Department für Informatik. oday.jubran@gmail.com

Zustand) auszuführen, wenn genügend Zeit zur Wiederherstellung gegeben wurde. Dieses Konzept nennt sich *Selbststabilisierung*, und wird modelliert – in Bezug auf Zustand und Ausführungen – durch zwei Eigenschaften:

1. *Konvergenz*: Wenn ein System sich in einem unzulässigem Zustand befindet, dann garantiert das System in endlicher Zeit einen erlaubten Zustand zu erreichen.
2. *Abgeschlossenheit*: Wenn ein System sich in einem erlaubten Zustand befindet, wird es von selbst niemals einen unzulässigen Zustand erreichen.

Verteilte Systeme sind in der Regel Fehlern ausgesetzt. Als Beispiele, Rauschen kann Funksignale überlagern und die Kommunikation in drahtlosen Sensornetzen stören. Hinzufügen oder Entfernen von Komponenten in einem Netzwerk kann die Topologie und das Routing beeinflussen. Somit hat die Selbststabilisierung einen signifikanten Einfluss in verteilten Systemen, da in den meisten Fällen die Komponenten eines Systems nicht über die volle Kenntnisse der Konfiguration verfügen.

1.1 Problemstellung

Die klassische Selbststabilisierung bezieht sich auf Eigenschaften die über Konfigurationen definiert sind und eine Konfiguration gilt als erlaubt, genau dann wenn sie die gegebene Eigenschaft erfüllt. So garantieren die Konvergenz- und Abschlussaspekte, dass eine ein Ausführungssuffix erreicht wird, in dem eine Eigenschaft von allen Konfigurationen erfüllt wird. Solche Eigenschaften sind in der Praxis *Sicherheitseigenschaften*. Von diesem Zeitpunkt an wird, die Stabilisierung eines Systems oder die Wiederherstellung von transienten Fehler an Sicherheitseigenschaften festgemacht, die über Konfigurationen definiert sind. Darüber hinaus, ist das Zeit-Effizienz-Problem in der klassischen Selbststabilisierung tatsächlich eine Frage die Konvergenzzeit zu reduzieren, um Sicherheit zu erreichen. Beachten Sie, dass einige andere Konzepte der Selbststabilisierung Spezifikationen über das Systemverhalten betreffen, jedoch folgen diese nicht unbedingt einem bestimmten Muster und sie untersuchen ungewöhnliche Fälle.

In der Selbststabilisierung gibt es die Annahme, dass Fehler transient sind. Zusätzlich, kann ein selbststabilisierendes System praktisch sein z.B. durch das Erreichen von hoher Verfügbarkeit unter der Annahme, dass Fehler selten aufkommen. Mit diesen beiden Annahmen gibt es implizit genug Zeit für jedes Verhalten eines Systems zu stabilisieren und eine erwünschte Servicequalität zu erreichen, bevor der nächste Fehler das System trifft. Diese Tatsachen begrenzen die Untersuchungen der Selbststabilisierung auf Sicherheitseigenschaften, die über Konfigurationen definiert sind und vernachlässigen andere Aspekte der Servicequalität eines Systems.

Die Hauptfrage welche in dieser Arbeit angegangen wurde, ist, wie Selbststabilisierung Eigenschaften der Performanz des Systems abdecken kann: die Menge an Ausgabe die das System pro Zeiteinheit produziert, oder Effizienz: Die Menge an Ausgabe in Abhängigkeit der Eingabe, und Servicequalität? In anderen Worten, wie kann Selbststabilisierung,

um andere Arten von Eigenschaften abzudecken, verallgemeinert werden, in welchen Wiederherstellung nicht nur Sicherheit, sondern einen gewisse Servicequalität zu erreichen bedeutet.

1.2 Beitrag

Der Beitrag dieser Arbeit ist wie folgt. Zuerst wird ein generalisiertes Modell der Selbststabilisierung, das Performanzaspekte abdeckt, präsentiert. Das Modell beinhaltet ein allgemeines Schema der Eigenschaften, die wir als *Rekurrenzeigenschaften* bezeichnen. Dieses Modell deckt beides, Diskrete und Echtzeitsysteme, ab. Zweitens, Selbststabilisierung in Bezug auf Rekurrenzeigenschaften wird auf drei Probleme der verteilten Berechnung angewandt: (1) *Wechselseitiger Ausschluss*, (2) eine erweiterte Version davon, nämlich *eindeutige Prozessauswahl*, und (3) *Zeitmultiplexverfahren (TDMA) Zeitschlitzzuteilung* in einer gegebenen Umgebung im Einklang mit der *Europäischen Norm* EN 54-25 [DI05]. Schließlich wird ein automatischer Verifikationssansatz für selbststabilisierende Systeme unter Beachtung von Rekurrenzeigenschaften, einschließlich einer Fallstudie, gezeigt.

1.3 Inhalt

Der Rest dieses Artikels ist wie folgt strukturiert. Abschnitt 2 zeigt Rekurrenzeigenschaften und das verallgemeinerte Konzept der Selbststabilisierung. Als nächstes, Abschnitt 3 wendet das verallgemeinerte Konzept auf das Problem des wechselseitigen Ausschluss an. Dann zeigt Abschnitt 4 die begründete und eindeutige Prozessauswahlprobleme und wendet das verallgemeinerte Konzept an um diese zu lösen. Hierauf betrachtet Abschnitt 5 das TDMA Zeitschlitzzuteilungsproblem im verallgemeinerten Konzept. Danach zeigt Abschnitt 6 den automatischen Verifikationssansatz. Schließlich legt Abschnitt 7 die Schlussfolgerung dar.

2 Rekurrenz in der Selbststabilisierung

Dieser Abschnitt zeigt die neue Bedingung, die im Rahmen der Selbststabilisierung betrachtet wird. Diese Bedingung basiert auf einem Kernmaß, das auf endliche Ausführungen von Systemen (oder Algorithmen), wie folgt angewandt wird: gegeben eine endliche Ausführung Ξ, dieses Maß ist das Verhältnis der Konfigurationen, die eine Bedingung con in Ξ erfüllen. Dieses Verhältnis wird die **Rekurrenz** der Bedingung in der Ausführung genannt, gegeben durch $\mathsf{Rek}_{\mathsf{con}}(\Xi)$.

Zum Beispiel, sei con eine Bedingung definiert über die Konfigurationen einer Topologie. Als Notation, bezeichne $\underline{\gamma}$, dass eine Konfiguration γ con erfüllt. Für die folgende endliche Ausführung: $\Xi : \underline{\gamma_0}, \underline{\gamma_1}, \gamma_2, \gamma_3, \underline{\gamma_4}, \gamma_5, \underline{\gamma_6}, \gamma_7, \gamma_9, \underline{\gamma_9}, \gamma_{10}, \underline{\gamma_{11}}$, ist die Rekurrenz von con in Ξ das Ergebnis einer Divisionsoperation: $\mathsf{Rek}_{\mathsf{con}}(\Xi) = \frac{6}{12} = 0.5$.

Mit dem Begriff der Rekurrenz über endlichen Ausführungen definiert, wird die Eigenschaft über möglicherweise unendliche Ausführungen wie folgt definiert: gegeben eine unendliche Ausführung Ξ', die Eigenschaft zeigt, dass jedes Ausführungspräfix von Ξ' zumindest Δ Rekurrenz einer Bedingung con besitzt. Die Eigenschaft wird $\mathbf{con_\Delta}$ genannt.

Als nächstes wird die Definition der Selbststabilisierung verallgemeinert, um die Eigenschaft con_Δ wie folgt abzudecken: Man sagt eine Ausführung $\gamma_0, \gamma_1, \ldots$ hat eine **con_Δ-Konvergenzzeit** von **c** schritten, genau dann wenn c die kleinste Zahl ist, so dass das Ausführungssuffix $\gamma_c, \gamma_{c+1}, \ldots$ von Ξ con_Δ erfüllt. Zum Beispiel die folgende Ausführung hat eine con_Δ-Konvergenzzeit von 5.

$$\gamma_0, \gamma_1, \gamma_2, \gamma_3, \gamma_4, \underline{\gamma_5}, \gamma_6, \gamma_7, \gamma_8, \underline{\gamma_9}, \gamma_{10}, \gamma_{11}, \gamma_{12}, \underline{\gamma_{13}}, \ldots$$

Man spricht bei einem System von einer con_Δ-Konergenzzeit von c schritten, genau dann wenn c die maximale (dass heißt der schlimmste Fall) con_Δ-Konvergenzzeit unter allen Ausführungen des Systems ist. Abbildung 1 illustriert diese Intuition. Merke dass die klassische Selbststabilisierung sich widerspiegelt, durch setzen von $\Delta = 1.0$.

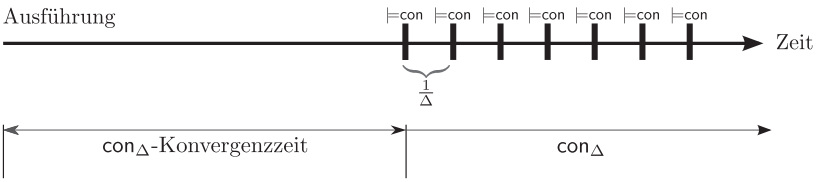

Abb. 1: con_Δ-Konvergenz

Neben der con_Δ-Konvergenz, ist die con_Δ-Aufwärmzeit vorhanden, um die geleistete Dienstqualität während der Konvergenz eines Systems zu betrachten. Man sagt eine Ausführung $\gamma_0, \gamma_1, \ldots$ hat eine **con_Δ-Aufwärmzeit** von w schritten, genau dann wenn w die minimale Zahl, so dass für jedes $i \geq w$, die Rekurrenz von con im Ausführungspräfix $\gamma_0, \ldots, \gamma_i$ von Ξ (das heißt $\mathsf{Rek}_{\mathsf{con}}(\gamma_0, \ldots, \gamma_i)$) größer oder gleich Δ ist. Abbildung 2 stellt die Aufwärmzeit dar für eine Ausführung Ξ.

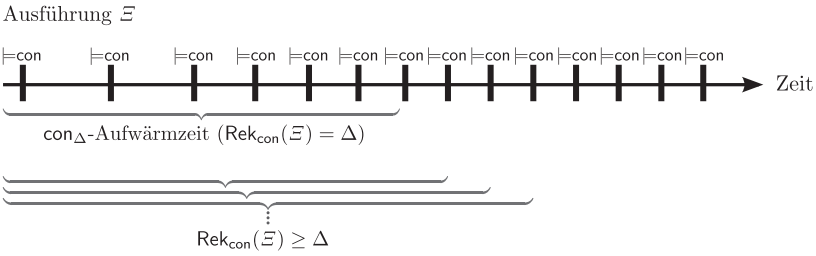

Abb. 2: con_Δ-Aufwärmzeit

Das gleichen Konzept wird auch für Echtzeitsysteme modelliert, in welchen Evolutionen über die Zeit eigenständige Ausführungen ersetzen. Die Rekurrenz einer Bedingung in einer Evolution über ein endliches Zeitintervall, ist die akkumulierte Zeit in welcher con erfüllt ist, dividiert durch das gesamte Intervall.

3 Wechselseitiger Ausschluss

Das verallgemeinerter Konzept der Selbststabilisierung wird angewandt zum Entwurf eines effizienten selbststabilisierenden Algorithmus in Bezug auf *wechselseitigen Ausschluss* (Engl. *Mutual Exclusion*, kurz *Mutex*), welches ein bekanntes Problem aus dem Gebiet der Selbststabilisierung ist. Die Spezifizierung des wechselseitigen Ausschluss ist wie folgt: gegeben eine Menge von mehreren Prozessen die eine Aufgabe nebenläufig ausführen, dann besteht Mutex aus zwei Eigenschaften: Ersten, wenn einem Prozess die Ausführung einer bestimmten Aktion freigegeben ist, dann sollte diese andere Prozesse nicht freigegeben sein. Zweitens, jedem Prozess sollte es gestattet sein, die Aktion unendlich oft auszuführen. Die zweite Eigenschaft wird *Fairness* genannt.

Im Rahmen dieser Arbeit, gibt die Effizienz eine hohe Rekurrenz des gewähren eines Privilegs zu Prozessen wieder, was eine hohe Servicequalität des Algorithmus impliziert. Drei Algorithmen wurden entworfen, um ein Abwägen zwischen folgenden Komplexitätsmaßen zu messen: die Konvergenzzeitkomplexität bzgl. wechselseitigen Ausschluss, die Konvergenzzeit bzgl. Δ Rekurrenz auf gewähren eines Privilegs und der Platzbedarf. Die drei Algorithmen basieren auf Algorithmus 3 aus [BPV08], welcher ein synchrones Inkrementierungssystem mit einem endlichen Bereich von ganzzahligen Werten umsetzt. Das Ziel dieses Systems ist, in jedem synchronen Schritt, wiederholt eine Variable hochzuzählen, die einem jeweiligen Prozess gehört, während die Werte aller Prozesse gleich bleiben sollen. Dem Ansatz von [DG13] folgend, kann Mutex erreicht werden, in dem man jedem Prozess ein Privileg gewährt, wenn seine Kennung gleich dem momentanen Ganzzahlwert ist. Tabelle 1 fasst die Ergebnisse zusammen, wobei $\mathsf{diam}(\mathscr{G})$ den Durchmesser eines Graphen, n die Größe des Graphen, und *priv* die Bedingung das ein Prozess privilegiert ist angibt.

	Algorithmus A	Algorithmus B	Algorithmus C
Konvergenzzeit bzgl. Mutex	$\mathsf{diam}(\mathscr{G})-1$	$\lceil \mathsf{diam}(\mathscr{G})/2 \rceil -1$	$\lceil \mathsf{diam}(\mathscr{G})/2 \rceil -1$
Rekurrenz Δ	1.0	$\mathsf{n}/(\mathsf{n}+\lceil \mathsf{diam}(\mathscr{G})/2 \rceil -1)$	1.0
$priv_{\Delta}$-Konvergenzzeit	$2 \cdot \mathsf{diam}(\mathscr{G})$	$\mathsf{max}\{(\lceil 2.5 \cdot \mathsf{diam}(\mathscr{G}) \rceil -1), (\mathsf{n} + \lceil \mathsf{diam}(\mathscr{G})/2 \rceil -2)\}$	$\lceil 2.5 \cdot \mathsf{diam}(\mathscr{G}) \rceil -1$
Platz	$\mathsf{n}+\mathsf{diam}(\mathscr{G})$	$\mathsf{n} + \lceil 1.5 \cdot \mathsf{diam}(\mathscr{G}) \rceil -1$	$\mathsf{n}+\lceil 1.5 \cdot \mathsf{diam}(\mathscr{G}) \rceil -1$

Tab. 1: Zeit- und Platzkomplexität für die drei Algorithmen

Durch Überwachen der Ergebnisse, Reduzierung der Konvergenzzeit bzgl. Mutex oder der Platzanforderung kann ein weiteren Aspekt der Performanz beeinflusst werden: eine schnelle Konvergenz bzgl. Mutex impliziert keine schnelle Einrichtung der gewünschten Rekurrenz. Dieser Zielkonflikt wird hinsichtlich Algorithmen A und B offensichtlich. Der wesentliche Algorithmus ist Algorithmus C, welcher eine verbesserte Version der ersten Beiden, durch Überarbeitung des Inkrementierungssystems von [BPV08] ist. Algorithmus C erreicht beides 1.0 Rekurrenz von *priv* und optimale Konvergenzzeit bzgl. Mutex. Bemerke, dass dieser Algorithmus eventuell nicht bzgl. $priv_{1.0}$ in $2 \cdot \mathsf{diam}(\mathscr{G})$ schritten konvergiert, wie Algorithmus A es tut.

Die Optimalität der Konvergenzzeit bzgl. Mutex unter dem synchronen Prozess-Scheduler, die $\lceil \mathsf{diam}(\mathscr{G})/2 \rceil - 1$ gleicht, ist begründet im Verfeinern des früheren Beweises von [DG13], welcher die Optimalität von $\lceil \mathsf{diam}(\mathscr{G})/2 \rceil$ einfordert.

4 Schnelle und begründete eindeutige Prozessauswahl

In Mutex ist das Ziel von Fairness sicherzustellen, dass jeder Prozess die Chance erhält seine Aktionen unendlich oft auszuführen, ohne für immer blockiert zu werden. Dies fügt üblicherweise Einschränkungen zur Optimierung des Algorithmus bzgl. anderer Eigenschaften hinzu. Zu diesem Zweck, zeigt dieser Abschnitt ein spezielles Problem, dass mit wechselseitigem Ausschluss in Bezug steht. Das Problem betrachtet das Gewähren eines einmaligen Privilegs zu Prozessen um eine spezielle Aktion auszuführen, ähnlich zu Mutex, jedoch mit einer speziellen Beachtung der Fairness. Dieses Problem wird *eindeutige Prozessauswahl* genannt und hat eine einfache und eine erweiterte Definition.

In der einfachen Definition wird Fairness im Design des Algorithmus vernachlässigt, während man eine Anforderung behält, dass wenn ein anfragender Prozess p existiert, dann wird einem anfragenden Prozess q (d.h. nicht notwendigerweise p) schließlich Privileg gewährt. Das ermöglicht immer noch allen anfragenden Prozessen privilegiert zu werden, unter der Bedingung: Der Algorithmus sollte hoch rekurrent sein im Gewähren von Privilegien, verglichen mit der Häufigkeit des Anfragens von Privilegien. Ein praktisches Beispiel für solch eine Erscheinung ist ein Überwachungssystem von Umweltaktionen, wie Erdbeben, wobei es praktikabler ist eine schnelle Auswahl zu ermöglichen, anstelle gleichmäßig alle Prozesse abzudecken.

In der erweiterten Definition, der Auswahl von Prozessen denen Privileg gewährt wird, sollte begründet sein in lokalen und globalen Kriterien, welches als *begründete eindeutige Prozessauswahl* referenziert wird. Die ist für Systeme nützlich die als Prozess-Scheduler auftreten, welche die Ausführung anderer Systeme optimieren.

In dieser Arbeit, folgt der Entwurf von selbststabilisierenden Algorithmen bzgl. eindeutiger Prozessauswahl der Auffassung von *Weiterleitung von Information mit Rückmeldung* (Engl. *Propagation of Information with Feedback*, kurz *PIF*) [Ch82, Se83]. PIF ist eine Art Wellenalgorithmus der Token in einer Art Breitensuche und Rückmeldung erhält. Zwei selbststabilisierende PIF-Algorithmen wurden entworfen um jeweils eine schnell und begründete eindeutiger Prozessauswahl zu erreichen.

Eine Ausführung von beliebigen Algorithmen ist begründet in einer unendlichen Wiederholung von zwei PIF Zyklen, wobei in jedem Zyklus, die Wurzel Token verbreitet und Rückmeldungen erhält. Im ersten Zyklus verbreitet die Wurzel ein *Such*-token, dass sich durch den Baum ausbreitet. Dann wird ein *Feedback*-token zur Wurzel gesendet. Im einfachen Fall wird das Feedback-token direkt gesendet, sobald der anfragende Prozess das Such-token erhält. Im erweiterten Fall werden die Feedback-tokens von allen Blättern zur Wurzel aggregiert, um eine begründete Auswahl zu erreichen. Im zweiten Zyklus wählt die Wurzel einen Prozess und sendet zu diesem ein *Ausführungs*-token. Sobald der ausgewählte Prozess seine Aktion ausführt, sendet er ein *Fertig*-token zur Wurzel.

Merke, dass obwohl dieses Szenario einfach erscheint, ist dies was Selbststabilisierung bzgl. eindeutiger Prozessauswahl anbelangt nicht der Fall. Tabelle 2 fasst die Komplexitätsmaße für eindeutige Prozessauswahl zusammen (kurz *EPA*) und vergleicht diese mit zwei anderen Ansätzen, unter der Annahme, dass ein Prozess anfragt: (1) der virtuelle Ring Ansatz, in welchem ein Token in einer Art Tiefensuche in einem Baum herumgereicht wird, um den anfragenden Prozess zu finden und (2) die synchrone Übereinstimmung, welche auf dem Inkrementierungssystem, präsentiert in Abschnitt 3, basiert ist. Offensichtlich ist der bedeutende Beitrag, dass die Rekurrenz Δ in diesem Fall $\mathcal{O}(\frac{1}{\mathsf{depth}(\mathscr{T})})$ ist, während sie für die anderen Fälle $\mathcal{O}(\frac{1}{\mathsf{n}})$ ist.

	EPA Algorithmus	Virtueller Ring	Synchrone Übereinstimmung
EPA-Konvergenzzeit	$\mathsf{depth}(\mathscr{T})$	$\mathcal{O}(\mathsf{depth}(\mathscr{T}))$	$\lceil \mathsf{diam}(\mathscr{G})/2 \rceil - 1$
Rekurrenz Δ	$1/(4 \cdot \mathsf{depth}(\mathscr{T}))$	$1/(\mathsf{n} + \lvert\mathscr{E}\rvert - 1)$	$1/\mathsf{n}$
$priv_\Delta$-Konvergenzzeit	$7 \cdot \mathsf{depth}(\mathscr{T})$	$\mathcal{O}(\mathsf{depth}(\mathscr{T}))$	$2 \cdot \mathsf{diam}(\mathscr{G}) + \mathsf{n}$
Platz pro Prozess	$4(\mathsf{n}+1)$	3	$\mathsf{n} + \mathsf{diam}(\mathscr{G})$

Tab. 2: Eindeutige Prozessauswahl von einigen Arten von Algorithmen, unter der Annahme, dass es genau einen anfragenden Prozess gibt ($\mathscr{T} \to$ Baum, $\mathscr{G} \to$ Graph, $\mathsf{n} \to$ Baumgröße, und $\mathscr{E} \to$ Kanten)

5 TDMA Zeitschlitzzuteilung

Das Problem der Zeitmultiplexverfahren (Engl. *Time-Division-Multiple-Access*, kurz *TDMA*) Zeitschlitzzuteilung [Ra02] wird in diesem Abschnitt betrachtet, für Systeme die sich an die Europäische Norm EN 54-25 [DI05] halten. Allgemein beschäftigt TDMA sich mit der Zuteilung von Knoten einer Topologie zu Zeitschlitzen, zugunsten der Zeitplanung von Kommunikation zwischen Knoten, wenn ein Kommunikationsmedium nur eine begrenzte Anzahl von Nachrichten zu jedem Zeitpunkt erlaubt. TDMA garantiert, dass die Knoten das Medium ohne Nachrichtenkollision oder Verlust teilen.

In TDMA wird Zeit in periodische Intervalle genannt *Zeitfenster* aufgeteilt, wobei jedes Fenster in *Zeitschlitze* geteilt wird. Jeder Schlitz wird einem Knoten zugeteilt und ein Knoten sendet Nachrichten nur während seines zugeteilten Zeitschlitzes. Möglichkeiten allen Netzwerkknoten die selbe Zeit anzubieten, hängen von der Netzwerkarchitektur ab. In vielen praktischen Fällen, z.B. drahtlose Sensornetzwerke, ist jeder Knoten mit einer Hardware-Uhr ausgestattet. Jedoch Temperatur-, Spannungsänderungen und Rauschen können zu einer Veränderung der Kristallfrequenz führen, weswegen eine Gangabweichung auftritt. Deswegen ist eine Uhrensynchronisation erforderlich.

Allgemein können Uhrensynchronisationsmechanismen die Kommunikation zwischen mehreren Knoten, um Zeitstempel auszutauschen, beinhalten. Jedoch können sie nicht die Abwesenheit von Gangabweichung aus zwei Gründen garantieren: (1) Zwei aufeinanderfolgende Uhrsynchronisationen können weit genug auseinander liegen, dass eine Uhr kritisch hoch dazwischen abweicht. (2) Erhaltene Zeitstempel können unkorrekt sein, wenn diese von einem Knoten kommen, der keine Referenzuhr bereitstellt, sondern ebenso seine Uhrzeit von einem dritten Knoten erhält.

Eine beschränkte Gangabweichung kann geduldet werden, durch die Nutzung der Idee einer Schutzzeit [PS13]. Schutzzeit besteht aus zwei Zeitintervallen die am Anfang und am Ende jedes Zeitschlitzes hinzugefügt werden. Während der Schutzzeit eines Zeitschlitzes, wenn ein Knoten beauftragt ist zu senden, dann sendet er nicht. Jedoch, wenn er mit Zuhören beauftragt ist, tut er dies. Mit diesem Aufbau bringt eine beschränkte Gangabweichung keine Nachrichtenkollision oder Nachrichtenverlust. Jedoch vergrößert die Schutzzeit die Zeitschlitzlänge und verzögert Nachrichten, was wiederum die Performanz reduziert: Es senkt die Rekurrenz des Sendens von Nachrichten mit der Zeit. Das Ziel ist zu optimieren, d.h. minimieren der Schutzzeit.

Der Beitrag betreffend dieses Problems ist wie folgt. Zuerst, wird ein formales Modell von Systemen im Sinne von EN 54-25 [DI05] einschließlich Echtzeitaspekte und unter einigen Annahmen gezeigt. Das Modell formalisiert einen TDMA Zeitplan mit einem Uhrensynchronisationsmechanismus für Baumtopologien. Zweitens, werden basierend auf dem Modell, jeweils die Topologische Beschreibung der Zeitschlitzzuteilung, welche die größte und kleinste optimale Schutzzeit unter allen Zuteilungen benötigt gegeben. Dies spezifiziert welche Zeitschlitzzuteilungen die höchste Rekurrenz vom senden der Nachrichten hat. Als nächstes werden Gleichungen zum Berechnen der optimalen Schutzzeit für jeden der vorherigen Fälle gegeben. Darauf werden scharfe untere und obere Schranken, für die Rekurrenz Δ vom Nachrichten senden, für beliebige Zeitschlitzzuteilung gegeben. Schließlich zeigt eine Fallstudie eines Drahtlosen Feueralarmsystems, wie das gegebene Modell leicht erweitert werden kann, um auf die Alltagssituation zu passen.

6 Automatische Verifikation von Rekurrenzeigenschaften

Dieser Abschnitt untersucht die Durchführbarkeit der Verifikation von Rekurrenzeigenschaften mittels automatischer Verifikationswerkzeugen. Rekurrenzeigenschaften – con_Δ-Konvergenz und con_Δ-Aufwärmen – sind über unendlichen Ausführungen definiert, und die Rekurrenz von con ist nicht notwendigerweise einheitlich während jeder Ausführung. Dies erschwert automatische Verifikation. Das Ziel dieser Arbeit ist zu vereinfachen und automatische Verifikation von Rekurrenzeigenschaften zu ermöglichen, im Grunde durch das Überwinden des Problems des Vorhandenseins unendlicher Ausführungen.

Der Vereinfachungsansatz zu Verifizieren beliebiger con_Δ-Konvergenz oder con_Δ-Aufwärmen ist wie folgt. Zuerst wird gezeigt, dass wenn eine Ausführung Ξ eines Systems (oder eines Algorithmus) eine betrachtete Eigenschaft verletzt, dann existiert eine *endliche* Unterausführung Ξ' von Ξ mit einer *festen Länge*, so dass (1) Ξ' die Eigenschaft verletzt , und (2) Ξ' ist *minimal*; d.h. es gibt keine strikte Unterausführung von Ξ', die die Eigenschaft verletzt. Ξ' wird ein *minimales Gegenbeispiel* genannt. Zweitens, da jede Konfiguration eines selbststabilisierenden Systems als Anfang betrachtet wird, folgt das Ξ' ein Ausführungspräfix einer Ausführung des Systems ist. Schließlich, durch die ersten und zweiten Gegebenheiten, falls eine Ausführung des Systems existiert, welche die Eigenschaft verletzt, dann existiert ein Ausführungspräfix des Systems,

welches ein minimales Gegenbeispiel ist. Dies impliziert, dass das Überprüfen von Ausführungspräfixen endlicher Länge ausreicht, um zu verifizieren, ob ein System die Eigenschaft erfüllt.

Es ist bekannt, dass für jedes System das con_Δ-Konvergenz in c Schritten (bzw. con_Δ-Aufwärmen in w Schritten) erfüllt, gibt es ein minimales Gegenbeispiel dessen Länge $\mathsf{c}+1$ (bzw. in $[w+1, 2w+1]$) ist.

Um dies zu illustrieren, die folgenden Ausführungen sind Gegenbeispiele der Eigenschaft $\mathsf{con}_{0.5}$-Konvergenz von $3 = \mathsf{c}$ Schritten. Die minimalen Gegenbeispiele haben Größe $4 = \mathsf{c}+1$ und sind zwischen den Klammern gekennzeichnet.

$\Xi_1 : (\gamma_0, \gamma_1, \gamma_2, \gamma_3,) \quad \underline{\gamma_4}, \gamma_5, \underline{\gamma_6}, \gamma_7, \underline{\gamma_8}, \gamma_9, \underline{\gamma_{10}}, \gamma_{11}, \underline{\gamma_{12}}, \cdots$
$\Xi_2 : \gamma_0, \gamma_1, \gamma_2, \underline{\gamma_3}, \quad (\gamma_4, \underline{\gamma_5}, \gamma_6, \gamma_7,) \quad \underline{\gamma_8}, \gamma_9, \underline{\gamma_{10}}, \gamma_{11}, \underline{\gamma_{12}}, \cdots$
$\Xi_3 : \underline{\gamma_0}, \quad (\underline{\gamma_1}, \gamma_2, \gamma_3, \gamma_4,) \quad \underline{\gamma_5}, \gamma_6, \underline{\gamma_7}, \gamma_8, \underline{\gamma_9}, \gamma_{10}, \underline{\gamma_{11}}, \cdots$

Die folgende Ausführung ist ein minimales Gegenbeispiel, welches nicht $\mathsf{con}_{0.5}$-Aufwärmzeit von $3 = \mathsf{w}$ Schritte erfüllt. Es hat eine Länge von $7 = 2\mathsf{w}+1$.

$\Xi : \gamma_0, \gamma_1, \underline{\gamma_2}, \underline{\gamma_3}, \underline{\gamma_4}, \gamma_5, \gamma_6.$

Modellprüfung (Engl. *Model Checking*) kann angewandt werden, um die Nichtexistenz eines Gegenbeispieles zu verifizieren und zu schlussfolgern, dass ein System eine Rekurrenzeigenschaft erfüllt. Als eine Fallstudie wird der Modellprüfer NUXMV [Ca14] angewandt, um die Rekurrenzeigenschaften des wechselseitigen Ausschlussalgorithmus zu verifizieren, wobei dies Algorithmus C in Abschnitt 3 ist. Die Ergebnisse der Modellprüfung bestätigen die theoretische Analyse. Das Haupthindernis ist, dass die Zustandsexplosion durch die Anzahl an Prozessen die Prüfzeit enorm anhebt.

7 Fazit

Das verallgemeinerte Konzept der Selbststabilisierung vergrößert die Klasse der Eigenschaften, die die klassische Selbststabilisierung abdeckt. Es modelliert Performanzeigenschaften, die über die Ausführungen für beide, Diskrete und Echtzeitsysteme, definiert sind. Es zeigt wie selbststabilisierende Algorithmen entworfen oder umgearbeitet werden können, um Effizient bzgl. ihrer Servicequalität zu werden.

Das neue Konzept der Selbststabilisiserung kann noch weiter erweitert werden, um mehr Eigenschaften abzudecken. Erstens kann das Konzept weiter verallgemeinert werden, um abzudecken, dass eine Bedingung notwendig zumindest einmal halten muss, in einem Teilabschnitten von Ausführungen oder Evolutionen, anstelle von Konfigurationen. Dies stellt eine Möglichkeit bereit, eine Gleichförmigkeit in der Häufigkeit der Erfüllung einer Bedingung, während einer Ausführung, zu erzwingen. Zweitens kann die Rekurrenz Δ neu definiert werden, um *Gewicht* oder *Grad* des Erfüllens einer Bedingung pro Konfiguration zu beinhalten. Das Gewicht kann z.B. das Verhältnis aus lokalen Zuständen in einem verteilten System widerspiegeln, die eine lokale Bedingung erfüllen.

Literaturverzeichnis

[BPV08] Boulinier, Christian; Petit, Franck; Villain, Vincent: Synchronous vs. Asynchronous Unison. Algorithmica, 51(1), 2008.

[Ca14] Cavada, Roberto; Cimatti, Alessandro; Dorigatti, Michele; Griggio, Alberto; Mariotti, Alessandro; Micheli, Andrea; Mover, Sergio; Roveri, Marco; Tonetta, Stefano: The nuXmv Symbolic Model Checker. In: Proceedings of the 26th International Conference on Computer Aided Verification (CAV). Jgg. 8559 in LNCS. Springer, S. 334–342, 2014.

[Ch82] Chang, Ernest J. H.: Echo Algorithms: Depth Parallel Operations on General Graphs. IEEE Transactions on Software Engineering (TSE), 8(4):391–401, 1982.

[DG13] Dubois, Swan; Guerraoui, Rachid: Introducing Speculation in Self-Stabilization – An Application to Mutual Exclusion. In: Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC). ACM, S. 290–298, 2013.

[Di74] Dijkstra, Edsger W.: Self-stabilizing Systems in Spite of Distributed Control. Communications of the ACM, 17(11):643–644, 1974.

[Di86] Dijkstra, Edsger W.: A Belated Proof of Self-Stabilization. Distributed Computing, 1(1):5–6, 1986.

[DI05] DIN e.V.: , Fire Detection and Fire Alarm Systems – Part 25: Components Using Radio Links and System Requirements, German Version EN 54-25:2005, 2005.

[PS13] Perahia, Eldad; Stacey, Robert: Next Generation Wireless LANS: 802.11 n and 802.11 ac. Cambridge university press, 2013. ISBN: 978-1107016767.

[Ra02] Rappaport, Theodore S.: Wireless Communications: Principles and Practice, Jgg. 2. Prentice Hall, 2002. ISBN: 978-0130422323.

[Se83] Segall, Adrian: Distributed Network Protocols. IEEE Transactions on Information Theory, 29(1):23–34, 1983.

Oday Jubran wurde am 5. Mai 1986 in Jerusalem geboren. Er erhielt den Bachelor in Wirtschaftsinformatik – mit dem Nebenfach Betriebswirtschaftslehre – von der Universität Bethlehem, Palästina im Jahr 2008. Danach hat ihm der DAAD ein Master-Stipendium in Deutschland zugesprochen. Er erhielt den Master in Informatik von der Albert-Ludwigs-Universität Freiburg im Jahr 2012. Später beschäftigte er sich als Wissenschaftlichermitarbeiter an der Carl von Ossietzky Universität Oldenburg, und er hat dort seine Doktorarbeit in Informatik im Jahr 2016 abgeschlossen. Jetzt beschäftigt er sich mit der Softwareentwicklung und Forschung in der Industrie.