Bewertung des Kommunikations-verhaltens eingebetteter Systeme mit Hilfe selbstlernender Verfahren

Dissertation

Zur Erlangung des akademischen Grades des

Doktors der Ingenieurswissenschaften

der Fakultät für Angewandte Informatik
der Universität Augsburg

eingereicht von
Dipl.-Ing. Falk Langer
im Juli 2014

Erstgutachter: Prof. Dr. Rudi Knorr
Zweitgutachter: Prof. Dr. Alexander Knapp

Tag der mündlichen Prüfung: 05.02.2015

Kurzfassung

Moderne Automobile bestehen heute aus einer Vielzahl interagierender mechanischer und elektronischer Komponenten. Mit zunehmender Vernetzung der einzelnen Komponenten und der immer größer werdenden Zahl von softwarebasierten Funktionen im Fahrzeug sind Automobile heute schon ein Paradebeispiel für vernetzte eingebettete Systeme. Die großen Vorteile softwarebasierter Lösungen sind deren Flexibilität zum Entwicklungszeitpunkt und die Möglichkeit, sehr viele Informationen aus der Umwelt zu nutzen, um eine optimale Funktion des Fahrzeuges zu erzielen. Erst dadurch konnten in den vergangenen Jahren viele neue Techniken zur Erhöhung der Energieeffizienz und zur Verbesserung der Sicherheit im Fahrzeug entwickelt werden. Doch dieser Grad der Vernetzung und Adaptivität im Fahrzeug stellt auch neue Anforderungen an die Entwicklung und den Test dieser Systeme. Ein großes Problem bei der Absicherung der komplexen Interaktion dieser hochgradig vernetzten Systeme sind unvollständige oder fehlerhafte Spezifikationen des erforderlichen Systemverhaltens. Um dies zu verbessern, konzentriert sich ein Großteil der Forschungsaktivitäten darauf, Methoden und Verfahren zu entwerfen, die es ermöglichen, das gewünschte Verhalten dieser eingebetteten verteilten Systeme besser und einfacher beschreibbar und somit auch besser testbar zu machen.

Die vorliegende Arbeit geht an dieser Stelle einen anderen, in der Fachwelt bislang neuen Weg, indem sie die Möglichkeit untersucht, für den Testprozess das Kommunikationsverhalten eines verteilten Systems anhand spezieller Netzwerktraces besser bewertbar zu machen. Dies soll helfen Entwicklungs- oder auch Softwarefehler im Rahmen von Fahrzeugtests schneller zu identifizieren. Zu diesem Zweck wird in dieser Arbeit ein Verfahren entwickelt, welches Abweichungen des Kommunikationsverhaltens des zu testenden Systems von vorher aufgenommenen Netzwerk- bzw. Referenztraces feststellen kann. Dieses neue Verfahren erlaubt es, die Bewertung des Kommunikationsverhaltens und die Analyse des Netzwerktraces ohne das Vorliegen einer Spezifikation des Verhaltens des Systems durchzuführen. Das Verfahren gliedert sich in eine Lernphase und eine Anwendungsphase. In der Lernphase werden mittels speziell angepasster Clustering- und Lernalgorithmen Überwachungsmodelle aus dem Netzwerktrace extrahiert. Diese Überwachungsmodelle werden für die Bewertung und Analyse weiterer Netzwerkkommunikation des zu testenden Systems in der Anwendungsphase eingesetzt. Um die Komplexität des zu erlernenden Verhaltens für die Lernalgorithmen zu reduzieren, werden in der Lernphase einzelne lösbare Teilprobleme extrahiert. Dies wird durch eine neue Methodik zur Identifizierung von parallelen Handlungssträngen in Netzwerktraces erreicht. Diese Methodik ermöglicht es, Lernverfahren ohne Vorwissen über das System zur Extrahierung von Verhaltensmodellen aus dem Netzwerktrace anzuwenden. Auf Basis einer eigens eingeführten Bewertungsmethodik werden verschiedene Algorithmen und Verfahrensschritte systematisch verglichen und verbessert. Anhand einer abschließenden Evaluierung mit realen Netzwerkdaten eines Fahrzeuges werden die Machbarkeit und Anwendbarkeit des neu eingeführten Verfahrens unter Beweis gestellt. Die Ergebnisse zeigen, dass es möglich ist, Überwachungsmodelle zur Bewertung der Netzwerkkommunikation aus vorher bekannten Netzwerk- bzw. Referenztraces zu erstellen. Somit können Änderungen im Kommunikationsverhalten erkannt und potenzielle Fehler frühzeitig entdeckt werden. Im Vergleich zu anderen Testverfahren sind vor allem die mit 45 % ermittelte hohe Abdeckung des bewerteten Netzwerkverkehrs und eine mit heuristischen Analyseverfahren vergleichbare Fehlalarmrate von ca. 35 % hervorzuheben.

Vorwort

Die vorliegende Arbeit entstand begleitend zu meiner Tätigkeit am Fraunhofer-Institut für Eingebettete Systeme und Kommunikationstechnik ESK.

Ich möchte mich an dieser Stelle bei meinem Doktorvater Prof. Dr.-Ing. Rudi Knorr für die Betreuung der Arbeit und bei Herrn Prof. Dr. Alexander Knapp für die Übernahme des Zweitgutachtens bedanken.

Ein besonderer Dank gilt meiner Frau Franziska und meiner Familie, die mich im Laufe der Jahre fortwährend unterstützt und motiviert haben.

Inhaltsverzeichnis

1 Einleitung

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."[1] – Zitat Leslie Lamport (Lamport 1987)

Im Bereich der eingebetteten Systeme ist seit einigen Jahren ein stark zunehmender Grad der Vernetzung zu beobachten. Dieser Trend spiegelt sich auch in den aktuellen Forschungsschwerpunkten zu Cyber-Physical-Systems (CPS) (Geisberger, Broy 2012), zum Internet of Things (Uckelmann, Harrison, Michahelles 2011) und zu Mixed-Criticality-Systemen (Thompson 2012) wider.

Auch bei der Elektronikarchitektur moderner Kraftfahrzeuge zeigt sich diese Entwicklung. So wird derzeit nicht mehr nur über die Anzahl der Steuergeräte im Fahrzeug diskutiert, sondern auch darüber, mit welcher Bandbreite und welchen Netzwerktechnologien die verbleibenden Geräte zu verbinden sind. Mehr als 80 Recheneinheiten (ECUs – Electronic Control Units) mit ca. 2 000 ausgeführten Diensten und Übertragungsraten von bis zu 100 Mbit/s sind schon heute in aktuellen Fahrzeugen zu finden (Manderscheid, Langer 2011). Dies als ein verteiltes System nach (Tanenbaum, van Steen 2006) zu bezeichnen, scheint also angemessen. Es ist spätestens seit (Lamport 1987) bekannt, dass verteilte Systeme schwieriger zu entwickeln sind als zentral gesteuerte Systeme. Demgegenüber stehen die sicherheitskritischen Funktionen des Fahrzeuges, welche einen Teil dieses verteilten Systems darstellen und entsprechend abgesichert werden müssen.

Die hier vorliegende Arbeit ist im Anwendungsbereich des Testens von vernetzten eingebetteten Systemen angesiedelt, wobei der spezielle Fokus auf dem Testen verteilter eingebetteter Systeme liegt. Dort soll das Verhalten geschlossener Kommunikationssysteme im Fahrzeug besser bewertbar gemacht werden. Bei diesen Systemen treten im Gesamtsystemtest immer noch unerwartet viele Fehler auf, obwohl sie auf Funktions- und Modulebene bereits sehr ausgiebig getestet worden sind. Ein Grund für dieses Problem ist das Vorliegen von unvollständiger oder fehlerhafter Spezifikation (Ramberger, Gruber, Herzner 2004). Daher wird in dieser Arbeit ein Verfahren entwickelt, welches das Kommunikationsverhalten im Fahrzeug auf Basis bekannten Verhaltens und ohne den Rückgriff auf eine Spezifikation bewerten kann.

Der Anwendungsschwerpunkt des in dieser Arbeit entwickelten Verfahrens liegt in der Auswertung von Netzwerktraces[2], welche aus Fahrzeugtests stammen. Das Ziel ist es, die Fehlersuche bei Fahrzeugtests zu unterstützen, indem die aufgezeichneten Netzwerktraces effizienter ausgewertet werden. Die Auswertung beruht auf dem Vergleich des Kommunikationsverhaltens mit einem zu einem früheren Zeitpunkt aufgezeichneten Referenztrace[3].

[1] zu deutsch: Ein verteiltes System erkennt man daran, dass ein defekter Computer, dessen Existenz dir nicht bekannt ist, die Benutzung deines eigenen Computers einschränkt.
[2] Ein Netzwerktrace ist die Aufzeichnung des Datenverkehrs in einem Netzwerk.
[3] Ein Referenztrace ist ein Netzwerktrace, welches als Referenz für da Verhalten eines Systems benutzt wird.

Im folgenden Kapitel 1.1 wird eine tiefer gehende Erläuterung zum Kontext und der Motivation der vorliegenden Arbeit gegeben. Dort wird ein Überblick gegeben, wo die Defizite respektive konzeptionellen Grenzen etablierter Entwicklungs- und Testmethoden liegen und welche Ursachen dies hat.

In Kapitel 1.2 wird anschließend die Zielstellung der Arbeit detailliert erörtert. Darin wird ein kurzer Überblick zu den wissenschaftlichen Schwerpunkten der Arbeit aufgezeigt, da diese nicht zwangsläufig auf dem Gebiet des Testens angesiedelt sind.

Abschließend wird in Kapitel 1.3 zur besseren Orientierung die Gesamtstruktur und damit die generelle Vorgehensweise der Arbeit vorgestellt.

1.1 Motivation

Für die Entwicklung von sicherheitskritischen Funktionen hat sich das sogenannte V-Modell (d. h. in Phasen organisiertes Vorgehensmodell für die Softwareentwicklung nach (Friedrich 2008)) als gängige Methode der Software- und Systementwicklung etabliert. Grundlegende Voraussetzungen für das Vorgehen nach dem V-Modell sind das Vorhandensein einer vollständigen Anforderungsliste und die Erstellung einer vollständigen Spezifikation. Es zeigt sich aber, dass die Vollständigkeit der Spezifikation vor allem im Bereich der Beschreibung des Verhaltens von verteilter Funktionalität nicht immer gegeben ist (Lutz 1993; Ramberger, Gruber, Herzner 2004; Ebert, Jones 2009). Folglich stoßen die im V-Modell angewandten Methoden und Verfahren bei der Überprüfung des Kommunikationsverhaltens der verteilten Funktionen im Fahrzeug an ihre methodenbedingten Grenzen.

Diese Grenzen zeigen sich vor allem bei System- und Fahrzeugtests. Dort ist es oft sehr schwierig, die Interaktion der einzelnen Systeme genau zu bewerten, da die Spezifikation an diesem Punkt häufig nur Auskunft über das zu erwartende übergeordnete Systemverhalten gibt. So wird eine Spezifikation beispielsweise vorgeben, dass im Fall der Betätigung eines bestimmten Bedienelements das System eine dedizierte Reaktion in Form einer für Nutzer sichtbaren Aktion zeigen muss. Der Rückschluss, was genau aufgrund der Bedienaktion innerhalb des Systems stattfinden muss, ist bei verteilten Funktionen aufgrund von stark verzweigten Wirkketten zumeist schwierig. In der Folge ist es bei System- bzw. Fahrzeugtests im Regelfall nur möglich, einzig den Teil des nicht erwünschten Kommunikationsverhaltens zu entdecken, welcher letztendlich in den Testszenarien eine Auswirkung auf die vom Nutzer sichtbaren Aktionen hat (Lutz 2003).

Daraus schlussfolgernd muss ein Fehler im Verhalten des Systems eine für den Nutzer nicht erwünschte, sichtbare Aktion verursachen, um vom Tester entdeckt zu werden. Es wird in diesem Zusammenhang auch von Defekten gesprochen. Ein Defekt ist, auf diesen Fall bezogen, eine vom Nutzer wahrnehmbare Fehlfunktion. Ein Fehler hingegen ist eine Fehlfunktion des Systems, welche aber nicht unbedingt eine Auswirkung für den Nutzer haben muss und somit nicht zwingend zu einem Defekt führt (Avizienis u. a. 2000). Verursacht der Fehler im Verhalten des Systems keinen Defekt, so bleibt er unentdeckt. Es ist davon auszugehen, dass auch in einem fertigen Produkt noch eine signifikante Anzahl an Fehlern im Kommunikationsverhalten enthalten ist, diese aber in den Zuständen, die das System im Betrieb einnehmen kann, nicht zu einem Defekt führen (Avizienis u. a. 2000). Das Risiko, dass ein unentdeckter Fehler in nicht

getesteten Systemzuständen einen Defekt verursacht, ist entsprechend größer als in bereits getesteten Systemzuständen. Das Risiko eines Defektes vergrößert sich demnach durch die abnehmende Testabdeckung des Kommunikationsverhaltens bei Systemtests im Vergleich zu Modultests (Langer, Prehofer 2011). Aus diesem Grund werden bei Fahrzeugtests (d. h. Tests, in denen das Fahrzeug als komplettes System getestet wird) vergleichsweise viele Fehler gefunden, obwohl jedes einzelne Modul für sich allein fehlerfrei getestet wurde.

Die mangelnde Testabdeckung resultiert vornehmlich aus fehlender oder unvollständiger Spezifikation des Verhaltens des verteilten Systems. Da das Verhalten eines verteilten Systems auf Anwendungsebene schwierig zu beschreiben ist, wird sehr intensiv an neuen Methoden zur Modellierung des Systemverhaltens geforscht. Diese stützen sich meist auf spezielle Beschreibungssprachen oder Modelle, um die Zusammenhänge des Systems besser erfassen zu können (z. B. Weiss u. a. 2010; Becker, Zeller, Weiss 2012; Drabek u. a. 2013).

Es kann aber auch mit diesen Methoden nicht verhindert werden, dass die Spezifikation durch eine abweichende Implementierung im Laufe des Entwicklungsprozesses nicht mehr der vorliegenden Software entspricht. Aus diesem Grund gibt es einige Arbeiten, welche sich im Bereich der Reverse-Engineering-Techniken bedienen, um bestehende Spezifikationen aus bereits entwickelten Systemen zu aktualisieren (z. B. Grinchtein, Jonsson, Leucker 2004; Bollig u. a. 2007). Diese Verfahrensweise hilft dem Entwickler, die Spezifikation aktuell zu halten und eventuelle Spezifikationslücken zu schließen.

All diesen Methoden gemein ist der verhältnismäßig hohe Aufwand, der geleistet werden muss, um eine entsprechende schlüssige und lückenlose Spezifikation und daraus resultierende Testfälle zu erstellen. Daher gibt es Ansätze, die Spezifikation direkt mit der Testausführung zu koppeln. So werden beispielsweise ausführbare Spezifikationsmodelle erstellt, welche auch als Referenzmodelle zum Test des Kommunikationsverhaltens benutzt werden können (Eilers, Weiss 2010). Ein ähnlicher Ansatz wird auch von (Adzic 2011) unter dem Begriff „Specificaton by Example“ verfolgt, wenngleich dort keine ausführbaren Modelle beschrieben sind. Grundlegend scheint es aber sinnvoll und in manchen Fällen auch einfacher, eine Spezifikation in Form eines reellen Beispiels zu liefern.

Diese Arbeit widmet sich dem Problem unvollständiger Spezifikation und daraus resultierender „blinder Flecken“ im Test verteilter Systeme. Zu diesem Zweck werden Referenztraces als Beispiel für erwünschtes Kommunikationsverhalten analysiert und somit als alternative Informationsquelle verwendet. Das Referenztrace ist dabei als Beispiel des Normalverhaltens zu interpretieren, gegen welches weitere Traces verglichen werden. Bewegen sich die zu prüfenden Traces nicht in dieser Norm, so wird dies als Abweichung von dem bisherig beobachteten Normverhalten interpretiert. Im Folgenden wird daher auch von einer Out-of-Norm(OoN)-Erkennung gesprochen und das entwickelte Verfahren als selbstlernendes OoN-Erkennungsverfahren bezeichnet.

1.2 Zielstellung

Das übergeordnete Ziel der Arbeit ist es, das Kommunikationsverhalten im Rahmen von System- bzw. Fahrzeugtests besser bewertbar zu machen. Da es in diesen Bereichen häufig keine ausreichend genaue Spezifikation gibt, soll hierzu aufgezeichnetes

Kommunikationsverhalten aus früheren Testdurchläufen benutzt werden. Vor diesem Hintergrund soll in dieser Arbeit ein Verfahren entwickelt werden, welches die Bewertung des Kommunikationsverhaltens eventbasierter, reaktiver verteilter eingebetteter Systeme im Vergleich zu einer Referenzaufzeichnung ermöglicht.

Dazu muss eine entsprechende Repräsentation bzw. Abbildung des Kommunikationsverhaltens gefunden werden, so dass eine allgemeingültige Beschreibung desselben möglich ist. Der Schwerpunkt soll hierbei vor allem auf die Abbildung von Abhängigkeiten in Form von Sequenzen bzw. Protokollautomaten gelegt werden, da diese eine sehr gute qualitative Beurteilung des Kommunikationsverhaltens erlauben. Im Rahmen des zu erstellenden Verfahrens sollen Algorithmen vorgestellt und untersucht werden, welche es erlauben, entsprechende Informationen aus bekanntem Kommunikationsverhalten zu extrahieren, so dass diese Informationen dazu genutzt werden können, neues Kommunikationsverhalten in einer geeigneten Art und Weise klassifizieren zu können.

Der Forschungsgegenstand dieser Arbeit ist somit die Untersuchung der Machbarkeit eines Verfahrens, welches eine automatisierte qualitative Bewertung des Kommunikationsverhaltens verteilter Systeme erlaubt. Das Verfahren soll, ausgehend von vorliegenden Referenztraces, den Inhalt weiterer Netzwerktraces prüfen und einen Hinweis geben können, ob diese ein anderes Verhalten des Systems enthalten, als dies im Referenztrace der Fall ist. Somit wird weitgehend ohne die Zuhilfenahme einer Spezifikation ein Verfahren ermöglicht, welches auf mögliche Fehler im Kommunikationsverhalten und somit auf mögliche Bugs des verteilten Systems hinweisen kann.

1.2.1 Wissenschaftliche Schwerpunkte

Der wissenschaftliche Beitrag der Arbeit liegt in der Entwicklung und der Machbarkeitsanalyse für ein Verfahren zur qualitativen Bewertung des Kommunikationsverhaltens auf Basis von Referenztraces. Die wesentlichen Schwerpunkte hierbei sind:

(i) die Definition der Evaluierungsmethodik und der Bewertungskriterien,

(ii) die Adaption, die Erweiterung und der Vergleich geeigneter Algorithmen zum Erstellen von Verhaltensmodellen aus Referenztraces,

(iii) die Entwicklung eines Verfahrens zur Identifizierung von parallelen Handlungssträngen im Kommunikationsverhalten von verteilten eingebetteten Systemen und

(iv) die Erkennung von überspezialisierten Verhaltensmodellen zur Reduzierung der Fehlalarmrate.

Um die Anwendbarkeit des Verfahrens zu zeigen, wird auf der Basis realer Fahrzeugdaten eine abschließende Evaluierung durchgeführt.

Dieser systematische Ansatz und dessen Validierung sind in dieser Form bislang weder in der wissenschaftlichen noch wirtschaftlichen Fachwelt veröffentlicht respektive bekannt.

1.2.2 Einordnung

In der Arbeit werden keine neuen Testmethoden erstellt oder untersucht, da ein Test ohne Spezifikation oder Anforderung quasi nicht möglich ist. Vielmehr ist es das Ziel, eine zusätzliche Möglichkeit der qualitativen Bewertung des Systemverhaltens im Laufe des Entwicklungszyklus zu etablieren. Zu diesem Zweck werden zur Bewertung des Kommunikationsverhaltens Abhängigkeitsmodelle für die im Referenztrace enthaltenen Events erlernt.

Die grundlegend angewandte Methodik der Erkennung bzw. Beschreibung von nicht exakt formulierbarem Verhalten ist dabei ähnlich einiger selbstlernenden Fehlererkennungsmechanismen. Beispiele hierfür sind Arbeiten im Bereich Anomaly Detection (Hangal, Lam 2002; Maxion, Tan 2002), Immune Inspired Fault Detection (Lemos u. a. 2007) oder auch der Einsatz künstlicher neuronaler Netze zur Fehlererkennung (Atkinson, Long, Hanzevack 1998).

Der grundlegende Unterschied zu dem in dieser Arbeit entwickelten selbstlernenden OoN-Erkennungsverfahren liegt darin, dass bei den oben aufgezählten Verfahren nach Fehlersymptomen gesucht wird. Es werden ausschließlich die Negativ- bzw. Fehlerfälle und nicht die Positiv- bzw. Gutfälle bewertet. Zu diesem Zweck müssen Fehler im Vorfeld bekannt sein, welche die Fehlererkennung gezielt als negatives Verhalten identifizieren soll. Somit werden ausgewählte negative Verhaltensmuster eines Systems erlernt und später wieder erkannt.

Demgegenüber muss für die zu lösende Problemstellung der vorliegenden Arbeit das vollständige (positive) Systemverhalten erlernt werden. Ein potenzieller Fehler wird durch das hier entwickelte selbstlernende OoN-Erkennungsverfahren dann identifiziert, wenn das im Netzwerk beobachtbare Verhalten des Systems durch die erlernten Modelle *nicht* abgebildet ist.

Einen ähnlichen Weg der Beschreibung des positiven Systemverhaltens gehen einige Arbeiten der Anomaly Detection aus dem Bereich der Einbruchserkennung in IT-Systemen (Chandola, Banerjee, Kumar 2009). In der Regel werden hierzu quantitative Eigenschaften wie Häufigkeit oder Frequenz einer Kommunikationsbeziehung betrachtet und beschrieben (Lazarevic, Kumar, Srivastava 2005). Eine qualitative Analyse in der Form der Beschreibung des zu erwartenden Kontrollflusses des Programms findet aufgrund der mangelnden Reproduzierbarkeit von IT-Systemen nicht oder nur sehr begrenzt statt.

1.3 Struktur der Arbeit

Der grundlegende Aufbau und die Anwendung des selbstlernenden OoN-Erkennungsverfahrens werden in Kapitel 2 vorgestellt. Einführend werden hier zunächst das Anwendungsumfeld und die Anwendungsschritte des selbstlernenden OoN-Erkennungsverfahrens näher erläutert. Anschließend werden grundlegende Eigenschaften des zu überprüfenden Systems bzw. der zu prüfenden Daten analysiert. Zu diesem Zweck wird eine Systemvermutung definiert, welche eine möglichst einfache Hypothese zur Beschreibung der Zusammenhänge der Ereignisse im Trace aufstellt. Darauf aufbauend werden die grundlegende Problemstellung des selbstlernenden OoN-Erkennungsverfahrens dargelegt und die letztendlichen Bewertungskriterien für die OoN-Erkennung definiert.

Im Kapitel 3 wird die Anwendbarkeit verschiedener Lernverfahren bzw. Algorithmen auf die gegebene Problemstellung untersucht und evaluiert. Im Fokus stehen dabei Verfahren aus dem Bereich der künstlichen neuronalen Netze (KNN), Lernverfahren für Markov-Modelle und ein Verfahren zum Lernen von diskreten endlichen Automaten (DEA). Ausgehend von den dort gewonnenen Ergebnissen wird das Verfahren zum Lernen von DEA mittels des Angluin L^*-Algorithmus tiefer gehend untersucht. Zu diesem Zweck wird eine eigens für die Problemstellung optimierte Implementierung erstellt und mittels einer dafür entworfenen Bewertungsmethodik in Bezug auf die Problemstellung evaluiert. In einer abschließenden Evaluierung des Verfahrens auf Basis des Angluin L^*-Algorithmus, wird mittels realer Fahrzeugdaten gezeigt, dass die Komplexität des Netzwerktraces die Leistungsfähigkeit des eingesetzten Lernverfahrens übersteigt. Dieses Problem resultiert aus der Existenz paralleler Handlungsstränge im Netzwerktrace, mit deren Auflösung sich das nachfolgende Kapitel beschäftigt.

Kapitel 4 der Arbeit widmet sich dem Problem der Identifizierung paralleler Handlungsstränge im Kommunikationsverhalten. Parallelität ist ein typisches Merkmal für verteilte Systeme, erschwert aber die Extrahierung von Abhängigkeitsmodellen ungemein. Daher wird zunächst anhand der aufgestellten Systemvermutung aus Kapitel 2 gezeigt, dass es möglich ist, die Komplexität des Lernproblems durch die Auflösung der Parallelität in einzelne unabhängige Handlungsstränge signifikant zu reduzieren. Um dies zu erreichen, wird in diesem Kapitel ein Verfahren zur Separation der beobachteten Events in sinnvolle Gruppen vorgestellt und evaluiert. Zu diesem Zweck werden mehrere Clustering-Verfahren untersucht und angewendet. Die Herausforderung bei der Anwendung der Clustering-Verfahren sind die mangelnden Differenzierungsmerkmale der beobachteten Events. Zur Gewinnung von Eigenschaften für eine sinnvolle Gruppierung werden eine Methode zur Bewertung des zeitlichen Verhaltens mittels Spektralanalyse eingeführt und ein spezielles entropiebasiertes Clustering-Verfahren vorgestellt. Die Bewertung der Clustering-Ergebnisse wird mittels einer eigenen, auf dem Angluin L^*-Algorithmus aufbauenden Methodik auf Basis reeller CAN-Traces eines Fahrzeuges vorgenommen.

Im Kapitel 5 werden die Ergebnisse aus Kapitel 3 f. genutzt und zusammengefasst, um daraus das im Kapitel 2 beschriebene selbstlernende OoN-Erkennungsverfahren anzuwenden und zu bewerten. Zunächst werden der Aufbau und die Wirkungsweise des selbstlernenden OoN-Erkennungsverfahrens in der Anwendungsphase erläutert. Anschließend wird eine Optimierung des selbstlernenden OoN-Erkennungsverfahrens zur Verringerung der Fehlalarmraten mittels des Vergleichs der Ergebnisse aus mehreren geprüften Traces vorgestellt und evaluiert.

In Kapitel 6 wird eine zusammenfassende Bewertung des selbstlernenden OoN-Erkennungsverfahrens durchgeführt. Dazu werden die Ergebnisse aus den einzelnen Verfahrensschritten zusammengefasst, um eine Aussage über die letztendliche Performance der OoN-Erkennung im Hinblick auf Fehlalarmrate und die Erkennung von tatsächlichen Abweichungen geben zu können. Die daraus resultierenden Erkenntnisse ermöglichen eine Einschätzung über die Anwendbarkeit und den zu erwartenden Mehrwert des vorgestellten selbstlernenden OoN-Erkennungsverfahrens.

Kapitel 7 fast die Ergebnisse der Arbeit in einer Schlussbetrachtung zusammen und gibt einen Ausblick über mögliche Ansatzpunkte für zukünftige Untersuchungen.

2 Selbstlernende OoN-Erkennung in verteilten eingebetteten Systemen

Der Begriff Out-of-Norm-Erkennung für Kommunikation in eingebetteten Netzwerken wird in (Peti, Obermaisser, Kopetz 2005) eingeführt. Die dort vorgestellte Problemstellung und Methodik dienen zur Fehlererkennung im Kommunikationsverhalten eingebetteter Systeme[4] mit dem Fokus auf fahrzeuginterne Kommunikation. Darauf aufbauend wird in (Obermaisser, Peti 2007) ein Verfahren vorgeschlagen, welches zudem das zeitliche Verhalten bewerten kann. Diese Arbeiten zeigen erstmals, dass es bei einem verteilten eingebetteten System nicht ausreicht, das Verhalten der Einzelkomponenten zu beschreiben. Das dort beschriebene Meta-Level-Verhalten, welches sich hauptsächlich durch die Kombination verschiedener Zustände der Schnittstellen äußert, bedarf einer gesonderten Beschreibung. Diese Beschreibung ähnelt sehr stark der Spezifikation von Protokollautomaten. In (Peti, Obermaisser, Kopetz 2005) und (Obermaisser, Peti 2007) wird von einer vollständigen Spezifikation ausgegangen und der Einsatz des dort vorgestellten Verfahrens im fertigen Produkt in Form eines Fehlererkennungsmechanismus beschrieben.

Das in der vorliegenden Arbeit entwickelte selbstlernende OoN-Erkennungsverfahren zeigt einen neuen Ansatz, das Verhalten von verteilten eingebetteten Systemen anhand seiner Netzwerkkommunikation bewertbar zu machen, ohne auf eine vollständige Spezifikation angewiesen zu sein. Das selbstlernende OoN-Erkennungsverfahren soll dabei helfen, ein Systemverhalten, wie es von (Peti, Obermaisser, Kopetz 2005) und (Obermaisser, Peti 2007) beschrieben wird, besser bewertbar zu machen. Das Verfahren nach (Peti, Obermaisser, Kopetz 2005) und (Obermaisser, Peti 2007) wird ausschließlich zur Fehlererkennung im fertigen Produkt verwendet, im Gegensatz dazu soll das in der vorliegenden Arbeit entwickelte selbstlernende OoN-Erkennungsverfahren bei entwicklungsbegleitenden Tests eingesetzt werden.

Die Grundlage der Bewertung innerhalb des entwickelten selbstlernenden OoN-Erkennungsverfahrens bildet dabei die Analyse bereits vorliegender Kommunikationsdaten. Im Rahmen dieser Analyse werden Abhängigkeitsmodelle erlernt, welche zur weiteren Bewertung des System- bzw. Kommunikationsverhaltens verwendet werden. Somit bilden die zum Erlernen genutzten Referenztraces in gewisser Weise den Ersatz für eine nicht vorliegende Spezifikation.

In diesem Kapitel werden die grundlegenden Eigenschaften des selbstlernenden OoN-Erkennungsverfahrens zur Bewertung des Kommunikationsverhaltens vorgestellt und erläutert.

Als Einstieg wird in Kapitel 2.1 zunächst die Einbindung des selbstlernenden OoN-Erkennungsverfahrens in bestehende Testabläufe aufgezeigt. Hier wird dargelegt, wie die Referenztraces, welche die Grundlage für die selbstlernende OoN-Erkennung bilden, zustande kommen und wie sie eingesetzt und verarbeitet werden. Um die Anwendung

[4] Der Begriff „System“ wird in dieser Arbeit stets für das zu untersuchende bzw. ausgeführte verteilte eingebettete System oder einzelner Bestandteile davon verwendet. Technologien, welche dazu dienen, dieses System zu erstellen, zu testen oder zu analysieren, werden als „Verfahren“, „Methodik“ oder „Algorithmus“ bezeichnet.

des selbstlernenden OoN-Erkennungsverfahrens zu verstehen, wird in den folgenden Abschnitten dargelegt, zu welchem Zeitpunkt und in welchem Verfahrensschritt die Referenztraces verarbeitet werden. Weiterhin wird aufgezeigt, wie die erlernten Abhängigkeitsmodelle für die Bewertung von Netzwerkkommunikation genutzt werden.

Das Kapitel 2.2 beschreibt die grundlegenden Eigenschaften, welche für das ausgeführte System, das es zu erlernen gilt, vermutet werden. Diese Systemvermutung ist die Basis sowohl für die Anwendung und Evaluierung der Kapitel 3 vorgestellten Lernverfahren als auch der im Kapitel 4 eingeführten Clustering-Verfahren. Weiterhin wird basierend auf der Systemvermutung eine genaue Nomenklatur der Eigenschaften für das Referenztrace hergeleitet. Abschließend wird gezeigt, inwieweit das Referenztrace den tatsächlichen Ersatz für fehlende Spezifikation darstellen kann.

Abschließend wird in Kapitel 2.3 die Wirkungsweise des selbstlernenden OoN-Erkennungsverfahrens vorgestellt. Hierzu wird zunächst der Aufbau des entwickelten OoN-Erkennungsverfahrens erläutert. Anschließend werden die daraus resultierenden Problemstellungen für das Clustering und das Lernen von Abhängigkeitsmodellen dargelegt. Nachdem die zu lösenden Problemstellungen klar definiert sind, werden im letzten Abschnitt die Bewertungskriterien hergeleitet und definiert, anhand derer das vorgestellte OoN-Erkennungsverfahren beurteilt werden soll.

2.1 Anwendung des selbstlernenden OoN-Erkennungsverfahrens

Nachfolgend wird gezeigt, wie das selbstlernende OoN-Erkennungsverfahren im Anwendungskontext einsetzbar sein soll. Hierzu wird beschrieben, welche Testschritte im Entwicklungszyklus eines verteilten Systems durchlaufen werden und an welcher Stelle das selbstlernende OoN-Erkennungsverfahren eingesetzt werden soll.

Da es sich dabei um ein selbstlernendes Verfahren handelt, sind für die Anwendung zur OoN-Erkennung zwei Phasen notwendig. In der ersten Phase – der Lernphase - werden aus Referenztraces Abhängigkeitsmodelle erstellt. Diese Abhängigkeitsmodelle werden in der zweiten Phase – der Anwendungsphase - genutzt, um weitere Netzwerktraces zu prüfen.

2.1.1 Einordnung des OoN-Erkennungsverfahrens in den Entwicklungszyklus

Im Folgenden werden die in Frage kommenden Testmethoden, bei denen das selbstlernende OoN-Erkennungsverfahren eingesetzt werden kann, kurz erläutert. Das Augenmerk liegt hierbei auf dem Ursprung der Referenztraces als Input für die Lernphase und dem Kontext, in dem das selbstlernende OoN-Erkennungsverfahren in der Anwendungsphase ausgeführt wird.

Während des Entwicklungszyklus eingebetteter verteilter Systeme werden unterschiedliche Testmethoden angewandt. Im Kontext dieser Arbeit sind besonders die Modul-, Integrations- und Systemtests sowie der Fahrzeugtest als Erweiterung des Systemtests zu betrachten (Abbildung 2-1).

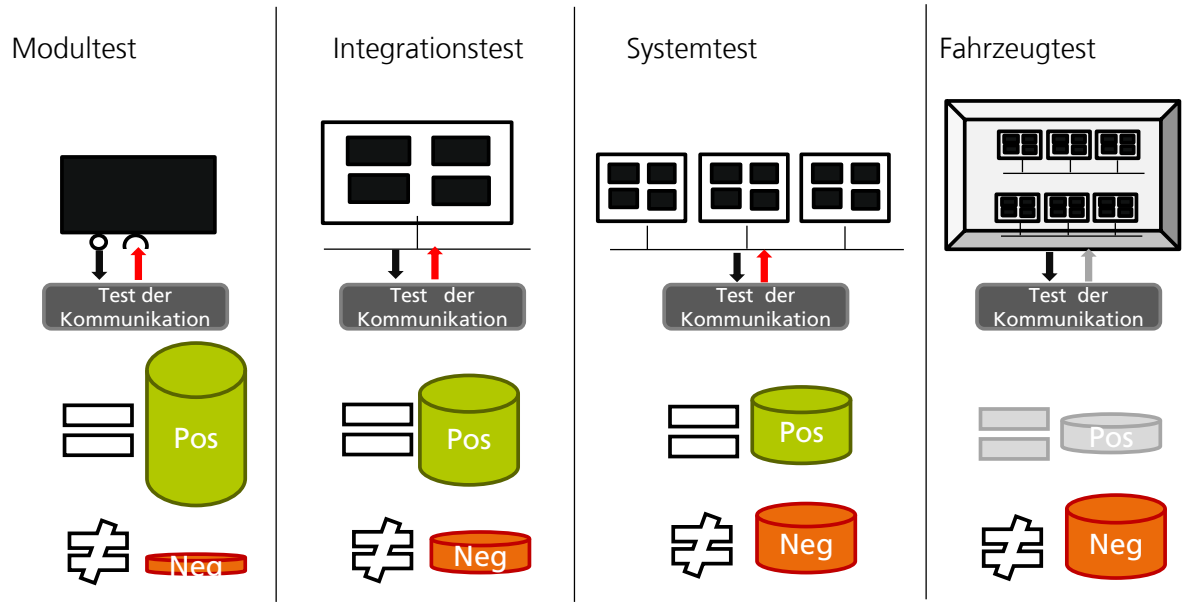

Abbildung 2-1 Darstellung der Testmethoden im Kontext des Entwicklungszyklus mit entsprechender Verfügbarkeit von Positiv- und Negativtestfällen zur Beurteilung des Netzwerk- bzw. Kommunikationsverhaltens

Modultest:

Die ersten Tests einer neuen Funktion sind i. d. R. Modul- oder auch Unit-Tests. Dies sind White-Box-Tests, bei denen einzelne Komponenten der Software getestet werden. In der Regel werden einzelne Schnittstellen mit einem Eingabewert beaufschlagt und nach einem Berechnungsschritt der Ausgabewert mit dem Sollwert verglichen. Wie in Abbildung 2-1 links dargestellt, ist die Anzahl der Testfälle, bei denen das Sollverhalten des Moduls explizit getestet wird (Positivtests), tendenziell am größten. Bei der Testdurchführung wird ein (Software-)Modul mit einem Testinput beaufschlagt und die beobachtete Reaktion des Moduls (Testoutput) mit der nach der Spezifikation erwarteten Reaktion verglichen. Derartige Testfälle werden im Folgenden Positivtests genannt, da sie überprüfen, ob eine bestimmte Anforderung aus der Spezifikation vom System erfüllt wird.

Integrationstest :

Hier werden einzelne, voneinander abhängige Komponenten bezüglich ihrer korrekten Interaktion miteinander bzw. Integration in ein Steuergerät getestet (Abbildung 2-1, zweite Spalte von links). Der Integrationstest stellt damit eine erste Validierung des Schnittstellendesigns dar. Da der Integrationstest häufig nicht im Kontext des Gesamtsystems durchgeführt wird, muss der Rest des Systems in einer geeigneten Art und Weise simuliert werden. Im Automobilbereich wird dabei oft von Restbussimulation gesprochen. In diesem Fall gibt es verschiedene Arten der Testüberwachung. Die gängigste Methode ist es, das Teilsystem gezielt mit Eingabedaten zu stimulieren und die Systemreaktion mit dem gewünschten Verhalten zu vergleichen. Diese Methodik ähnelt dem Modultest, unterscheidet sich jedoch insofern davon, dass gewöhnlich das zu testende System nicht transparent ist (Black-Box-Test) und dass es sich um eine für sich ausführbare Softwareeinheit oder ein Steuergerät handelt.

Eine weitere Möglichkeit, das beobachtete Verhalten zu prüfen, liegt darin, gezielt nach nicht erwünschten Verhaltensmustern zu suchen (im Folgenden Negativtest genannt). Hier wird z. B. nach gewissen explizit ausgewiesenen Fehlersignaturen der Funktionen gesucht oder auch nach Verhaltensmustern, welche bereits bekannte Implementierungsfehler verursacht haben, was häufig bei Regressionstests angewendet wird.

Systemtest :

Ein Systemtest soll das Systemverhalten eines Systemverbundes testen. Die nachfolgend genannten Teilsystem-, Abnahme- und Fahrzeugtests stellen Systemtests dar.

Teilsystemtest :

Bei der Fahrzeugentwicklung findet ein Teilsystemtest üblicherweise an sog. Teilsystemprüfplätzen oder in ähnlich prototypischer Umgebung statt. Die Übergänge vom Integrationstest zum Teilsystemtest bis zum nachfolgend beschriebenen Abnahmetest sind oft fließend. Da bei einem Fahrzeug einzelne Teilsysteme häufig relativ unabhängig voneinander ausgeführt werden, kann ein Teilsystemtest auch als Systemtests eingestuft werden. Ein solches Teilsystem ist oftmals noch mit einer Netzwerkschnittstelle zu anderen Teilsystemen verbunden (Abbildung 2-1, zweite Spalte von rechts).

Systemabnahmetest

Im Hinblick auf die Validierung eines im Fahrzeug verteilten Softwaresystems liegt der Unterschied zwischen Teilsystemtest und Abnahmetest darin, dass im Teilsystemtest zusätzliche Debugging- oder Tracing-Hardware verbaut ist, was beim Systemabnahmetest nicht der Fall ist. Diese zusätzliche Hardware erlaubt es, neben dem normalen Netzwerkverkehr auch sonst nicht sichtbares Verhalten analysierbar zu machen. Eine Instrumentierung wie im Integrationstest oder auch Modultest ist aber meist weder im Teilsystemtest noch im Systemabnahmetest vorgesehen.

Da beim Teilsystemtest und Systemabnahmetest die Stimulation des Systems meist ausschließlich über die letztendlich im Produkt vorhandenen Schnittstellen (Sensoren, Bedienelemente etc.) erfolgt, wird auch die Bewertung des Systemverhaltens meist auf dieser Ebene vorgenommen. Das Netzwerkverhalten wird bei diesen Systemabnahmetests zwar häufig aufgezeichnet, eine Auswertung erfolgt aber meist nur im Sinne von Negativtests. Das bedeutet, dass in den Netzwerktraces entweder nach vom System selbst generierten Fehlercodes oder nach bekannten Fehlermustern aus früheren Testfällen gesucht wird.

Fahrzeugtest:

Um die letztendliche Gewissheit über die richtige Funktion eines Systems im Fahrzeug zu erlangen, wird für jedes Teilsystem ein Systemabnahmetest im fertigen Fahrzeug unter realen Umweltbedingungen durchgeführt. Diese Art von Tests, wie sie in Abbildung 2-1 in der rechten Spalte dargestellt ist, soll im Folgenden als Fahrzeugtest bezeichnet werden.

Die bei den Fahrzeugtests anfallenden Netzwerktraces werden im Normalfall einer Standardprüfung unterzogen, bei der nach negativen Testsequenzen gesucht wird (Abbildung 2-2). In einem ersten Schritt wird nach einer durchgeführten Testfahrt an Traces, bei denen der Testfahrer eine Fehlfunktion des Fahrzeuges festgestellt hat, direkt eine manuellen Fehlersuche durchgeführt (Abbildung 2-2, Punkt 1). Hat der Testfahrer keine Fehlfunktionen feststellen können, wird das aufgezeichnete Netzwerktrace nach bekannten Fehlersymptomen durchsucht (Abbildung 2-2, Punkt 2). Wird ein Fehlersymptom gefunden, erfolgt eine manuelle Auswertung des Traces (Abbildung 2-2, Punkt 3). Wird kein Fehlersymptom gefunden, wird das Verhalten als akzeptabel angesehen und die Traces werden verworfen (Abbildung 2-2, Punkt 4).

Das Vorgehen für Positivtests ist auf Netzwerkebene bei Fahrzeugtest nur mit großem technischem Aufwand realisierbar und wird daher nur selten angewendet. Dies liegt hauptsächlich daran, dass das Kommunikationssystem im Fahrzeug nach außen als geschlossenes System betrachtet werden kann. Die Stimulation des Systems wird nicht über Netzwerkeingaben, sondern über externe Sensorik vorgenommen. Daher ist es auch nur selten möglich, bei einem Systemtest oder Fahrzeugtest gezielt Sequenzen in das Netzwerk einzuspeisen und eine entsprechende Antwortsequenz im Netzwerk zu beobachten.

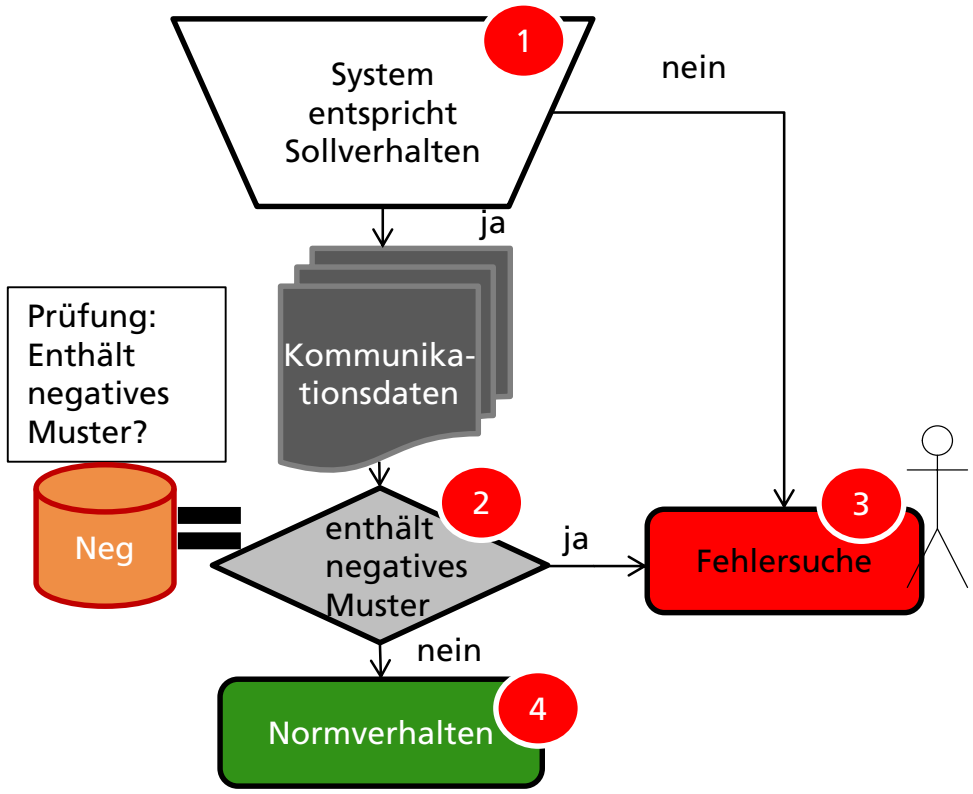

Abbildung 2-2 Standardvorgehen für Überprüfung der Netzwerktraces bei Fahrzeugtests

Verwertbare Daten für das in dieser Arbeit vorgestellte selbstlernende OoN-Erkennungsverfahren fallen hauptsächlich bei Teilsystem-, Systemabnahme- und Fahrzeugtests an, da dort das System in dem Kontext verwendet wird, in dem es später

auch eingesetzt wird. Weiterhin ist die Verwendung von Netzwerktraces aus Integrationstests möglich. Allerdings muss bei diesen sehr wahrscheinlich eine spezielle Auswahl der Netzwerkdaten vorgenommen werden, da bestimmte Stimuli von außen nicht als Normverhalten interpretiert werden können. Testdaten aus Modultests sind nur schwer verwertbar, da diese meist keine direkte Netzwerkkommunikation enthalten.

In dieser Arbeit soll zunächst ausschließlich das Kommunikationsverhalten aus Fahrzeugtests bewertet werden, da hier die wenigsten Positivtests vorhanden sind. Die entsprechenden Netzwerktraces aus Fahrzeugtests stellen somit die Grundlage für das selbstlernende OoN-Erkennungsverfahren.

2.1.2 Lernphase

Der Input für die Lernphase sind Netzwerktraces, welche bei positiv bewerteten Testdurchläufen in System- oder Fahrzeugtests entstanden sind (Abbildung 2-2). Diese Traces bilden die Referenztraces für das selbstlernende OoN-Erkennungsverfahren und werden diesem zum Lernen von Abhängigkeitsmodellen zugeführt. In der Lernphase des OoN-Erkennungsverfahrens müssen nun die Referenztraces aufbereitet und Abhängigkeitsmodelle erzeugt werden, um später zu prüfende Netzwerktraces bewerten zu können. Wie in Abbildung 2-3 zu sehen ist, wird das selbstlernende OoN-Erkennungsverfahren sehr einfach an bereits bestehende Testauswertungsverfahren angekoppelt.

In bestehenden Auswertungsverfahren werden aufgezeichnete Netzwerktraces aus Testläufen häufig schon automatisiert ausgewertet. Im Wesentlichen besteht das bekannte Auswertungsverfahren aus einer Datenbank, in der die zu prüfenden Muster hinterlegt sind. Dazu wird in den allermeisten Fällen eine einfache Mustersuche im zu prüfenden Netzwerktrace durchgeführt. Wird ein entsprechendes Muster gefunden, so wird eine manuelle Fehlersuche im Trace veranlasst. Wird kein negatives Verhaltensmuster gefunden, werden die geprüften Traces als fehlerfrei angenommen (Abbildung 2-2).

Bisher werden diese fehlerfreien positiven Testdaten anschließend verworfen. Das selbstlernende OoN-Erkennungsverfahren setzt genau an dieser Stelle an, indem es diese positiven Testdaten als Normverhalten annimmt (Abbildung 2-3, Punkt 1) und diese Traces als Referenztrace (Abbildung 2-3, Punkt 2) weiter verarbeitet. Dazu werden auf Basis der gesammelten Referenztraces entsprechende Abhängigkeitsmodelle generiert (Abbildung 2-3, Punkt 3). Diese Modelle enthalten somit ein Abbildbild des beobachteten Netzwerkverhaltens positiv bewerteter Testfälle.

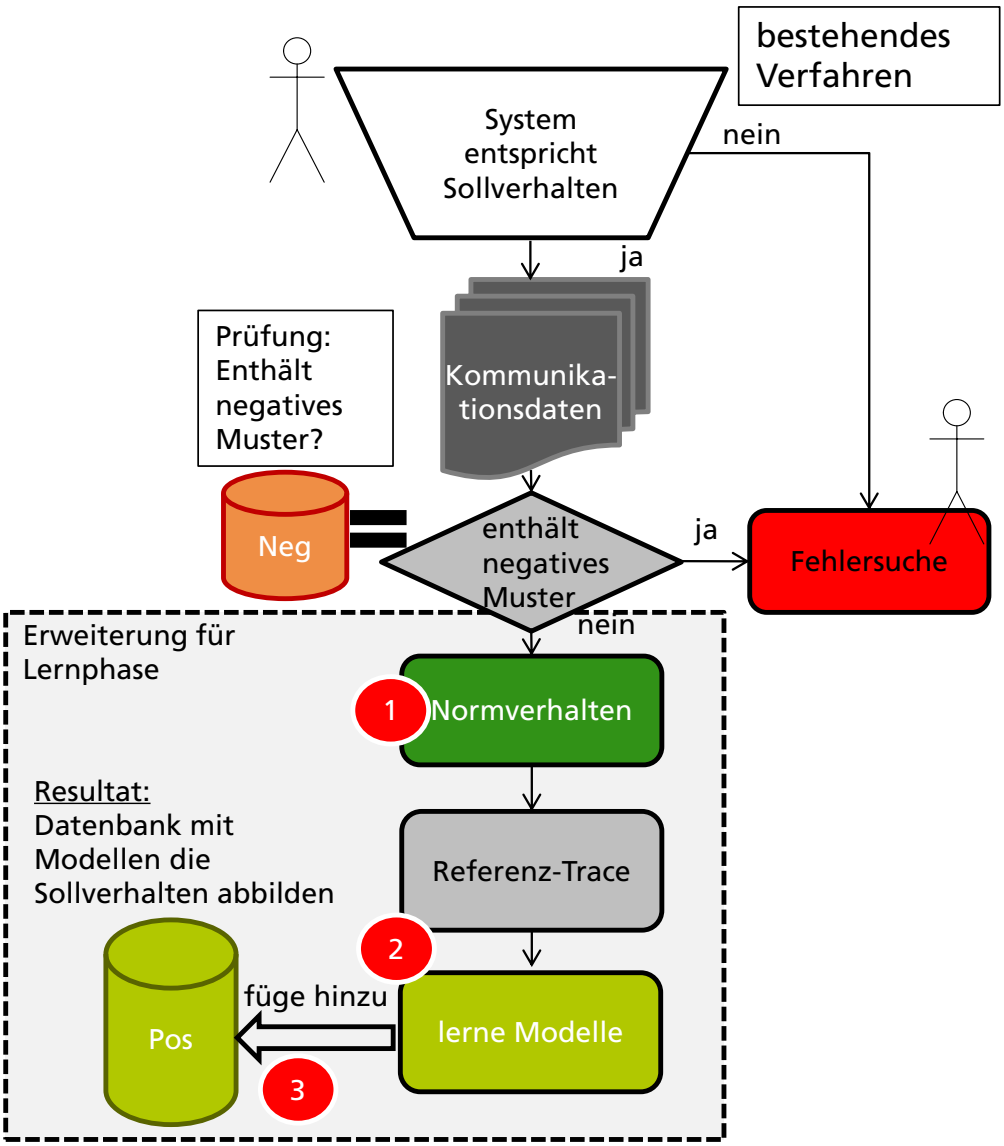

Abbildung 2-3 Lernphase des OoN-Erkennungsverfahrens und Anbindung an bestehende Verfahren zur Bewertung von Netzwerktraces bei Fahrzeugtests

2.1.3 Anwendungsphase

Wurden genügend Referenztraces analysiert und entsprechende Bewertungsmodelle extrahiert, können diese Bewertungsmodelle in der Anwendungsphase des selbstlernenden OoN-Erkennungsverfahrens genutzt werden, um Änderungen im Kommunikationsverhalten des Systems zu erkennen. Hierzu wird das bestehende Verfahren zur Prüfung der Netzwerktraces durch die während der Lernphase extrahierten Verhaltensbeschreibungen der OoN-Erkennung ergänzt, um auch das positive Verhalten bewerten zu können (Abbildung 2-4).

Im Wesentlichen sollen in der Anwendungsphase des selbstlernenden OoN-Erkennungsverfahrens Eventsequenzen gefunden werden, welche nicht durch die bisher aufgezeichneten Referenztraces (Punkt 1 in Abbildung 2-4) beschrieben werden. Findet das OoN-Erkennungsverfahren eine derartige Eventsequenz (Punkt 2 in Abbildung 2-4), kann dies zwei Ursachen haben. Zum Ersten kann es sich tatsächlich um eine nicht

erwünschte Sequenz handeln, welche einen Fehler im System repräsentiert (Punkt 4 in Abbildung 2-4). Zum Zweiten kann es sich auch um neues Verhalten handeln, welches bisher nicht beobachtet wurde (Punkt 3 in Abbildung 2-4). Die Bewertung, ob ein Fehler oder neues Verhalten vorliegen, muss manuell vom Tester oder einem Systemexperten vorgenommen werden.

Sollte es sich um neues Verhalten handeln, welches keinen Fehler darstellt, so kann dies in die Bewertungsmodelle des selbstlernenden OoN-Erkennungsverfahrens aufgenommen und so zukünftig als bekanntes Verhalten eingestuft werden. Sollte ein Fehlerverhalten vorliegen, ist eine Aufnahme in die Datenbank mit negativen Verhaltensmustern möglich.

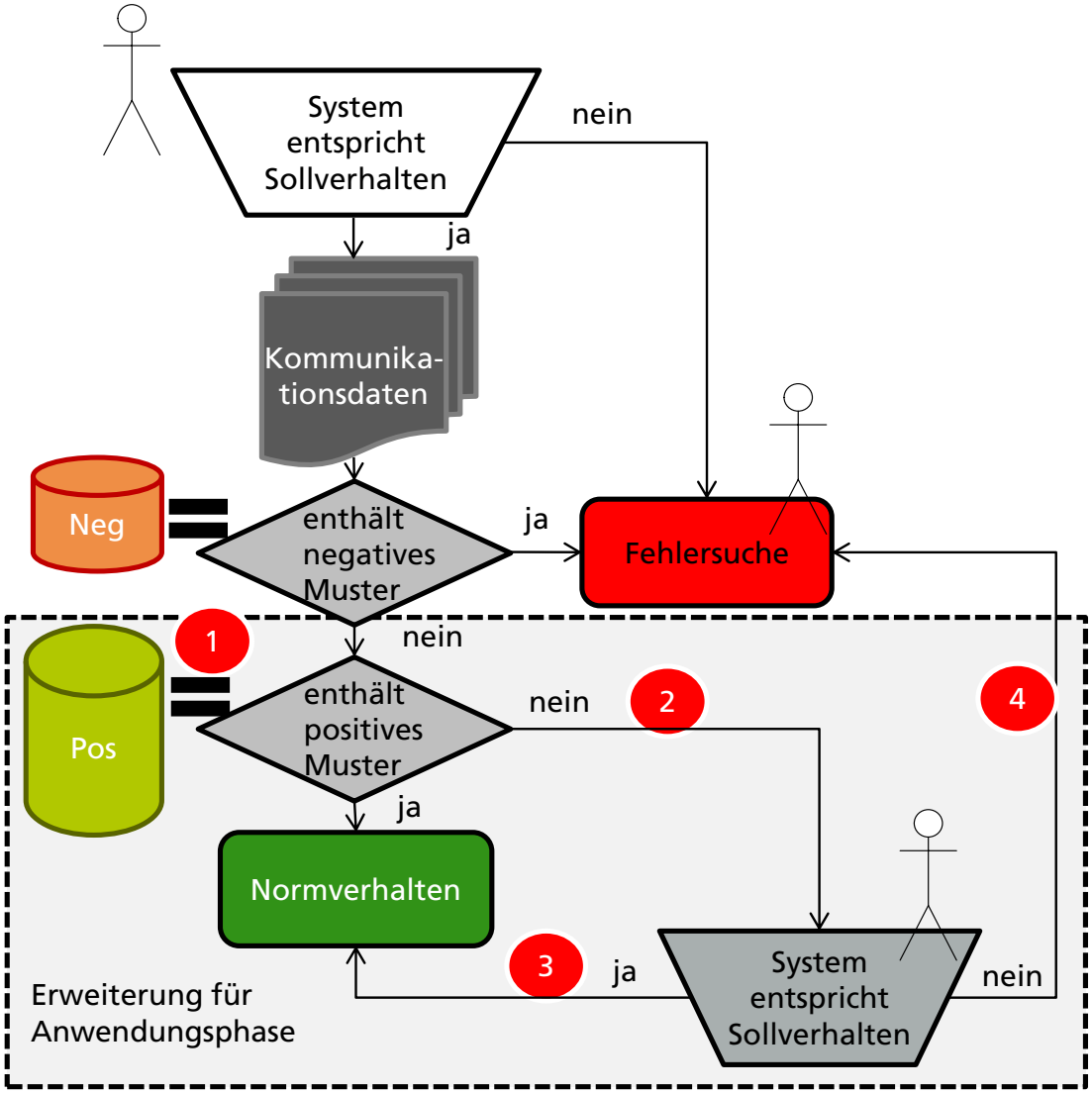

Abbildung 2-4 Anwendungsphase des selbstlernenden OoN-Erkennungverfahrens als Erweiterung des bestehenden Bewertungsverfahrens für Netzwerktraces

2.2 Systemtheorie

Die zu prüfenden Netzwerktraces sind ein Produkt des ausgeführten Systems bzw. der Interaktion mehrerer verteilter Einzelsysteme. Auch wenn die hier vorgestellte OoN-

Erkennung nicht auf die Spezifikation dieser Systeme zurückgreifen kann, so ist es doch wichtig, gewisse Eigenschaften des ausgeführten Systems zu kennen.

Im Folgenden werden daher allgemeingültige Eigenschaften der zu untersuchenden Systeme hergeleitet und beschrieben. Diese Eigenschaften sollen dabei helfen, bestimmte resultierende Eigenschaften des Netzwerktraces besser zu verstehen und Rückschlüsse zur Analyse des Netzwerktraces ziehen zu können.

2.2.1 Formale Beschreibung des ausgeführten Systems

Der Fokus des in dieser Arbeit zu untersuchenden Kommunikationsverhaltens liegt auf eventbasierten Steuerungs- bzw. Kontrollsystemen, welche meist durch logikbasierte oder automatenbasierte Ausdrücke beschrieben werden können. Nicht untersucht werden soll regelungstechnisches Verhalten, welches beispielsweise mittels Differenzialgleichungen oder anderer arithmetischer Methoden abgebildet wird.

Zur Abbildung bzw. zur Spezifikation des Kontrollverhaltens eingebetteter Systeme werden meist Automaten, Sequenzdiagramme oder auch temporale Logik verwendet (Bollig u. a. 2010; Bauer, Leucker, Schallhart 2009, Leucker, Schallhart 2010). Eine geschlossene Beschreibung des gesamten Systemverhaltens mittels Sequenzdiagrammen oder Beispieltraces ist mitunter schwierig. Daher wird in Spezifikationen häufig für komplexes Verhalten eine Darstellung mittels Automaten gewählt. So ist beispielsweise das Unified-Modeling-Language-(UML)-Zustandsdiagramm eine weit verbreitete Methode, Verhalten in verteilten Systemen abzubilden (Weiss u. a. 2010). Auch bei der Ausleitung von Testfällen und Validierung für eventbasierte Systeme wird davon ausgegangen, dass das zugrunde liegende System mittels Automaten abbildbar ist.

Im Zusammenhang mit der Beschreibung des Kommunikationsverhaltens wird ebenfalls häufig von Protokollautomaten gesprochen. Formale Methoden, welche zur Fehlererkennung und Diagnose dieser Systeme verwendet werden, benutzen u. a. auf Automaten basierende Verfahren (Lin, Wonham 1988). Die Implementierung der zugrunde liegenden Software erfolgt nicht zwangsläufig auf Basis eines Automaten. Es kann aber davon ausgegangen werden, dass das Interaktions- bzw. Kommunikationsverhalten von einzelnen Teilfunktionen mit Hilfe von Automaten in großen Teilen beschrieben werden kann. Ebenso kann die Interaktion einzelner Teilsysteme mittels eines Protokollautomaten abgebildet werden. Speziell in einem verteilten System wird die nach außen sichtbare Funktion durch ebendiese Interaktion mehrerer Teilsysteme realisiert. Werden durch das verteilte System mehrerer Funktionen realisiert, kann das Kommunikationsverhalten durch mehrere potenziell unabhängige Automaten beschrieben werden (Abbildung 2-5).

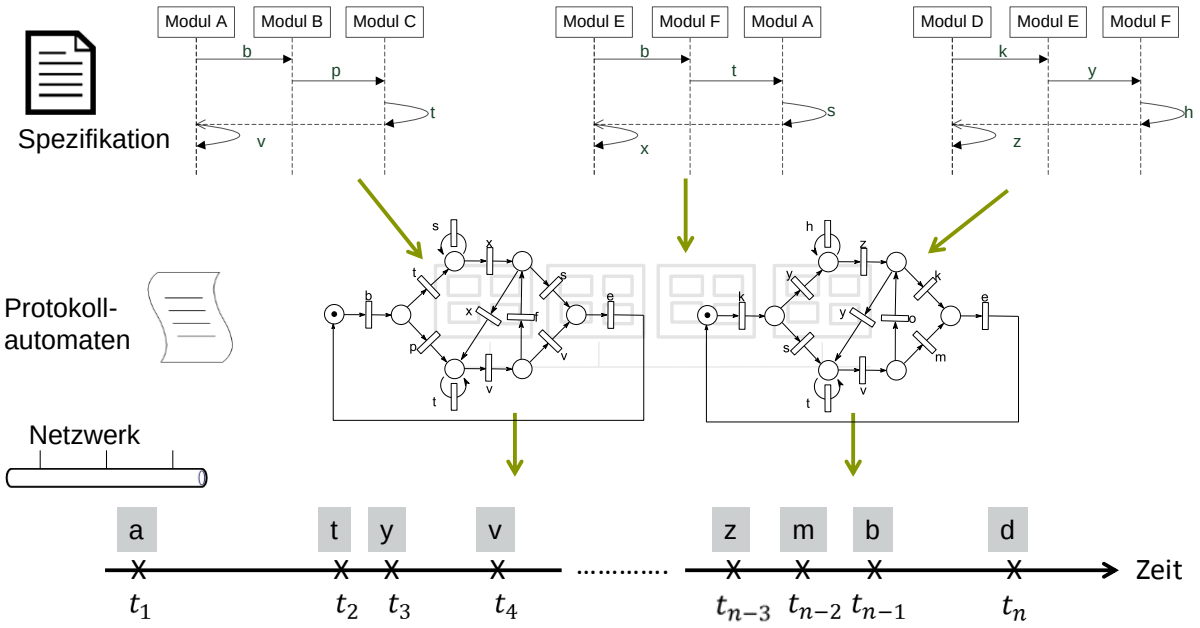

Abbildung 2-5 Beobachtbares Kommunikationsverhalten mehrerer parallel ausgeführter Automaten

Eine Beschreibung der Systemzustände der Automaten kann dabei wie folgt erfolgen. Die Menge aller Systemzustände eines verteilten Systems Z ergibt sich aus der Menge aller unabhängig voneinander agierenden (disjunkten) Funktionen des verteilten Systems, welche jeweils die Zustände Z_i einnehmen können (Definition 2-1). Es ist dabei jede Teilmenge Z_i eine Untermenge von Z ($Z_i \subset Z$), wobei die Teilmengen Z_i disjunkt sind.

Definition:
$$Z = \dot{\bigcup_{i \in I}} Z_i \qquad 2\text{-}1$$

Bezogen auf die Abbildung 2-5 existieren zwei Teilmengen an Zuständen Z_1und Z_2, jeweils für den linken und rechten Protokollautotomaten. Diese Teilmengen beschreiben je einen eigenen Automaten mit den Zuständen $z_1, z_2 \ldots z_n \; mit \; z_i \in Z_i$.

Eine Definition, wie diese Systemzustände zueinander in Beziehung stehen, ist im Bereich des Software- bzw. des Protokolltests zu finden. Dort wird häufig davon ausgegangen, dass das Verhalten eines Systems an dessen Schnittstellen durch einen Zustandsautomaten (Protokollautomat) abbildbar ist (Bochmann, Petrenko 1994; Broy u. a. 2005). Die Beschreibung erfolgt dort als diskreter endlicher Automat (DEA) in Form eines Mealy-Automaten (Mealy 1955).

In der Definition eines solchen Automaten werden zwei verschiedene Typen unterschieden: (i) Akzeptor Automat (Definition 2-2) und (ii) Transduktor Automat (Definiton 2-3) (Hopcroft, Ullman 1992; Broy u. a. 2005).

Definition eines Akzeptorautomaten 2-2

Der Akzeptorautomat ist definiert durch das 5-Tupel $A := (\Sigma, Z, z_0, \sigma, F)$, wobei

- Σ das Eingabealphabet, bestehend aus Zeichen $s \in \Sigma$, ist,
- Z die Menge an Zuständen, die A annehmen kann, ist,
- z_0 der Startzustand von A ; $z_0 \in Z$ ist,
- σ die Übergangsfunktion zwischen den Zuständen: $\sigma : \Sigma \times Z \rightarrow Z$ ist,
- F die Menge der finalen Zustände $F \subseteq Z$ ist.

Definition eines Transduktorautomaten: 2-3

Bei einem Transduktorautomat definiert sich durch das 7-Tupel $M := (\Sigma, \Gamma, Z, z_0, \sigma, \omega, F)$ wobei : $\Sigma, Z, z_0, \sigma, F$ äquivalent zu A sind und

- Γ das Ausgabealphabet, bestehend aus Buchstaben $a \in \Gamma$, ist,
- ω die Ausgangsfunktion: $\omega : \Sigma \times Z \rightarrow \Gamma$, ist.

Für einen Automaten A ergibt sich dadurch für einen bestimmten Zustand z_i ausgehend vom Startzustand z_0 eine Menge an Inputsequenzen S mit $\varphi_j = s_1, s_2 \dots s_n \quad s_i \in \Sigma$, welche den Zustand z_i eindeutig identifizieren (Definition 2-4 f.).

Definition Sequenz $$\boldsymbol{\varphi_j = s_1, s_2 \dots s_n}$$ 2-4

$$\boldsymbol{\varphi_j \rightarrow z_i}$$ 2-5

Welche Art von Automat genutzt wird, hängt davon ab, ob das Eingabealphabet vom Ausgabealphabet getrennt beobachtet werden kann oder diese disjunkt sind ($\Sigma \cap \Gamma = \emptyset$).

2.2.2 Formale Beschreibung eines Traces

Es kann davon ausgegangen werden, dass ein bestimmter Zustand z_i eines Systems nicht von jedem anderem Systemzustand erreicht werden kann. In (Broy, Olderog 2001) wird für die Bestimmung bzw. Definition eines Systemzustands aus der Sicht einer Spezifikation folgende Sequenz angegeben (Definition 2-6),

Definition eines Systemzustandes $$\boldsymbol{\overline{z}_0 \overset{\overline{a}_0}{\rightarrow} \overline{z}_1 \overset{\overline{a}_1}{\rightarrow} \overline{z}_2 \dots}$$ 2-6

wobei $\overline{z}_i$ einen Systemzustand darstellt und $\overline{a}_i$ eine ausgeführte Aktion, um in den nächsten Systemzustand zu kommen. Da in dem vorliegenden Fall das zugrunde liegende System nicht bekannt ist (weder $\overline{z}_i$ noch $\overline{a}_i$ sind bekannt), können nur die sichtbaren Aktionen des Systems bewertet werden. Nach (Mühl, Fiege, Pietzuch 2006) wird ableitend von Definition 2-6 ein sichtbarer Zustand z_i eingeführt, der sich jeweils aus einem Paar $(\overline{z}_i, \overline{a}_i)$ ableitet (Definition 2-7). Die Aneinanderreihung der Zustände z_i ergibt ein Trace t, welches die Ausführung eines Systems beschreibt (Definition 2-8).

Definition sichtbarer Zustand

$$(\overline{z}_i, \overline{a}_i) \rightarrow z_i \qquad 2\text{-}7$$

Definition Trace nach (Mühl, Fiege, Pietzuch 2006)

$$t = z_0, z_1, z_2 \ldots z_n \qquad 2\text{-}8$$

Diese Definition ist für das vorliegende Problem nur bedingt anwendbar, da das Verhalten des Systems nicht im Sinne einer Spezifikation vorliegt. Ein Rückschluss auf den so definierten Zustand und eine ausgeführte Aktion ist nicht ohne Weiteres möglich.

Definition Spezifikation (nach Mühl, Fiege, Pietzuch 2006)

Es kann die Spezifikation Ω eines Systems als Set von Traces t (Definition2-8) definiert werden. Ein System entspricht der Spezifikation, wenn es nur Traces t_i generiert, welche in der Spezifikation Ω enthalten sind. 2-9

Nach (Mühl, Fiege, Pietzuch 2006) ist das Verhalten eines Systems definiert und beschreibbar durch Systemzustände (Definition 2-6) und mehrere Traces (nach Definition 2-8). Innerhalb der Problemstellung für die selbstlernende OoN-Erkennung bezieht sich das bewertbare Systemverhalten ausschließlich auf die sichtbare Kommunikation zwischen einzelnen Komponenten. Es ergibt sich somit eine Menge an (Netzwerk-)Zuständen $\mathfrak{Z}$, welche auf Basis der Kommunikation im System zu beobachten sind. Diese Menge $\mathfrak{Z}$ ist eine Teilmenge aller Systemzustände Z (nach Definition 2-1) die ein System einnehmen kann ($\mathfrak{Z} \subseteq Z$).

Wird die Übergangsfunktion σ des Automaten A konkretisiert zu:

$$\Sigma \times \mathfrak{Z} \mapsto \mathfrak{Z} \qquad 2\text{-}10$$

mit

$$z_0 \xrightarrow{s_0} z_1 \xrightarrow{s_1} z_2 \ldots \xrightarrow{s_{n-1}} z_n \,, \qquad 2\text{-}11$$

so bildet das Paar (z_i, s_i) nach Definition 2-4 einen von außen sichtbaren Zustand ab. Dieser sichtbare Zustand wird hier als Netzwerkzustand z , $z \in \mathfrak{Z}$ bezeichnet.

Im Folgenden wird das Paar (z_i, s_i) Event E genannt (Definition 2-12).

Definition Event

$$(z_i, s_i) \rightarrow E \qquad 2\text{-}12$$

Somit wird durch das Auftreten eines Event e_i im Netzwerk die Änderung des Netzwerkzustandes $\mathfrak{Z}$ beschrieben.

Bei Anwendung von Gleichung 2-8 wird ein Netzwerktrace τ wie folgt definiert (Gleichung 2-12):

Definition Trace für OoN-Erkennung

$$\tau = e_0, e_1, e_2 \ldots \qquad 2\text{-}13$$

mit

$$e_i \in E$$

Das bedeutet, dass ein Trace einen Teil der Spezifikation eines Systems darstellen kann.

Der Zustand des Systems, welcher auf Netzwerkebene beobachtet werden kann, hängt sehr stark von der Interaktion der einzelnen Komponenten ab. Gerade aber bei sehr stark vernetzten bzw. verteilten Systemen gibt der beobachtbare Zustand auf Netzwerkebene einen guten Anhaltspunkt über den tatsächlichen Systemstatus (Lin, Wonham 1988; Ozveren, Willsky 1990).

2.2.3 Problem nicht vollständiger Spezifikation

Die Qualität einer Spezifikation nach Definition 2-9 hängt direkt von der Aussagekraft der zugrunde liegenden Traces ab. Diese Aussagekraft eines Traces ist dabei aber auf einen Systemkontext bzw. eine Systemebene beschränkt. Sie erlaubt keinen Rückschluss über die Art und Weise, wie die verschiedenen Traces in Zusammenhang stehen.

Bei komplexen Systemen, wie es verteilte Systeme meist sind, liegen Spezifikationen häufig auf mehreren Systemabstraktionsebenen vor. Das macht es in der Umsetzung der Definition 2-9 schwer zu bewerten, ob vom System ausschließlich Traces generiert werden, welche in der Spezifikation enthalten sind. Dies liegt meist daran, dass nicht alle Abstraktionsebenen gleichzeitig erfasst bzw. beobachtet werden können. Zudem ist das Mapping zwischen den Abstraktionsebenen häufig sehr schwierig. So kann es selbst für einen Experten kompliziert sein zu bewerten, welche Kommunikationssequenz durch ein bestimmtes vom Nutzer ausgelöstes Event bzw. eine Interaktionssequenz mit dem Nutzer verursacht wird.

Es kann also durchaus vorkommen, dass bei Tests auf einer hohen Abstraktionsebene das System korrekt im Sinne der Spezifikation ausgeführt wird, aber die Ausführung in niederen Abstraktionsebenen nicht dem entspricht, wie es in der Spezifikation vorgesehen ist oder umgekehrt. In diesem Zusammenhang wird von einem Metalevel in der Spezifikation (Peti, Obermaisser, Kopetz 2005) gesprochen. Dieses Metalevel ergibt sich aus der Interaktion verschiedener Applikationen, deren Verhalten implizit von Zuständen anderer Applikationen abhängt, ohne dass diese Abhängigkeit explizit angegeben ist.

An dieser Stelle liegt auch das Problem der vollständigen Testabdeckung nach Definition 2-9 vor. Die Forderung, dass ein System nicht mehr oder andere Traces produzieren darf als die Spezifikation, soll verhindern, dass es im System kein nicht spezifiziertes Verhalten gibt. Spezifikationen sind nicht immer vollständig und eine Implementierung entspricht nicht immer der Spezifikation. Selbst wenn ein System der Spezifikation entspricht und augenscheinlich funktioniert, heißt das nicht, dass es im System kein nicht spezifiziertes Verhalten gibt. Dieses nicht spezifizierte Verhalten gehört aber ebenso wie spezifiziertes Verhalten zum Zustandsraum des Systems und ist dafür notwendig, dass ein System funktioniert, wie man es von ihm erwartet.

Nicht spezifiziertes Verhalten in einem System kann aber sehr schnell zu Problemen führen, so dass vermeintlich rückwirkungsfreie Änderungen in einem System zu Fehlern und schließlich auch zu Defekten führen. Einige Arbeiten aus dem Bereich des Softwaretestens bzw. auch des Software-Engineering beschäftigen sich daher mit der

Problemstellung, nicht spezifiziertes Systemverhalten vorhandenen Spezifikationen hinzuzufügen (Berg u. a. 2005; Werner 2010 ; Bollig u. a. 2007). Neben dem relativ hohen manuellen Aufwand, den solche Verfahren immer noch mit sich bringen, ist ihre Anwendung bei Spezifikationslücken beim Mapping zwischen den Abstraktionsebenen stark eingeschränkt. Nichtsdestotrotz zeigen diese Ansätze, dass bei der Validierung von Softwaresystemen nicht nur der Bezug zur Spezifikation zu suchen ist, sondern auch das Verhalten analysiert werden muss, welches nicht spezifiziert ist. Das entspricht in etwa dem Ansatz des „proven in use" (Betriebsbewährtheit), wie er auch in verschiedenen Safety-Normen (Schlummer 2012; ISO 26262 2009) zu finden ist. So können ein System oder eine Software, welches bzw. welche über einen längeren Zeitraum keine sichtbaren Fehler produziert haben/hat, auch ohne formellen Nachweis als zuverlässig eingestuft werden. Somit müssen nicht die sonst üblichen Validierungsnachweise erbracht werden, sondern der Nachweis einer längeren fehlerfreien Funktion.

Ausgehend von der Definition einer Spezifikation nach 2-9 und dem oben erläuterten Ansatz der Betriebsbewährtheit untersucht die vorliegende Arbeit die Möglichkeit, Netzwerktraces als Ersatz für nicht vorliegende Spezifikationen zu verwenden. Eine ähnliche Vorgehensweise, wenn auch im Anwendungsfeld der Systembeschreibung, schildert (Adzic 2011) und nennt dieses Vorgehen „Specification by example".

2.3 Grundlegende Eigenschaften des selbstlernenden OoN-Erkennungsverfahrens

Wenn das Verhalten eines Systems über einen längeren Zeitraum keine Defekte (hier im Sinne nicht erwünschten Systemverhaltens) aufweist, so wird davon ausgegangen, dass das Netzwerkverhalten während dieser Zeit als Normverhalten angesehen werden kann. Dieses Verhalten wird somit als Ersatz bzw. Erweiterung einer Spezifikation angesehen und Referenztrace genannt.

Bei näherer Betrachtung mehrerer Netzwerktraces eines verteilten Systems unter ähnlichen Randbedingungen ist festzustellen, dass es nicht ohne weiteres möglich ist zu prüfen, ob diese Netzwerktraces ein gleiches oder zumindest ähnliches Verhalten des verteilten Systems repräsentieren. Gründe dafür sind unter anderem:

1) Die Menge der Informationen und der daraus resultierenden Events ist sehr groß. Im Anwendungskontext des Fahrzeuges liegt die genutzte Übertragungsrate im Netzwerk zwischen 0,1 Mbit/s bei CAN (Zimmermann, Schmidgall 2011) und 100 Mbit/s für 100 BASE TX Ethernet (Lim, Volker, Herrscher 2011).

2) Die Menge der ausgetauschten Events zwischen den Applikationen ist sehr groß. So wurden für das Beispielszenario eines Antriebs-CAN ca. 7500 verschiedene Events identifiziert.

3) Ein Netzwerktrace besitzt keine Markierung von Sequenzen. Gewöhnlicherweise werden Traces pro Kommunikationssequenz definiert ($\tau = \boldsymbol{e_0}, \boldsymbol{e_1} \dots \boldsymbol{e_n}$). Ein Netzwerktrace kann dabei aber nicht als einzelne Sequenz angesehen werden, da die Anzahl der Events eines Netzwerktraces gegen unendlich tendiert ($n \rightarrow \infty$).

4) Gleichung 2-1 folgend besteht nicht zwischen allen Events eines Netzwerktraces eine Abhängigkeit. Somit besitzen zumindest Teile eines Netzwerktraces eine zufällige Anordnung, welche nicht reproduzierbar ist.

5) Die Randbedingungen bzw. von außen einwirkenden Events des eingebetteten verteilten Systems unterscheiden sich potenziell zwischen jeder Durchführung. Dabei sind die von außen einwirkenden Events nicht unmittelbar im Netzwerktrace sichtbar.

Die oben genannten fünf Punkte verdeutlichen die Herausforderung des Vergleichens von Netzwerktraces, welche wichtige Randbedingungen für das selbstlernende OoN-Erkennungsverfahren darstellen. Im Folgenden wird zunächst die generelle Wirkungsweise des selbstlernenden OoN-Erkennungsverfahrens erläutert. Anschließend werden die grundlegenden Problemstellungen, welche das selbstlernende OoN-Erkennungsverfahrens lösen muss, formal dargestellt und erläutert. Nach der Erläuterung der zu lösenden Probleme werden entsprechende Bewertungs- bzw. Evaluierungskriterien definiert, anhand derer die Lösungsqualität in Bezug auf die gegebenen Problemstellungen bestimmt und verglichen werden kann.

2.3.1 Wirkungsweise und Aufbau des selbstlernenden OoN-Erkennungsverfahrens

Wie bereits im Kapitel 2.1.2 eingeführt werden in der Lernphase des OoN-Erkennungsverfahrens Abhängigkeiten bzw. Klassifizierungsmodelle aus dem Referenztrace extrahiert. Im Kern besteht die Lernphase aus zwei Verfahrensschritten. Der erste Verfahrensschritt gruppiert die aufgezeichneten Events, so dass zwischen den Events in einer Gruppe eine möglichst hohe Abhängigkeit besteht. Dieser Verfahrensschritt wird im folgenden Clustering genannt. Der zweite Verfahrensschritt extrahiert bzw. lernt aus längeren Eventsequenzen Verhaltensabhängigkeiten zwischen den Events. Diese Verhaltensabhängigkeiten werden durch entsprechende Abhängigkeitsmodelle repräsentiert. Die Lernphase mit ihren zwei Verfahrensschritten ist in Abbildung 2-6 auf der linken Seite dargestellt. Die eingesetzten Lernverfahren werden in Kapitel 3 und die Clustering-Verfahren in Kapitel 4 vorgestellt.

Mit den erstellten Abhängigkeitsmodellen kann nachgelagert in der Anwendungsphase festgestellt werden, ob im zu prüfenden Netzwerktrace andere Verhaltensabhängigkeiten existieren als im Referenztrace (Abbildung 2-6, Mitte). Es wird mittels der Akzeptanzfrage festgestellt, ob das Verhalten des zu prüfenden Traces „im Referenztrace enthalten“ oder „nicht im Referenztrace enthalten“ ist. In Kapitel 5 wird die Anwendungsphase des entwickelten Verfahrens vorgestellt und die Qualität der Akzeptanzprüfung der Traces genauer untersucht.

Wird die Akzeptanzfrage negativ beantwortet, ist das Verhalten des zu prüfenden Netzwerktraces nicht im Referenztrace enthalten. In diesem Fall wird eine OoN-Indikation ausgelöst (Abbildung 2-6, rechts). Die Aufgabe der OoN-Indikation ist es, einen entsprechenden Hinweis für den Nutzer des selbstlernenden OoN-Erkennungsverfahrens auszugeben. Nach dem Auslösen der OoN-Indikation kann der Nutzer entsprechende weitergehende manuelle Prüfungen des entsprechenden Netzwerktraces durchführen.

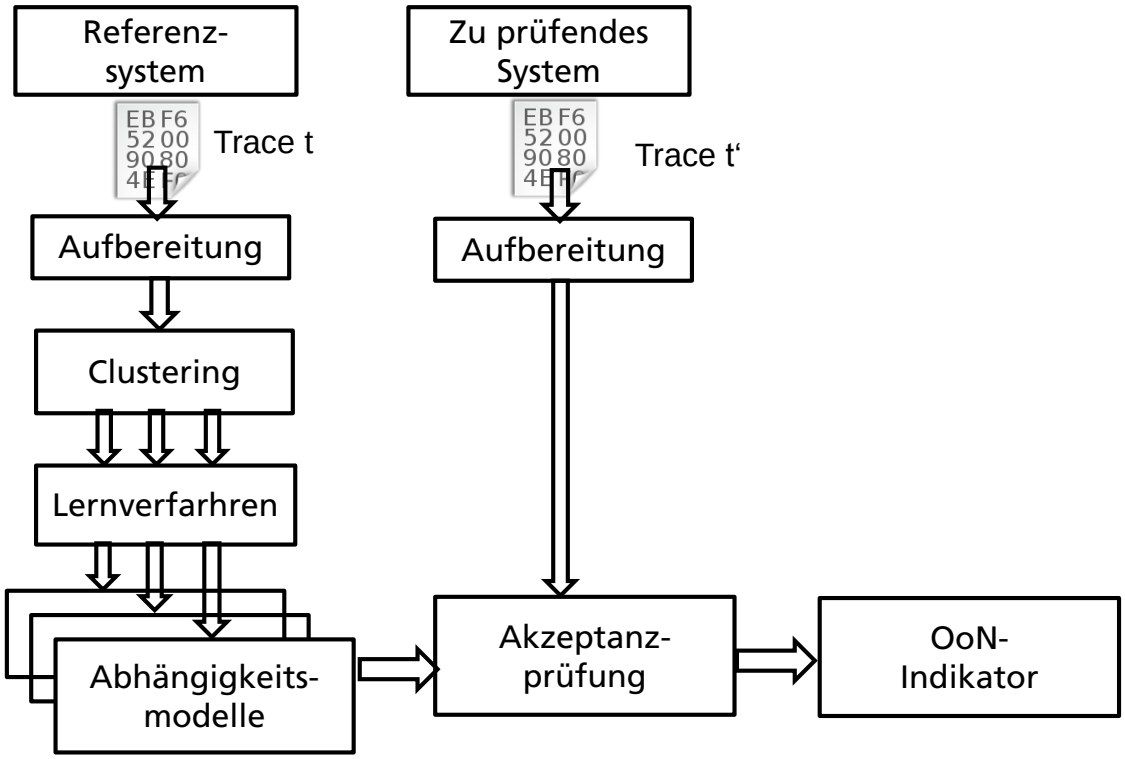

Abbildung 2-6 Wirkungsweise des selbstlernenden OoN-Erkennungsverfahrens: Lernphase (links), bestehend aus Datenaufbereitung, Clustering- und einem Lernverfahren; Anwendungsphase (Mitte): Akzeptanzprüfung weiterer Traces; OoN-Indikation (rechts): Ausgabe des Ergebnisses der Akzeptanzprüfung

Sowohl das Referenztrace als auch weitere zu prüfende Netzwerktraces müssen für ihre Verwendung einer Datenaufbereitung unterzogen werden. Nach der Datenaufbereitung arbeiten die einzelnen Verfahrensschritte generisch. D. h. ein Event ist repräsentiert durch eine eindeutige numerische Identifikation (ID), welche jeweils ein Zeichen des Alphabetes nach Definition 2-2 darstellt. Ein Trace wird somit durch die Aneinanderreihung von IDs eines Events repräsentiert (Definition 2-19). Die Übersetzung des unbearbeiteten Netzwerktraces in ein Trace nach Definition 2-19 wird in Kapitel 3.4.2 genauer beschrieben.

2.3.2 Formale Beschreibung des OoN-Erkennungsproblems

Das selbstlernende OoN-Erkennungsverfahren erstellt eine Beschreibung des Normverhaltens eines Referenzsystems auf Netzwerkebene. Anhand der Beschreibung des Normalverhaltens wird anschließend das zu prüfende System bewertet, ob es der vorgegebenen Norm entspricht. Die vorgegebene Norm wird dabei durch die Referenztraces repräsentiert, welche im Sinne der Definition 2-9 als Spezifikation Ω betrachtet werden können. Somit muss nachgewiesen werden, dass in den Netzwerktraces des zu prüfenden Systems keine Traces enthalten sind, die nicht auch in den Referenztraces enthalten sind.

Bei der OoN-Erkennung wird, im Gegensatz zur Fehlererkennung, das Normalverhalten und nicht der Fehlerfall beschrieben. Das Verhalten des zu prüfenden Systems wird als potenziell fehlerhaft klassifiziert, wenn es nicht dem Normalverhalten entspricht. Da die Anzahl und Komplexität bekannter Fehler wesentlich kleiner ist als die Komplexität des

normalen Funktionsraumes (des Normalverhaltens eines Systems), stellt die OoN-Erkennung eine besondere Herausforderung für die angewandten Lernverfahren dar. Um dies etwas konkreter darzulegen, wird im Folgenden eine Beschreibung der wichtigsten Eigenschaften und Herausforderungen für ein selbstlernendes OoN-Erkennungsverfahren gegeben.

Im Sinne einer eindeutigen Spezifikation wird in dieser Arbeit von einem diskreten endlichen Automaten (DEA) ausgegangen. Wie bereits in den Definitionen von Transduktor- und Akzeptorautomaten gezeigt wurde (Definition 2-2 und 2-3), können DEAs Eventsequenzen als Output generieren oder als Input akzeptieren. Eine Eventsequenz $e_0, e_1 \dots e_n$ (Input) muss daher eindeutig einen Zustand z identifizieren. Von diesem Zustand kann eine definierte Anzahl von Transitionen mit jeweils einem Event e_i als Outputfunktion verbunden sein (Kapitel 2.2).

Soll ein Netzwerktrace mit dem Referenztrace verglichen werden, so kann dies auch als Vergleich der zugrunde liegenden Automaten angesehen werden. Die zugrunde liegenden Automaten der Netzwerktraces sind allerdings nicht bekannt, womit ein direkter Vergleich nicht möglich ist. Das hier vorgestellte selbstlernende OoN-Erkennungsverfahren versucht daher, anhand von Lernverfahren entsprechend vergleichbare Informationen bereitzustellen.

Zur Anwendung der Lernverfahren ist es wichtig, eine Hypothese über das Systemverhalten des zu lernenden Referenzsystems aufzustellen. Auch wenn diese Hypothese nicht vollständig zutreffen sollte, so stellt sie zumindest eine sehr einfache Beschreibung von Zusammenhängen dar, welche als Basis für weitere Schlussfolgerungen verwendet werden kann. Unter dem Aspekt des Sparsamkeitsprinzips sind einfache, wenn auch nicht vollständige Hypothesen eine wichtige Basis für die Anwendungen des maschinellen Lernens. Dies wird auch als „Ockhams Rasiermesser“ bzw. „Occam's razor“ bezeichnet (Blumer u. a. 1987).

Nach Definition 2-1 können voneinander unabhängige Subsysteme existieren. Daher müssen deren Automaten A_i^* voneinander unabhängig ausgeführt werden können und deren Alphabete Σ_i^* disjunkt sein. Das Alphabet Σ_i^* muss disjunkt sein, da Ein- und Ausgabealphabet identisch sind und unabhängige Subsysteme in diesem Zusammenhang keine gemeinsamen Zeichen besitzen können. Somit wird ein Netzwerktrace τ durch den fiktiven Automaten A^{**} mit dem Alphabet Σ^{**} beschrieben, wobei A^{**} das Produkt der Automaten A_i^* darstellt (Gleichung 2-14 und Abbildung 2-7).

Produkt der Automaten A_i^*

$$A^{**} = A_0^* \times A_1^* \times A_2^* \times \dots A_i^* \qquad 2\text{-}14$$

mit

$$Z^{**} = Z_0^* \times Z_1^* \times Z_2^* \times \dots Z_i^* \qquad 2\text{-}15$$

$$\Sigma^{**} = \Gamma^{**} = \dot{\bigcup} \Sigma_i^* ; \qquad 2\text{-}16$$

$$\tau = e_0, e_1, e_2 \dots e_i, e_i \in \Sigma^{**}$$

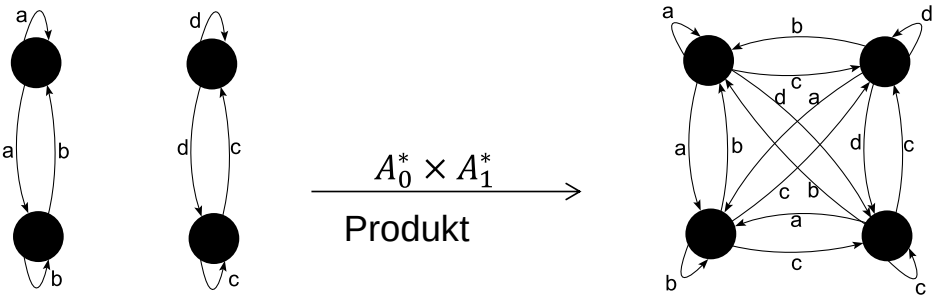

Abbildung 2-7 Beispiel eines Produktes zweier Automaten
Sind die beiden Automaten links komplett unabhängig, so ergibt sich das Produkt der Automaten (rechts). Im Unterschied zur Komposition ist hier von jedem Zustand jeder andere Zustand erreichbar.

Somit ergibt sich die Gesamtmenge der zu beobachtenden Events E aus:

$$E = \Sigma^{**} = \dot{\bigcup} \Sigma_i^* \qquad 2\text{-}17$$

Dabei sind weder die beschreibenden Automaten noch deren Zustände bekannt.

Die in dieser Arbeit aufgestellte Hypothese geht davon aus, dass das beobachtbare Verhalten eines Systems das Resultat der Ausführung von mehreren parallel ausgeführten, im Ein- und Ausgabeverhalten abhängigen Automaten A_i darstellt. Ein Automat A_i stellt die Ausführung einer Funktion dar. Die Interaktion mehrerer Funktionen kann als Komposition der Automaten A_i gesehen werden und resultiert in den Automaten A^* (Gleichung 2-18). Dabei muss die Menge der Zeichen der Ausgangsalphabete aller Automaten gleich der Menge der Eingangsalphabete aller Automaten sein (Abbildung 2-8).

Komposition der Automaten A_i

$$A^* = A_0 \circ A_1 \circ A_2 \circ \dots A_i \qquad 2\text{-}18$$

mit

$$A^* : \bigcup Z_i \mapsto Z^* \qquad 2\text{-}19$$

$$\Sigma^* = \dot{\bigcup} \Sigma_i \,;\; \Gamma^* = \dot{\bigcup} \Gamma_i \,;\; \Gamma^* = \Sigma^* \qquad 2\text{-}20$$

Der Automat A^* repräsentiert somit das Kommunikationsverhalten der interagierenden und die abhängigen Funktionen eines verteilten Systems. Es ist zu beachten, dass bei einer Komposition von Automaten keine genauen Überführungsregeln der Automaten A_i zu A^* angegeben werden können. Die Komposition besagt vielmehr, dass es sich nicht um ein vollständiges Produkt der Automaten nach Gleichung 2-14 handelt.

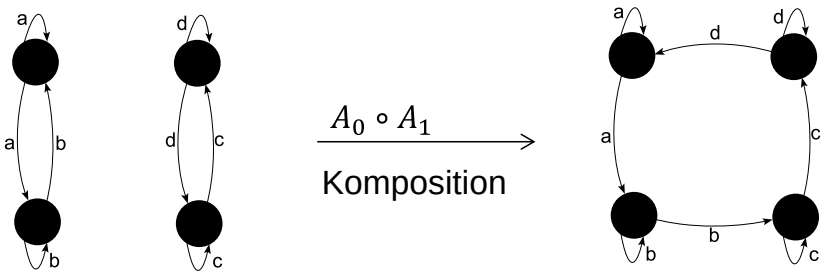

Abbildung 2-8 Beispiel der Komposition zweier Automaten
Bei der Komposition der Automaten links kann z.B. der rechte Automat entstehen. Sicher zu berechnen ist nur die rückwärtige Abbildung des rechten Automaten auf die Sub-Automaten links.

Es ist leicht zu erkennen, dass durch die Kombination unabhängiger Subsysteme (Produkt der Automaten) in einem Netzwerktrace die potenzielle Anzahl der Zustände bzw. der Transitionen des möglichen beschreibenden Automaten exponentiell wächst. Hingegen wächst bei der Komposition einzelner Funktionsautomaten zu einem resultierenden Protokollautomaten A_i^* die Anzahl der resultierenden Zustände im besten Fall weniger als linear.

Im Umkehrschluss ist davon auszugehen, dass die Komplexität eines einzelnen Subsystems exponentiell mit der Anzahl der Subsysteme im Gesamtsystem abnehmen wird. Das heißt, je mehr unabhängige Alphabete Σ_i^* in einem Referenztrace identifiziert werden können, desto einfacher wird die Aufgabe des Lernverfahrens der OoN-Erkennung. Aus diesem Grund ist das Clustering des Alphabetes Σ^{**} in disjunkte Subalphabete Σ_i^* ein wichtiger Verfahrensschritt des selbstlernenden OoN-Erkennungsverfahren.

Wären die Automaten A_i^* bekannt, so könnten diese als Spezifikation für das Netzwerkverhalten betrachtet werden. Jeder Automat kann eine Menge an Traces generieren, welche sein Verhalten eindeutig beschreiben. Im Sinne der Aufgabenstellung muss das OoN-Erkennungsverfahren beurteilen, ob ein Referenztrace von dem zugrunde liegenden Automaten hätte generiert werden können. Somit wäre der Nachweis erbracht, dass ein zu prüfendendes System keine Traces generiert, welche nicht in der Spezifikation enthalten sind.

Die Automaten A_i^* sind allerdings nicht bekannt, sondern werden durch ein Referenztrace repräsentiert. Ein Lernverfahren muss demzufolge in einer geeigneten Weise das Verhalten der Automaten A_i^* abbilden und aus dem Referenztrace extrahieren können. Diese Repräsentation sollte einen bewertbaren Rückschluss zulassen, ob das zu beurteilende Netzwerktrace ein Output des gemutmaßten Automaten A_i^* hätte sein können.

Die Lernphase des selbstlernenden OoN-Erkennungsverfahren besteht demzufolge aus den zwei Verfahrensschritten: (i) dem Clustering des Alphabetes Σ^{**} in disjunkte Subalphabete Σ_i^* und (ii) dem Extrahieren des potenziellen Verhaltens eines Automaten A_i^*. Die OoN-Erkennung wird durch die Anwendung der extrahierten Modelle des Lernverfahrens auf das zu prüfende Netzwerktrace des zu prüfenden Systems durchgeführt. Dabei wird das zu prüfende Netzwerktrace entweder als „wird vom

Modell akzeptiert“ oder „wird nicht vom Modell akzeptiert“ klassifiziert. Die Verfahrensschritte des Clusterings und des Lernens werden im Folgenden vorgestellt.

2.3.3 Formale Beschreibung des Clustering-Problems

Die Aufgabe des Clustering-Verfahrensschrittes ist es, das Alphabet Σ^{**} in disjunkte Subalphabete Σ_i^* zu unterteilen, so dass diese die Gleichung 2-16 erfüllen. Bei der Unterteilung in Subalphabete Σ_i^* wird folglich auch das Netzwerktrace τ in Subtraces τ_i unterteilt (Gleichung 2-23):

Vereinigung disjunkter Subtraces

$$\text{Wenn } \Sigma^{**} = \Gamma^{**} = \dot{\bigcup} \Sigma_i^* \text{ dann ist auch } \tau = \dot{\bigcup} \tau_i \qquad \text{2-21}$$

Die Unterteilung soll dabei so erfolgen, dass die das Trace beschreibenden Automaten möglichst nur in Form einer Komposition (Gleichung 2-18) und nicht in Form des Produktes (Gleichung 2-14) zusammengeführt sind und dabei Gleichung 2-22 erfüllen.

Beziehung Subtrace zu Subzustand und Subautomat

$$\tau_i \mapsto z_i^* \; ; \; z_i^* \in Z_i^* ; \quad Z_i^* \mapsto A_i^* \qquad \text{2-22}$$

D. h., der Clustering-Verfahrensschritt soll Events eines Traces so gruppieren, dass die funktionalen Abhängigkeiten oder auch Korrelationen der Events innerhalb eines Subtraces möglichst hoch sind.

Die grundlegende Clustering-Aufgabe wurde oben dargestellt. Das Clustering-Problem ergibt sich aus den für das Clustering-Verfahren nutzbaren Eigenschaften. Geclustert werden sollen Events, welche aus Referenztraces extrahiert werden können. Events sind durch eine eindeutige ID und einen Zeitstempel gekennzeichnet. Weder die ID noch der Zeitstempel lassen dabei einen direkten Rückschluss auf die Zugehörigkeit zu bestimmten Zuständen oder Automaten zu. Der Clustering-Verfahrensschritt wird in Kapitel 4 genauer vorgestellt. Die Herausforderung bei diesem Verfahrensschritt ist die Generierung von Eigenschaften, welche auf entsprechende Abhängigkeiten hindeuten. Diese können u.a. aus der Abfolge der Events, deren zeitlichem Verhalten und der Beziehung zu andern Events gewonnen werden.

2.3.4 Formale Beschreibung des Lernproblems

Wie in Kapitel 2.3.2 beschrieben, muss der Lernalgorithmus aus einem Referenztrace Abhängigkeiten zwischen Events extrahieren. Diese extrahierten Informationen müssen später dazu genutzt werden können, weitere Netzwerktraces der zu prüfenden Systeme dahingehend zu beurteilen , ob sie den gleichen Abhängigkeiten genügen. Im Folgenden werden die zu erfüllenden Anforderungen und Randbedingungen aufgezeigt, unter denen ein anzuwendendes Lernverfahren Abhängigkeitsmodelle aus einem Trace extrahieren muss.

(1) Keine Separation zwischen Ein- und Ausgabedaten möglich

Aus den Traces ist keine Separation nach Eingabe- und Ausgabealphabet möglich. Es ist demzufolge nicht möglich, ausgehend von einer bestimmten Eingabemenge an Events, die Systemreaktion als Ausgabemenge von Events zu beobachten.

Für Lernalgorithmen erweist sich ein derartiges Systemverhalten häufig als problematisch, da es hierdurch schwieriger ist, einzelne Lernergebnisse zu bewerten. Dies wird dadurch verstärkt, dass Lernalgorithmen meist die Eingabedaten an ein System beobachten und anhand der Ausgabedaten des Systems ihr Verhalten anpassen bzw. lernen. Dies wird auch als Batchverfahren bezeichnet.

(2) Trace enthält mehrere nicht voneinander trennbare Sequenzen

Nach der Definition eines Traces identifiziert jedes Trace genau einen Systemzustand z_i. Der Systemzustand z_i wird aber über vorhergehende Systemzustände $z_0 \dots z_{i-1}$ identifiziert. D. h. das Subtrace $\tau = e_0 \dots e_{i-1}$ identifiziert einen Systemzustand.

Diese Subtraces können auch als Sequenzen bezeichnet werden, welche eine bestimmte Systemreaktion auf ein bestimmtes Ereignis repräsentieren. Ein Trace enthält die Systemreaktion auf mehrere Ereignisse. Es ist dabei aber nicht möglich, die Systemreaktionen auf einzelne Ereignisse klar zu unterscheiden. Daher ist auch eine Zuordnung von Sequenzen zu Systemreaktionen nicht möglich. Das Trace ist sozusagen sequenzlos.

(3) Wiederholung von Sequenzen (Schleifen)

Wird ein Punkt im System gefunden, bei dem die vorhergehenden Systemzustände keinen Einfluss auf das Verhalten nach einem Zustand haben, so ist dies ein rückwirkungsfreier Zustand eines Systems. Erreicht ein System einen solchen Zustand, so ist es ebenfalls möglich, das Trace in Subtraces zu unterteilen. Beispielsweise kann ein Automat in einer Schleife Input von außerhalb des Systems verarbeiten und dabei immer wieder in seinen Startzustand zurückkehren. Es kann also davon ausgegangen werden, dass ein Zustand z_i eines Systems mehr als einmal durch ein Trace eingenommen wird. Dies ist möglich, da die Beziehung von Trace zu Zustand nur eindeutig einen Zustand identifiziert, ein Zustand aber nicht eindeutig ein Trace identifiziert. Es ist also möglich, dass ein Subtrace $\tau = e_i \dots e_{i+n}$ ausgehend von einem rückwirkungsfreien Zustand z_i, eindeutig einen Zustand z_{i+n} beschreibt.

(4) Keine Interaktion mit dem zu lernenden System möglich

Das Lernverfahren kann nicht aktiv mit dem Referenzsystem interagieren, welches die Referenztraces generiert. Viele Lernverfahren benutzen gezielt aktive Anfragen an ein System, um den daraus resultierenden Output des Systems mit dem von ihm erwarteten zu vergleichen. Da das Systemverhalten nur als Referenztrace repräsentiert ist, ist dies im vorliegenden Fall nicht möglich.

(5) Vermeidung von Überspezialisierung

Es darf keine Überspezialisierung des Lernergebnisses erfolgen. Da ein Netzwerktrace das Ergebnis der Ausführung eines Systems ist, auf das Ereignisse einwirken, welche nicht im Trace sichtbar sind, werden sich einzelne Traces in ihrer

Gesamtheit immer voneinander unterscheiden. Würde ein Lernverfahren ausschließlich das absolut gleiche Verhalten des Referenztraces akzeptieren, wäre es überspezialisiert.

2.3.5 Bewertung selbstlernender OoN-Erkennung

Um die Güte des vorgestellten selbstlernenden OoN-Erkennungsverfahren bei der Suche nach potenziellen Fehlern in Netzwerktraces beurteilen zu können, werden an dieser Stelle Bewertungskriterien eingeführt. Diese Kriterien sollen zum einen die Vergleichbarkeit verschiedener Ansätze gewährleisten und zum anderen einen Hinweis über zu die zu erwartende Aussagekraft der gelernten Modelle für die OoN-Erkennung geben.

2.3.5.1 Klassifizierungsgüte bzw. Konfidenz

Da die OoN-Erkennung eine Klassifizierung darstellt, ist es naheliegend, Bewertungsgrößen aus dem Bereich der Klassifizierung anzuwenden. Die wichtigsten Bewertungsgrößen hierbei sind die Folgenden:

Tabelle 2-1 Bewertungsgrößen für Klassifizierungsgüte oder Konfidenz

Bewertungsgröße	Erläuterung
Richtig positiv – rp	Das zu prüfende Netzwerktrace wurde als dem Referenztrace entsprechend eingestuft und diese Aussage war korrekt (korrekte Erkennung des Gleichverhaltens).
Richtig negativ – rn	Das zu prüfende Netzwerktrace wurde als nicht dem Referenztrace entsprechend eingestuft und diese Aussage war korrekt (korrekte Erkennung einer Abweichung).
Falsch positiv – fp	Das zu prüfende Netzwerktrace wurde als dem Referenztrace entsprechend eingestuft und diese Aussage war nicht korrekt (nicht erkannte Abweichung).
Falsch negativ – fn	Das zu prüfende Netzwerktrace wurde als nicht dem Referenztrace entsprechend eingestuft und diese Aussage war nicht korrekt (Fehlalarm).

Es ist davon auszugehen, dass die Anzahl der rp-Fälle den Großteil aller Klassifizierungsergebnisse darstellen wird. Der Grund hierfür liegt darin, dass der Anteil des Falschverhaltens nur einen Bruchteil des Gesamtverhaltens ausmacht. Anderenfalls wäre das vorgestellte Verfahren nicht anwendbar.

Die Größenordnungen der angegebenen Kriterien werden daher wie folgt erwartet (Gleichung 2-23):

$$rp \gg rn \approx fp \approx fn \quad \text{2-23}$$

Als kennzeichnende Relationen werden die Falsch-Negativ-Rate (Fehlalarmrate, Gleichung 2-24) und die Richtig-Negativ-Rate (Wahrscheinlichkeit, dass eine Abweichung gefunden wird, Gleichung 2-25) herangezogen.

Fehlalarmrate $$P(fn|rp) = \frac{fn}{rp+fn} \quad \text{2-24}$$

Abweichung gefunden $$P(rn|fp) = \frac{rn}{rn+fp} \quad \text{2-25}$$

Da die Auswertung der gefundenen Abweichung (OoN-Indikation des OoN-Erkennungsverfahrens) manuell vorgenommen wird, sollte die Anzahl der Fehlalarme fn möglichst klein sein. Ein Netzwerktrace kann durchaus $> 10^6$ Events aufweisen. Bei einer manuellen Bearbeitung sollte die absolute Anzahl entsprechend handhabbar bleiben. Die Richtig-Negativ-Rate sollte hingegen möglichst hoch sein, da nach Möglichkeit alle Abweichungen zwischen Referenztrace und zu prüfenden Traces gefunden werden sollen. Inwieweit es Wechselwirkungen zwischen den beiden Wahrscheinlichkeiten gibt, wird im Kapitel 3 in Bezug auf die Lernverfahren erörtert.

2.3.5.2 Blick in die Vergangenheit

Nach der Definition eines Automaten ist der nächste einzunehmende Zustand abhängig von einer gewissen Folge von Zuständen in der Vergangenheit. Alternativ kann man auch sagen, dass ein System ein „Gedächtnis" besitzt. Daraus ergibt sich die Forderung an das selbstlernende OoN-Erkennungsverfahren, zur Beurteilung eines beobachteten Zustandes, eine möglichst lange Historie des Systems einzubeziehen (Gleichung 2-26).

Beschreibung Langzeitabhängigkeit: $$e_{i-n} \dots e_i \mapsto z_i \quad \text{2-26}$$

Je größer das maximal auflösbare n des selbstlernenden OoN-Erkennungsverfahrens ist, desto besser ist die Qualität im Hinblick auf sogenannte Langzeitabhängigkeiten. Die Auflösung von Langzeitabhängigkeiten ist streng genommen ein Bestandteil der Klassifizierungsgüte des Verfahrens. Allerdings lässt sie sich mit der Definition der Klassifizierungsgüte nur ungenügend beschreiben, so dass eine dedizierte Auswertung sinnvoll ist.

2.3.5.3 Bootstrapping

Die oben genannten Bewertungskriterien lassen sich fehlerfrei nur ermitteln, wenn das zugrunde liegende System A^{**} bekannt ist. Ist dies nicht der Fall, lassen sich diese Gütekriterien möglicherweise über empirische Vorgehensweisen wie z. B. Fehlerinjektion in Traces ermitteln. Es zeigt sich aber, dass die Qualität der angewandten Lernverfahren von der Komplexität des zugrunde liegenden Systems abhängt. Somit ergibt sich eine nicht ohne Weiteres auflösbare Wechselwirkung zwischen dem zugrunde liegenden System, dem Lernalgorithmus und der Qualität der OoN-Erkennung. Um die Anwendbarkeit der gelernten Abhängigkeitsmodelle zu beurteilen, ist es demnach von Vorteil, Informationen über das zugrunde liegende System aus dem Lernprozess selbst ziehen zu können. In der Statistik wird ein derartiges Vorgehen Bootstrapping genannt. Je mehr Informationen über das zu zugrunde liegende System aus dem Lernprozess

extrahiert werden können, umso besser wird die Qualität der Lernergebnisse für ein entsprechendes Referenzsystem bewertbar sein. Bootstrapping ist somit kein direkt bewertbares Kriterium, es gibt aber einen Aufschluss darüber, wie viele Informationen ein Lernverfahren aus dem gegebenen Referenztrace extrahieren kann.

2.3.5.4 Systemabdeckung

Ein wichtiges Kriterium für jedes Testverfahren ist die Systemabdeckung. Da im vorliegenden Fall das zugrunde liegende System nicht bekannt ist, wird stattdessen die Abdeckung des im Referenztrace enthaltenen Verhaltens herangezogen. Hierzu werden eine Event-Coverage (Gleichung 2-27) und eine Trace-Coverage (Gleichung 2-28) definiert. Die Event-Coverage bezeichnet dabei den Anteil des abgebildeten Alphabetes und die Trace-Coverage den Anteil aller abgebildeten Zeichen des Traces:

Event-Coverage $$C_e = \Sigma^* / \Sigma$$ 2-27

Trace-Coverage $$C_t = |\tau(e \in \Sigma^*)| / |\tau(e \in \Sigma)|$$ 2-28

2.4 Fazit

Kapitel 2 zeigt die Anwendung, die Wirkungsweise, die Problemstellungen und die Evaluierungskriterien des in dieser Arbeit entwickelten selbstlernenden OoN-Erkennungsverfahren. Hervorzuheben sei an dieser Stelle die einfache Anwendung des Verfahrens innerhalb bestehender Auswertungsverfahren für Daten aus Fahrzeugversuchen. Ein besonderes Augenmerk gilt der aufgestellten Systemvermutung, welche eine Hypothese über zugrunde liegende Eigenschaften des Netzwerkverkehrs erlaubt. Die Systemvermutung findet vor allem Anwendung in den zwei folgenden Kapiteln, in denen geeignete Lern- und Clustering-Verfahren vorgestellt und evaluiert werden. Die Evaluierungskriterien werden hauptsächlich im Kapitel 5 bei der Evaluierung des selbstlernenden OoN-Erkennungsverfahren angewendet.

3 Lernverfahren zur Generierung von Abhängigkeitsmodellen aus Netzwerktraces

Das grundlegende Ziel und die Problemstellung des Verfahrensschrittes zum Lernen von Abhängigkeitsmodellen aus Netzwerktraces wurden bereits in Kapitel 2 erläutert. Da für das dargestellte Lernproblem keine fertigen Lösungen bekannt sind, wird die Anwendung möglicher Lernverfahren im folgenden Kapitel näher untersucht. Ein wichtiger Bestandteil der Lösungsfindung ist dabei die Erarbeitung einer Evaluierungsmethodik und damit verbundener Evaluierungskriterien. Diese Kriterien ermöglichen die Vergleichbarkeit der vorgestellten Lernverfahren und definieren zugleich die Randbedingungen, mit welchen die Lernverfahren umgehen müssen. Es zeigt sich, dass unter den gegebenen Randbedingungen nur sehr wenige Lernverfahren potenziell anwendbar und belastbare Ergebnisse zur Anwendbarkeit der infrage kommenden Verfahren in der Literatur nicht verfügbar sind. Daher wird für zwei der drei potenziellen Lernverfahren zunächst die Anwendbarkeit auf die Problemstellung experimentell nachgewiesen und entsprechende Evaluierungsergebnisse für die hier gegebene Problemstellung vorgestellt. Da die Evaluierung auf Basis künstlicher Daten erfolgt, schließt das Kapitel mit der Anwendung des potenziell besten Lernverfahrens auf reale Daten eines Fahrzeuges.

Kapitel 3 ist wie folgt gegliedert: In Kapitel 3.1 wird zunächst ein grober Überblick zu den verschiedenen Klassen möglicher Lernverfahren gegeben und das vorliegende Problem darin eingruppiert. Anschließend werden in Kapitel 3.2 die Evaluierungsmethodik vorgestellt und Evaluierungskriterien definiert, anhand derer die Bewertung der Lernverfahren erfolgt. In Kapitel 3.3 werden drei potenzielle Lernverfahren diskutiert und deren Anwendbarkeit auf das gegebene Problem untersucht bzw. hergestellt. Dazu wurden ausgewählte Verfahren implementiert und mittels festgelegter Kriterien bewertet. In Kapitel 3.4 wird das am besten geeignete Lernverfahren auf reale Datensätze angewandt. Dazu werden zunächst die nötigen verfahrenstechnischen Schritte erläutert, welche notwendig sind, um reale Datensätze für die Lernverfahren aufzubereiten. Nach der Anwendung des Lernverfahrens auf die aufbereiteten Realdatensätze werden die daraus ersichtlichen nötigen Anpassungen und Optimierungen diskutiert.

3.1 Einordnung der Problemstellung für Lernverfahren

Die Aufgabenstellung kann als Unsupervised-Learning Problem betrachtet werden (Ghahramani 2004). Es ist keine Interaktion mit dem ausgeführten System möglich (verstärkendes Lernen oder auch aktives Lernen). Es gibt keine korrespondierenden Input- und Outputdatensätze eines Systems, womit ein Lernalgorithmus den Zusammenhang zwischen Input- und Output herstellen könnte (Supervised-Learning). Der Lernalgorithmus erhält lediglich eine Menge an ungekennzeichneten Daten und soll daraus Abhängigkeiten erlernen bzw. extrahieren. Im Gegensatz zum Supervised-Learning ist beim Unsupervised-Learning eine direkte Bewertung der Abbildungs- bzw. Klassifizierungsergebnisse nicht möglich.

Geht es speziell um Lernaufgaben und weniger um Klassifizierungsaufgaben, wird häufig davon ausgegangen, dass beim Unsupervised-Learning wenig bis keine Informationen über die Mächtigkeit des Abbildungsproblems vorhanden sind. Dies resultiert daraus, dass weder der Eingaberaum (Inputdaten) noch der Lösungsraum

(Outputdaten) bekannt sind. Ursache dafür ist meist der Mangel an Wissen über funktionsspezifische Merkmale. Daher werden die Daten oft als ungekennzeichnet (unlabeled) bezeichnet.

Weiterhin stehen beim vorliegenden Lernproblem nur positive Verhaltensbeispiele als Eingangswerte für den Lernalgorithmus zur Verfügung. Es stehen jedoch keine Verhaltensbeispiele zur Verfügung, welche explizit ausgeschlossen werden können (negative Verhaltensbeispiele).

Zudem ist die Bewertung eines Events in Bezug auf seine Vorgänger (Historie) vorzunehmen, also als Sequenz einzelner Informationen, was Problemstellungen aus der Text- oder Spracherkennung ähnelt. Einige der angewandten Verfahren bzw. Methoden entstammen daher der Sprachverarbeitung (Computerlinguistik).

3.2 Evaluierungsmethodik

Wie in Kapitel 2.3.4 beschrieben, muss ein Lernalgorithmus für die hier gegebene Problemstellung ein Abhängigkeitsmodell aus einem Trace $\tau = e_0 \dots e_i$ lernen. Das Abhängigkeitsmodell muss dann für die OoN-Erkennung genutzt werden können, um weitere Traces zu prüfen .

In diesem Kapitel wird die Evaluierungsmethodik zur Beurteilung der Eignung eines Lernverfahrens für die Anwendung im selbstlernenden OoN-Erkennungsverfahren dargestellt. Die Herausforderung bei der Bewertung des Lernverfahrens ist eine nicht vollständig formal beschreibbare Datengrundlage in Verbindung mit ebenfalls nicht vollständig formal beschreibbaren Gütekriterien der Lernergebnisse. Daher werden im Folgenden heuristisch bzw. statistisch basierte Evaluierungsmethoden vorgeschlagen, mit deren Hilfe die wichtigsten Eigenschaften der Lernverfahren vergleichbar gemacht werden.

Eine einfache Evaluierungsmethodik für Anomalie-Erkennungsverfahren wird in (Maxion, Tan 2000) beschrieben. Der Fokus liegt dort auf der Erkennung von kurzen, sehr seltenen Sequenzen, welche in den Referenzdaten nicht enthalten sind. Dies wird aber der bereits definierten Anforderung an die Erkennung von Langzeitabhängigkeiten nicht gerecht und kann daher hier nicht angewendet werden.

Um die Eignung des Lernverfahrens sinnvoll evaluieren zu können, werden in dieser Arbeit ausgehend von einem einfachen Systemmodell (Kapitel 2.2.1) künstliche Referenzmodelle erstellt. Diese Referenzmodelle stellen einen imaginären Protokollautomaten dar, welcher ein künstlich erzeugtes Referenztrace erzeugen kann. Durch die genaue Kenntnis des beschreibenden Automaten ist es möglich, die Ergebnisse des Lernverfahrens in Bezug auf die Abbildung der tatsächlichen Abhängigkeiten zu bewerten. Somit können bestimmte grundlegende Eigenschaften der Lernverfahren untersucht und bewertet werden. Die Erfahrungen aus der Anwendung der Lernverfahren auf künstlich generierte Daten sollen zudem dabei helfen, Rückschlüsse auf die Anwendung mit realen Daten ziehen zu können. Dies unterstützt dabei, Eigenschaften der erlernten Modelle bei realen Daten abzuschätzen, welche nur auf Basis realer Daten nicht herleitbar sind.

3.2.1 Referenzmodelle

Referenzmodelle dienen dazu, die evaluierten Lernverfahren mit bekanntem Systemverhalten zu konfrontieren, um bestimmte Eigenschaften bezüglich ihrer Abbildungsqualität beurteilen zu können. Die Abbildungsqualität ist ein Maß, wie gut ein Lernverfahren, ausgehend von einem Referenztrace, den tatsächlich zugrunde liegenden Protokollautomaten abbilden kann. Sind die Referenzmodelle bekannt, so können sowohl die Fehlalarmrate (Gleichung 2-24) als auch die Erkennung tatsächlicher Abweichungen (Gleichung 2-25) vergleichsweise genau ermittelt werden.

Die grundlegende Hypothese zur Beschreibung des Systemverhaltens wird in Kapitel 2.2 beschrieben. Es ist naheliegend, die dort beschriebenen DEA, welche zur Beschreibung des zugrunde liegenden Systems herangezogen werden, auch für die Beschreibung der Referenzmodelle zu nutzen.

Ein in der Sprachverarbeitung genutztes Referenzmodell ist die sog. Reber-Grammatik (RG) nach (Reber 1967). Diese wird sehr häufig als Evaluierungsgrundlage für Verfahren zum Erlernen von Sprachen herangezogen. Eine etwas erweiterte Form ist die eingebettete Reber-Grammatik (ERG) nach (Cleeremans, Servan-Schreiber, McClelland 1989). Diese zielt vor allem auf das Erlernen von Langzeitabhängigkeiten ab. Zur Simulation von aneinander-gereihten Sequenzen ohne eine eindeutige Sequenztrennung (vgl. Problemstellung des Lernverfahrens in Kapitel 2.3.4) haben (Gers, Schmidhuber, Cummins 1999) die kontinuierliche eingebettete Reber-Grammatik (KERG) eingeführt. Eine Erweiterung der Problemstellung einer KERG in Form einer rekursiv verschachtelten KERG wurde vom Autor bereits in (Langer, Eilers, Knorr 2009) vorgestellt und als Advanced Reber-Grammatik (ARG) bezeichnet.

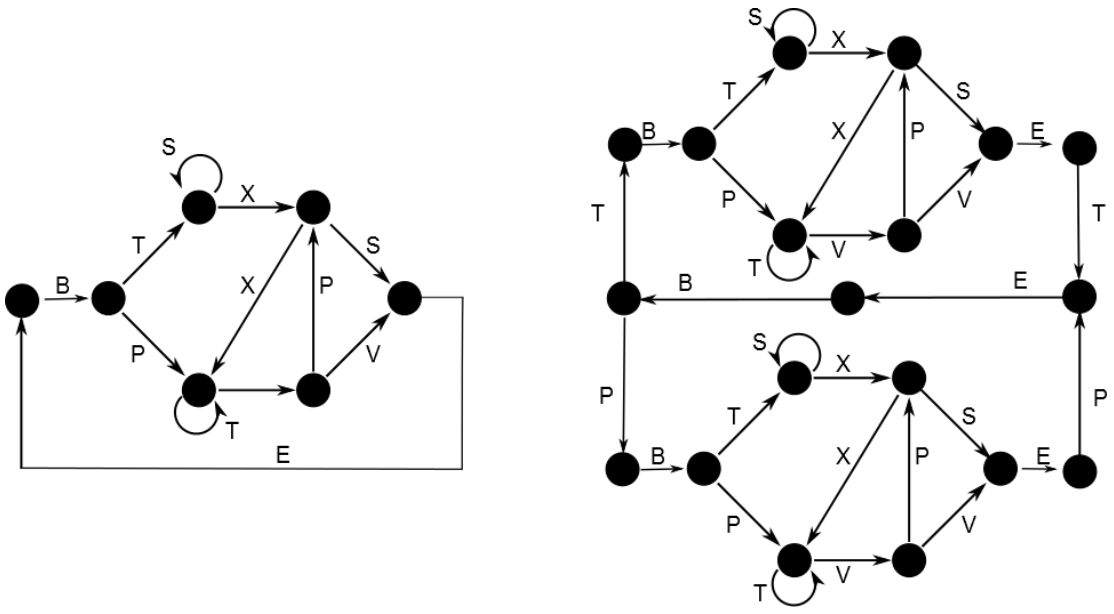

Abbildung 3-1 Darstellung der kontinuierlichen Reber-Grammatik (KRG) (links) und der kontinuierlichen eingebetteten Reber-Grammatik (KERG) (rechts)

In dieser Arbeit wird die kontinuierliche Form der Reber-Grammatik (KRG), wie in Abbildung 3-1 dargestellt, als grundlegendes Evaluierungskriterium herangezogen. Eine KRG hat eine Alphabetgröße von sieben Zeichen bei ebenfalls sieben Zuständen und 12

Transitionen. Bei zwei der Transitionen ist der Start- und Endzustand identisch, was eine direkte Schleife darstellt. Weiterhin gibt es eine größere interne Schleife. Der Startzustand ist auch der Endzustand, womit die Ausführung der Grammatik im Grunde unendlich fortgesetzt werden kann. Das Erlernen des Verhaltens einer KRG wird für die gegebene Problemstellung als Minimalanforderung angesehen.

Anhand der KERG (Abbildung 3-1, rechts), der ARG I (Abbildung 3-2) sowie der ARG II (Abbildung 3-3) soll vor allem die Erlernbarkeit von Langzeitabhängigkeiten untersucht werden. Hervorzuheben ist hierbei die variable Distanz zwischen den abhängigen Transitionen, da auf dem Ausführungsweg zwischen den Transitionen Schleifen mit beliebiger Wiederholung enthalten sind.

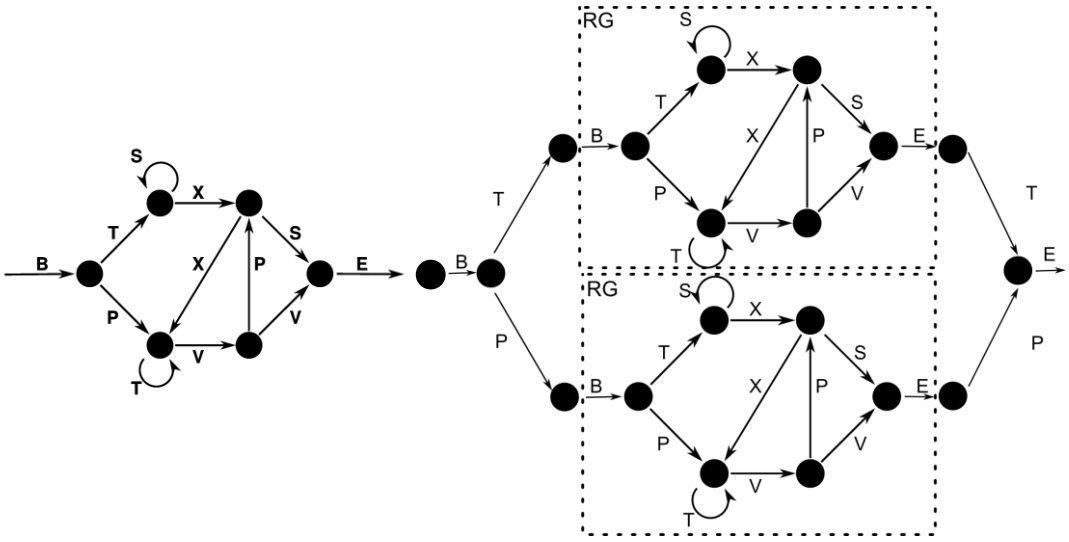

Abbildung 3-2 Darstellung der Advanced Reber-Grammatik I (ARG I)

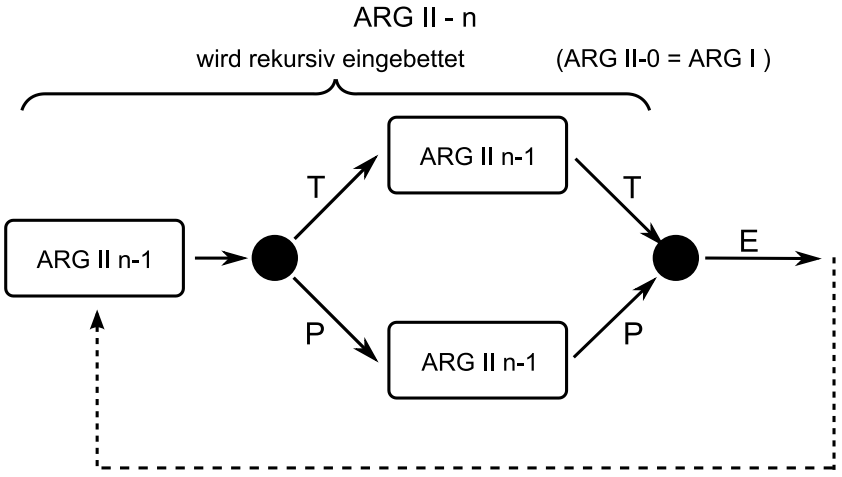

Abbildung 3-3 Darstellung der Struktur einer Advanced Reber-Grammatik II-n (ARG II-n)

Mittels der kontinuierlichen Formen der verschiedenen Reber-Grammatiken werden für die Evaluierung der Lernverfahren Zeichenströme erzeugt. Diese Zeichenströme entsprechen jeweils einem Trace τ. Für die Erzeugung der Traces wird ausgehend von

einem Zustand jeweils eine dort abgehende Transition zufällig gewählt. Dabei sind alle abgehenden Transitionen gleichberechtigt und werden mit der gleichen Wahrscheinlichkeit gewählt. Somit unterscheidet sich jeder erzeugte Trace eines Durchlaufes einer Grammatik.

Um herauszufinden, wie sich die Lernverfahren hinsichtlich der Komplexität der zugrunde liegenden Automaten verhalten, ist eine Evaluierung nur mit Reber-Grammatik basierten Modellen nicht ausreichend. Auch wenn diese unterschiedlich kaskadiert sind, bleibt doch die Komplexität des Automaten sowie die Alphabetgröße konstant.

Daher werden zur Evaluierung der Lernverfahren zudem zufällig erzeugte Automaten herangezogen. Mit der Hilfe von zufällig erzeugten Automaten ist es möglich, das Verhalten des Lernverfahrens auf unterschiedliche Alphabetgrößen, eine unterschiedliche Anzahl an Zuständen und ein unterschiedliches Verhältnis von Transitionen pro Zustand zu testen. Die zufällig erzeugten Automaten besitzen aber alle die gleiche Eigenschaft, dass sie durch mindestens eine große Schleife potenziell unendlich ausgeführt werden können (Startzustand gleich Endzustand). Es wird bei der Generierung die Anzahl der Zustände, der Transitionen und der Alphabetgröße variiert.

3.2.2 Evaluierungskriterien für die zu untersuchenden Lernverfahren

Die im Folgenden eingeführten Kriterien sollen dazu dienen, die Vergleichbarkeit der einzelnen evaluierten Lernverfahren im Sinne der Anwendung auf die spätere OoN-Erkennung zu gewährleisten. Die Anwendung bzw. Ermittlung der Evaluierungskriterien wird innerhalb dieses Kapitels mittels der aufgeführten Referenzmodelle durchgeführt (Kapitel 3.2.1).

Die wichtigste Eigenschaft für die Anwendung zur OoN-Erkennung ist die Klassifizierungsgüte, wie sie in Kapitel 2.3.5 beschrieben ist. Zur Evaluierung herangezogen werden die Fehlalarmrate (Gleichung 2-24) und die Rate für die tatsächlich erkannten Abweichungen (Gleichung 2-25). Die Bestimmung dieser Größen unter Zuhilfenahme der Referenzmodelle wird im Folgenden erläutert.

3.2.2.1 Klassifizierungsgüte

Bestimmung der Fehlalarmrate

Zur Bestimmung der Fehlalarmrate, als Teil der Klassifizierungsgüte nach Gleichung 2-24, wird dem Lernverfahren zur Bildung der Abhängigkeitsmodelle ein Referenztrace vorgelegt, welches aus einem Referenzmodell erzeugt wurde. Die Fehlalarmrate $P(fn|rp)$ wird bestimmt, indem weitere Traces mittels des Referenzmodells erstellt werden. Diese Traces werden den gelernten Abhängigkeitsmodellen zur Prüfung präsentiert. Wird ein Trace nicht akzeptiert, so ist dies ein Falsch-Negativ Fall (fn) (Tabelle 2-1). Dieser Vorgang kann mit verschiedenen Referenzmodellen durchgeführt werden.

Erkennung tatsächlicher Abweichungen

Die Rate der tatsächlich erkannten Abweichungen $P(rn|fp)$, als Teil der Klassifizierungsgüte nach Gleichung 2-25, wird nicht anhand der Referenzmodelle

bestimmt. Vielmehr gibt die unten aufgeführte Erkennung von Langzeitabhängigkeiten einen Aufschluss über tatsächlich erkennbare Abweichungen.

Eine empirische Bestimmung tatsächlich erkannter Abweichungen findet auf Basis eines reellen Referenztraces in Kapitel 5.4 statt.

3.2.2.2 Langzeitabhängigkeiten

Die Abbildung und Erkennung von Langzeitabhängigkeiten innerhalb des abzubildenden Referenztrace ist eine Teileigenschaft der Erkennungsrate für tatsächliche Abweichungen. Nach Gleichung 2-26 ergibt sich die Langzeitabhängigkeit aus der Anzahl n der Events, welche dazu beitragen, einen Zustand korrekt zu beschreiben. Bildlich betrachtet ist es die Entfernung zwischen zwei Transitionen innerhalb eines Automaten (Abbildung 3-4). Kann, wie in dieser Abbildung dargestellt, die Langzeitabhängigkeit zwischen den eingekreisten Transitionen nicht aufgelöst werden, würde die Abweichung T-P oder auch P-T nicht erkannt werden.

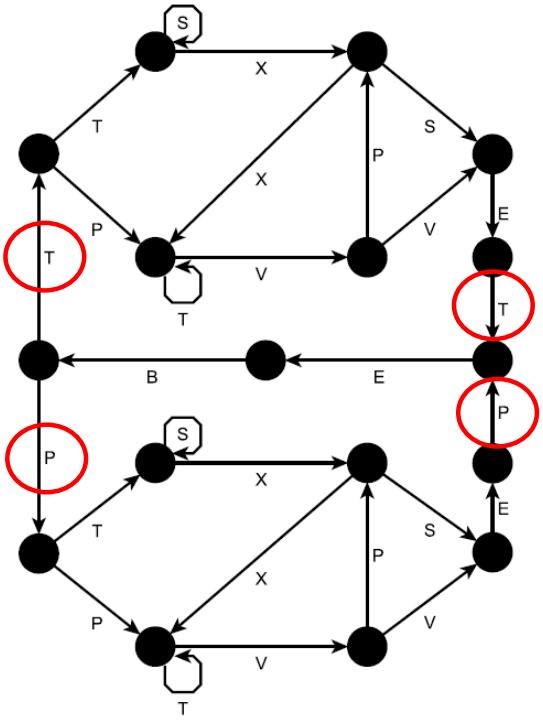

Abbildung 3-4 Darstellung einer Langzeitabhängigkeit mittels der KERG. Die eingekreisten Transitionen sind jeweils voneinander abhängig (T-T; P-P). Zwischen diesen sind aber mehrere unterschiedliche Eventverläufe möglich.

Die Überprüfung der Fähigkeit, Langzeitabhängigkeiten aufzulösen, wird durch die schrittweise Vergrößerung dieser Abhängigkeiten im Referenzmodell realisiert. Dazu werden jeweils komplexer werdende Reber-Grammatiken eingesetzt (KRG über die KERG, ARG I bis ARG II-3).

Die Bewertung, ob die Langzeitabhängigkeit aufgelöst werden kann, hängt dabei vom Lernverfahren ab. Lässt das Lernverfahren Rückschlüsse auf das zugrunde liegende System zu (siehe Bootstrapping in Kapitel 2.3.5.3), so kann das Referenzmodell direkt mit dem Abhängigkeitsmodell verglichen werden. Ist ein Rückschluss nicht möglich, so

muss über die künstliche Manipulation des erzeugten Referenztraces herausgefunden werden, ob der manipulierte Referenztrace noch akzeptiert wird oder nicht.

3.2.2.3 Einbeziehung von Wissen aus vorhandener Spezifikation

Das Ziel dieser Arbeit ist das Erlernen von Abhängigkeitsmodellen ohne die Zuhilfenahme einer Spezifikation. Nichtsdestotrotz stellt eine vorhandene Spezifikation eine Wissensbasis dar, welche zur Bewertung des Sollverhaltens einen nützlichen Beitrag leisten kann. Prinzipiell können aus einer vorhandenen Spezifikation direkt Überwachungsmodelle erstellt werden, ohne ein Lernverfahren zu nutzen.

Es ist aber vorstellbar, dass die gezielte Einbeziehung von spezifizierten oder auch explizit nicht erlaubten Verhaltensmustern die Qualität der erlernten Modelle signifikant verbessern kann. Daher sollen die Lernverfahren dahingehend evaluiert werden, inwieweit es möglich ist, Informationen aus zusätzlichen Quellen, beispielsweise einer Spezifikation, in den Lernablauf bzw. das Lernergebnis einzubeziehen.

Es wird davon ausgegangen, dass die Beschreibung des Verhaltens in der Form von Sequenzbeschreibungen des gewollten Verhaltens vorliegt. Dies scheint naheliegend, da eine sehr gebräuchliche Form der Darstellung die bereits erwähnten Message Sequence Charts (Abbildung 2-5) darstellen. Auch die Nutzung von Automaten oder Logikausdrücken ist in diesem Zusammenhang denkbar, wird in der unten ausgeführten Evaluierung aber nicht explizit berücksichtigt, da diese wiederum in Message Sequence Charts überführt werden können.

3.2.2.4 Overfitting

Overfitting ist ein bekanntes Problem beim maschinellen Lernen, welches auch als Überspezialisierung bezeichnet werden kann. Overfitting tritt dann auf, wenn das Lernverfahren zu speziell auf ein Problem trainiert wurde. In der Anwendung zur OoN-Erkennung würde das dazu führen, dass ausschließlich das Referenztrace von den erlernten Modellen akzeptiert würde und alle weiteren zu prüfenden Traces als nicht ähnlich zum Referenztrace deklariert würden.

3.2.2.5 Bootstrapping

Der Begriff Bootstrapping wurde bereits in Kapitel 2.3.5.3 erläutert. An dieser Stelle kann die dort gegebene Definition als Fähigkeit des Lernverfahrens, weitere Informationen über das zugrunde liegende System bereitzustellen, verstanden werden. Diese Informationen sind beispielsweise Aussagen über die Komplexität des Systems, aber hauptsächlich Aussagen über die Qualität der Lernergebnisse in Bezug auf die bisher vorgestellten Evaluierungskriterien (zu erwartende Fehlalarmrate, auflösbare Langzeitabhängigkeiten). Diese Erkenntnisse sind unter anderem maßgeblich für die Bereitstellung von Abbruchkriterien der Lernphase und werden zudem für die in Kapitel 4 vorgestellten Clustering-Verfahren verwendet.

Das Evaluierungskriterium Bootstrapping lässt sich dabei nicht zwangsläufig in Zahlen fassen. Da es aber die Grundlage für weitere Verfahrensschritte bildet, wird eine jeweils individuelle Evaluierung im Hinblick auf die Anwendbarkeit und Aussagekraft vorgenommen.

3.2.2.6 Abbruchkriterium für Lernverfahren

Für die Anwendbarkeit eines Lernverfahrens ist es wichtig, Kriterien zu ermitteln bzw. zu kennen, anhand derer ein definiertes Beenden des Lernverfahrens möglich ist. Viele der Lernverfahren arbeiten iterativ und nähern sich der richtigen Lösung schrittweise an. Die richtige Lösung für das Lernverfahren ist nicht bekannt, so dass über das Abbruchkriterium in irgendeiner Art und Weise entschieden werden muss, wie gut die bisher gefundene Lösung des Lernverfahrens ist. Hierfür können Informationen aus dem Bootstrapping verwendet werden oder auch andere Informationen, welche Aufschluss über den Zustand des Lernverfahrens geben.

3.2.2.7 Effizienz und Performance

Obwohl das hier vorgestellte OoN-Erkennungsverfahren nicht online arbeitet, also nicht in vorgegeben Zeiten eine Lösung erstellen muss, ist eine Aussage über die benötigte Rechenzeit oder andere Ressourcen für die praktische Anwendbarkeit wichtig. Daher sollen für die Lernverfahren Aussagen über die Performance des Lernprozesses, aber auch über Eigenschaften der erstellten Abhängigkeitsmodelle bei Anwendung zur OoN-Erkennung (Anwendungsphase) getroffen werden.

Performance bzw. Rechenkomplexität des Lernverfahrens

Im besten Falle kann die Rechenkomplexität des Verfahrens berechnet oder abgeschätzt werden. Ist dies nicht möglich, soll zumindest eine individuelle Aussage über die benötigte Rechenzeit gegeben werden. Um die Anwendbarkeit des Verfahrens zu gewährleisten, sollte die Rechenzeit auf einem derzeit handelsüblichen Computer (in den Versuchen wird ein 4 Kern à 2 GHz Prozessor verwendet) eher im Bereich von wenigen Stunden liegen.

Weiterhin ist von Interesse, wie das Verhalten des Lernverfahrens skaliert. Ein gut zu bewertendes Kriterium ist dabei, wie sich die Rechenzeit mit wachsender Länge des Referenztraces verhält. Ein weniger gut zu bewertendes Kriterium ist die Auswirkung der Komplexität der zugrunde liegenden Daten auf die benötigte Rechenzeit, da diese Komplexität bei reellen Daten nicht ohne weiteres bestimmbar ist.

Rechenkomplexität in der Anwendungsphase:

Weiterhin kann auch die Größe bzw. der Speicherverbrauch der Lernergebnisse bewertet werden. Die Lernergebnisse müssen für die nachgelagerte Anwendungsphase gespeichert werden. Besitzen die Ergebnisse eine signifikante Größe, kann dies den späteren produktiven Einsatz erschweren.

3.3 Anwendung und Evaluierung ausgewählter Lernverfahren

Nachdem im vorhergehenden Kapitel das Lernproblem, die Evaluierungsmethodik und die Evaluierungskriterien vorgestellt worden sind, werden in diesem Kapitel drei verschiedene Lernverfahren bezüglich ihrer Anwendbarkeit genauer untersucht.

In Kapitel 3.3.1 wird ein bereits bekanntes Verfahren vorgestellt, welches Markov-Modelle aus einem Referenztrace generieren kann. Zur Auswertung werden grundsätzliche Eigenschaften der Markov-Modelle bewertet und die Eigenschaften des verwendeten Lernverfahrens evaluiert.

In Kapitel 3.3.2 wird der Einsatz künstlicher neuronaler Netze zum Lernen von Abhängigkeiten aus Traces diskutiert. Dabei wird speziell der Einsatz eines rekurrenten Netzes mit sog. Gedächtniszellen vorgeschlagen. Hierzu existiert in der Literatur bereits ein Ansatz, wie das Lernverfahren auf quasi unendliche Traces angepasst werden kann. Da dieses Lernverfahren potenziell gute Ergebnisse verspricht, aber keine gesicherten Daten über die Lernergebnisse und die Performance für die hier vorliegende Anwendung vorliegen, wird mit Hilfe einer eigenen Implementierung eine ausführliche Evaluierung durchgeführt. Da sich im Rahmen der Evaluierung zeigt, dass das bestehende Lernverfahren nicht ohne Weiteres auf die Problemstellung angewendet werden kann, wird eine eigene Erweiterung des Lernverfahrens vorgeschlagen. Diese Erweiterung verbessert vor allem die Anpassung des Netzes an unterschiedliche Komplexitätsklassen des zu zugrunde liegenden Systems.

In Kapitel 3.3.3 wird ein Lernverfahren untersucht, welches diskrete endliche Automaten aus Eventsequenzen schlussfolgern bzw. erlernen kann. Hierzu wird das Lernverfahren dahin gehend erweitert, dass es in der Lage ist, quasi unendliche Traces zu verarbeiten. Das so erweiterte Verfahren wird anschließend ebenfalls einer ausführlichen Evaluierung unterzogen, so dass eine Vergleichbarkeit zu den beiden anderen untersuchten Lernverfahren hergestellt wird.

Eine abschließende Zusammenfassung der untersuchten Lernverfahren bezüglich der Eignung für das vorgestellte Lernproblem findet in Kapitel 3.3.4 statt.

3.3.1 Markov-Modelle

Markov-Ketten werden meist zur Beschreibung von gedächtnislosen stochastischen Prozessen genutzt. Eine Markov-Kette beschreibt dabei das Verhalten eines zeitdiskreten stochastischen Prozesses mit abzählbarem Zustandsraum (Waldmann, Stocker 2013; Kapitel 2). Die wichtigste Eigenschaft eines solchen Prozesses ist dabei die Gedächtnislosigkeit einer solchen Markov-Kette. D. h. das Eintreten eines bestimmten neuen Zustandes ist jeweils mit einer gegebenen Wahrscheinlichkeit nur vom aktuellen Zustand des Systems abhängig.

Der Fokus von Markov-Ketten liegt bei ihrem normalen Gebrauch eher darauf, die Wahrscheinlichkeiten der Transitionen möglichst korrekt abzubilden. Innerhalb dieser Arbeit liegt der Schwerpunkt jedoch eher auf der Erstellung der Struktur der Markov-Ketten, weniger auf deren Wahrscheinlichkeit. Um dies etwas stärker zu betonen, wird im Folgenden der sonst weniger übliche Begriff der Markov-Modelle verwendet.

3.3.1.1 Anwendung auf Problemstellung

Die Anwendung von Markov-Modellen zur Abbildung von Abhängigkeiten aus Referenztraces wird u. a. in (Jha, Tan, Maxion 2001) beschrieben. Der dort vorgestellte Algorithmus ist darauf ausgelegt, mehrere kürzere, abgeschlossene Sequenzen (wie z. B. AABC, ABCBC) als Trainingsset für das Normalverhalten zu verarbeiten. Der Algorithmus wird in (Maxion, Tan 2002) dahingehend erweitert, dass er mit längeren Traces ohne Sequenzgrenzen umgehen kann. Eine weitere Evaluierung des auf dem vorgeschlagenen Markov-Modell basierenden Verfahrens findet in (Zandrahimi, Zarei, Zarandi 2010) statt.

Grundlegend wird bei dem beschriebenen Verfahren davon ausgegangen, dass eine Sequenz von Events einen Zustand identifiziert. Ein solcher Zustand wird dann jeweils im

Markov-Modell als Zustand etabliert ($e_i \dots e_{i+n} \mapsto z_j$). Die Erweiterung nach (Maxion, Tan 2002) sieht eine sog. Sliding-Window-Technik vor. D. h. es wird aus dem Referenztrace jeweils eine Sequenz der Länge n, mit einer festen Länge beginnend, an Position i entnommen. Der nächste Zustand ergibt sich aus der Position $i+1$ und der darauf folgenden Eventsequenz der Länge n (Gleichung 3-1 und 3-2).

$$e_{i+1} \dots e_{i+1+n} \mapsto z_{j+1} \qquad 3\text{-}1$$

mit

$$z_j \xrightarrow{e_{i+1+n}} z_{j+1} \qquad 3\text{-}2$$

In einem ersten Schritt werden alle möglichen Sequenzen der Länge n aus dem Referenztrace extrahiert. Diese n-Tupels definieren die Zustände des Markov-Modells. Für eine Sequenz ABCDABCD würde sich für eine Tupel-Länge von n=2 ein Modell wie in Abbildung 3-5 ergeben. Wie dort zu sehen ist, werden alle möglichen Transitionen zwischen den Zuständen vorgesehen.

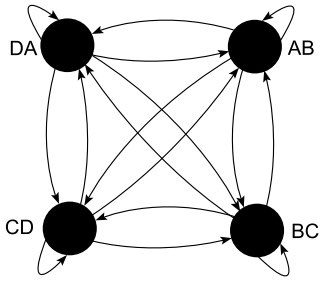

Abbildung 3-5 Markov-Modell für die Sequenz ABCDABCD mit der Window-Größe n=2

In einem zweiten Schritt werden dann die Wahrscheinlichkeiten für die Transitionen, welche durch Gleichung 3-2 vorgegeben sind, errechnet. Dazu wird das zugrunde liegende Referenztrace noch einmal mittels des zuvor ermittelten Modells verarbeitet. Dabei wird gezählt, wie häufig jede Transition durchlaufen wird. Die Wahrscheinlichkeit für eine Transition $t(z_j, z_{j+1})$ (für $z_j \xrightarrow{e_{i+1+n}} z_{j+1}$) ergibt sich dabei nach (Jha, Tan, Maxion 2001) aus der Anzahl der Ausführungen dieser Transition in Relation zur Summe der Ausführungen aller abgehenden Transition von z_j (Gleichung 3-3).

Wahrscheinlichkeit einer Transition nach (Jha, Tan, Maxion 2001)

$$P\big(t(z_j, z_{j+1})\big) = \frac{\sum\big(t(z_j, z_{j+1})\big)}{\sum\big(t\big(z_j\big)\big)} \qquad 3\text{-}3$$

Nach diesem zweiten Schritt steht ein Markov-Modell zur Verfügung, welches für jede Transition eine Wahrscheinlichkeit besitzt. Dieses Modell kann nun genutzt werden, um weitere Traces zu prüfen. Die Vorgehensweise wird auch als Klassifikation bezeichnet. Dazu wird eine Metrik definiert, anhand derer entschieden wird, ob weitere zu prüfende Traces ähnlich oder weniger ähnlich im Vergleich zum Referenztrace sind.

Diese Metrik wird in (Jha, Tan, Maxion 2001) als Diskrepanz bezeichnet. Zur Berechnung der Diskrepanz werden unterschiedliche Vorgehensweisen angeführt. In Bezug auf die Problemstellung der OoN-Erkennung scheint die vorgeschlagene Vorgehensweise von (Maxion, Tan 2002) am besten geeignet. Dort wird ein Überraschungsfaktor eingeführt,

der je bewerteter Transition berechnet wird und sich aus 1 minus der Wahrscheinlichkeit der ausgeführten Transition ergibt (Gleichung 3-4).

Überraschungsfaktor einer zu prüfenden Transition nach (Maxion, Tan 2002)

$$\mu = 1 - P\big(t(z_j, z_{j+1})\big) \qquad 3\text{-}4$$

Der Überraschungsfaktor nach (Maxion, Tan 2002) bewertet nur jeweils eine Transition. Es gibt nun die Möglichkeit, jede Transition einzeln zu bewerten oder die Summe bzw. den Durchschnitt über alle ausgeführten Transitionen zu bilden. Zur Prüfung des Traces ist es nun notwendig, Grenzwerte zu definieren, bei deren Überschreitung ein Trace als nicht mehr ähnlich zum Referenztrace zu bewerten ist (Festlegung einer Überraschungsgrenze).

Die allgemeingültige Bestimmung von Grenzwerten, ausgehend von einem Referenztrace, ist hier schwierig. In (Jha, Tan, Maxion 2001) wird dies über eine spezielle Berechnungsvorschrift gelöst, welche zwar allgemeingültig ist, aber nicht für unterschiedlich lange Traces geeignet ist und somit nicht auf die hier vorliegende Problemstellung angewendet werden kann. Die Bewertung nach (Maxion, Tan 2002) wird pro Transition vorgenommen, was zwar unabhängig von der Länge des zu prüfenden Traces ist, aber durch die schwierig auszuwertende Wahrscheinlichkeit keine belastbare Aussage über Gleichheit des Verhaltens zulässt.

3.3.1.2 Evaluierung für die Anwendung im selbstlernenden OoN-Erkennungsverfahren

In diesem Kapitel werden die Ergebnisse der Anwendung des Lernverfahrens für Markov-Modelle zusammengefasst und eine Evaluierung gemäß den aufgestellten Kriterien aus Kapitel 3.2.2 durchgeführt.

Klassifizierungsgüte

Erkennung tatsächlicher Abweichungen

Die Anwendung von Markov-Modellen für ein selbstlernendes OoN-Erkennungsverfahren ist rein konzeptuell möglich. Die gelernten Markov-Modelle können Traces beliebiger Länge klassifizieren bzw. prüfen. Die Evaluierungen in (Maxion, Tan 2002) und (Zandrahimi, Zarei, Zarandi 2010) bescheinigen dem Verfahren eine vollständige Abdeckung des Lösungsraumes, also keine sog. blinden Flecken. Bei den Untersuchungen von (Jha, Tan, Maxion 2001) konnten alle Referenztraces ohne Fehlalarme klassifiziert werden, was für statistische Ansätze eine hervorzuhebende Eigenschaft ist. Eine Aussage über die Erkennung tatsächlicher Abweichungen in realen Daten ist nicht vorhanden. Grundsätzlich ist die Erkennung tatsächlicher Abweichungen sehr davon abhängig, wie die entsprechenden Grenzwerte gewählt werden. Hier ist eine direkte Abhängigkeit zwischen der Fehlalarmrate und der Erkennung tatsächlicher Abweichungen festzustellen.

Fehlalarmrate

Konzeptuell ergibt sich eine gewisse Anfälligkeit für Fehlalarme, da selbst seltene Sequenzen in den Referenztraces, welche zulässiges Verhalten darstellen, von Markov-Modellen abgewiesen werden würden, wenn die Vertrauensgrenze entsprechend hoch

gesetzt würde. Um Fehlalarme dieser Art auszuschließen, müsste die Bewertung auf eine 0%-Schranke gesetzt werden. Das würde aber dem Konzept von Markov-Modellen nur bedingt gerecht werden und eher einer binären Beobachtungstabelle gleichen. Weiterhin kann die Berechnung der Wahrscheinlichkeiten der Übergänge für kleinere Datenmengen mit einem großen Alphabet zu Problemen führen. In (Maxion, Tan 2002) und (Zandrahimi, Zarei, Zarandi 2010) wird davon ausgegangen, (a) dass ein Trace im Großteil eine immer wiederkehrende Sequenz aufweist und (b) dass so viele Lerndaten vorliegen, dass selbst Sequenzen mit geringer Auftretenswahrscheinlichkeit noch hinreichend oft im Referenztrace enthalten sind. Wird Bedingung (b) nicht erfüllt, so liefert die Berechnung nach Gleichung 3-3 nur sehr grob abgestufte Werte, welche nicht statistisch abgesichert sind. Dieses Problem verschärft sich noch, wenn das Verhalten eines Systems nicht Bedingung (a) erfüllt.

In (Maxion, Tan 2002) wird eine Anomaliegrenze definiert, ab der eine bestimmte Sequenz als nicht normal eingestuft wird. (Maxion, Tan 2002) setzten diese für ihre Versuche auf 1 %. Dies heißt aber auch, dass Sequenzen, die seltener als 1 % in den Referenzdaten vorkommen, Fehlalarme produzieren werden.

Eine Evaluierung der Fehlalarmrate findet in (Jha, Tan, Maxion 2001) statt. Dort wird eine Fehlalarmrate von 0 bis 10 % ausgewiesen, wobei drei der neun untersuchten Testszenarien keine Fehlalarme ausgelöst haben. Aufgrund der mangelnden Diversifizierung sind allerdings keine weiteren gesicherten Aussagen über die Fehlalarmrate möglich.

Langzeitabhängigkeiten

Das Windowing-Konzept lässt sich theoretisch auf eine beliebige Window-Breite vergrößern, wobei die Vergrößerung des Windows durch die zur Verfügung stehenden Referenzdaten begrenzt wird. Durch die stark zunehmende Anzahl an Zuständen nimmt die Klassifizierungsgüte aufgrund der mangelnden statistischen Absicherung bei gleicher Referenzdatengröße ab.

In (Jha, Tan, Maxion 2001) wurde für die Untersuchung an realen Daten eine Window-Breite von max. 45 gewählt. In (Maxion, Tan 2002) wurde hingegen nur eine Window-Breite von 15 gewählt. Da (Maxion, Tan 2002) das besser vergleichbare Szenario in Bezug auf die OoN-Erkennung aufweist, wird für die Anwendung in der OoN-Erkennung mit einer ähnlichen Größenordnung gerechnet werden müssen.

Einbeziehung von Wissen aus vorhandener Spezifikation

Markov-Modelle werden tendenziell eher selten zur Spezifikation von technischen Systemen verwendet, vielmehr werden sie zur Beschreibung von wahrscheinlichkeits-basierten Beobachtungen oder Ereignissen genutzt.

Werden die Übergangswahrscheinlichkeiten der Markov-Modelle auf 0 bzw. 1 gesetzt, scheint es nicht völlig ausgeschlossen, dass erlaubte und nicht erlaubte Zustände aus anderen Informationsquellen einbezogen werden können. Das größte Problem ist aber die Codierung des Modells bzw. der Zustände und Übergänge. In dem erlernten Markov-Modell wird für jede mögliche Event-Kombination in der gegebenen Window-Breite ein eigener Zustand erstellt. Das Lernverfahren bewertet dann, inwieweit die Transitionen zwischen den verschiedenen Zuständen erlaubt sind.

Sollen vorgegebene Sequenzen aus einer Spezifikation in das Markov-Modell aufgenommen werden, so ist es notwendig, ein genaues Mapping zwischen den erlernten Zuständen in der Window-Breite und der gegebenen Sequenz herzustellen. Ist dies möglich, so können einzelne Transitionen jeweils als erlaubt (1) oder nicht erlaubt (0) gekennzeichnet werden. Weiterhin könnte es auch notwendig sein, neue Zustände zu dem erlernten Markov-Modell hinzuzufügen, falls die gegebene Sequenz nicht im Referenztrace enthalten war. Hierbei tritt aber die Schwierigkeit auf, dass entweder abgehende oder hinführende Transitionen zum neu hinzugefügten Zustand nicht aus der Sequenz oder auch dem Referenztrace extrahiert werden können. Somit könnte ein nicht brauchbares Modell entstehen und die entsprechende Information kann nicht integriert werden.

Es lässt sich festhalten, dass durchaus die Möglichkeit besteht, zusätzliche Informationen in den Lernprozess einzubeziehen. Zu diesem Zweck muss aber das Lernverfahren in einem erheblichen Maße erweitert werden. Zudem besteht das Risiko, dass nicht alle Informationen übernommen werden können.

Overfitting

Ein Overfitting der Markov-Modelle entsteht hauptsächlich durch zu strikt gewählte Bewertungsfunktionen. Allerdings kann dieser Effekt durchaus gewünscht sein, da hierdurch, wie oben beschrieben, die Fehlalarmrate reduziert werden kann.

Sonstige Overfitting-Eigenschaften, wie beispielsweise eine Initiierungsphase wie bei den LSTM-Netzen (Kapitel 3.3.2.2), sind bei den Markov-Modellen nicht zu erwarten.

Bootstrapping

Es ist prinzipiell möglich, über die Anzahl der erstellten Zustände im Markov-Modell einen indirekten Rückschluss auf die Systemkomplexität zu ziehen. Die Komplexität eines Systems ist allerdings auch abhängig davon, wie die Zustände vernetzt sind. Somit muss auch die Anzahl der Transitionen des Modells bewertet werden, welche oberhalb der Überraschungsgrenze liegt.

Es ist nicht möglich, anhand der Lernergebnisse Rückschlüsse über die Klassifizierungsgüte zu treffen.

Abbruchkriterien für Lernverfahren

Die Erstellung der Markov-Modelle inklusive der Wahrscheinlichkeitsberechnung und der Bestimmung der Bewertungseigenschaften stellt ein terminiertes Verfahren dar. D. h. das Lernverfahren ist nicht iterativ und muss daher auch nicht abgebrochen werden.

Ein Problem bei der Erstellung der Markov-Modelle stellt die Bestimmung der eingesetzten Window-Breite dar. Da es keine feste Vorschrift zur Berechnung der Window-Breite gibt, müssten Modelle mit unterschiedlicher Window-Breite angelegt werden. Die Bestimmung der besten Window-Breite ist nur mittels Evaluierung des erstellten Markov-Modells gegenüber weiteren Referenzdaten möglich und wäre wiederum iterativ. Somit müssen bis zu einer maximal zu prüfenden Window-Breite alle Markov-Modelle erstellt und bewertet werden.

Effizienz und Performance

Performance bzw. Rechenkomplexität des Lernverfahrens

Die Komplexität des Rechenaufwandes für die Lernphase ist abhängig von der Länge n des Referenztraces und der Anzahl der gefundenen n-Tupels (Anzahl der Zustände $|Z|$), woraus sich $\mathcal{O}(\mathrm{n}|Z|)$ ergibt. Die Anzahl der n-Tupels wächst allerdings mit steigender Window-Breite i mindestens exponentiell. Das bedeutet einen exponentiell steigenden Rechenaufwand mit der Window-Breite $\mathcal{O}(n^i)$.

Rechenkomplexität in der Anwendungsphase

Der Funktionsraum wächst mindestens exponentiell mit der Alphabetgröße und mit der Window-Breite. Dass selbst für die künstlichen Daten zur Evaluierung der Methodik die Anzahl der Zustände exponentiell wächst, haben (Zandrahimi, Zarei, Zarandi 2010) gezeigt. Da die Anzahl der Zustände mit dem benötigten Speicher korreliert, ergibt sich ein exponentielles Wachstum in Abhängigkeit von der Window-Breite für den benötigten Speicherbedarf pro Markov-Modell. (Zandrahimi, Zarei, Zarandi 2010) haben für eine Window-Breite von 15 ein Markov-Modell mit ca. $4*10^5$ Zuständen und $1{,}7*10^9$ Speicherelementen ermittelt. Vor allem der hohe Speicherbedarf kann bei der Erstellung mehrerer benötigter Markov-Modelle problematisch bei der Anwendung in der Praxis werden.

3.3.2 Künstliche neuronale Netze

Künstliche neuronale Netze (KNN) werden sehr oft zum Lernen bzw. zur Klassifizierung bestimmter Verhaltensmuster eingesetzt, häufig dann, wenn eine exakte Beschreibung der Muster entweder nicht vorhanden oder nicht ohne weiteres berechenbar ist. Dies wird häufig zur Diagnose komplexer mechatronischer Systeme, wie sie in der Robotik (z. B. Vemuri, Polycarpou 1997), Fahrzeugtechnik (z. B. Atkinson, Long, Hanzevack 1998) oder auch Avionik (z. B. Chen, Lee 2002) zu finden sind, eingesetzt.

Eine der herausragenden Fähigkeiten von KNN ist es, auf eine sehr kompakte Art und Weise komplexe Zusammenhänge wiederzugeben. Vor allem im Vergleich zu den im vorhergehenden Kapitel beschriebenen Markov-Modellen wird dies für diese Arbeit als Vorteil angesehen, da Markov-Modelle beim Auftreten von vielen Netzwerkzuständen sehr schnell sehr groß werden.

3.3.2.1 Anwendung auf Problemstellung

Für die Anwendung eines KNN auf die Problemstellung kommen prinzipiell zwei Verfahrensweisen in Frage. Die erste Verfahrensweise entspricht der Anwendung eines Sliding-Window, welches über das Referenztrace gelegt wird und das KNN aus der Eventfolge $e_i \dots e_{i+n}$ mögliche Events e_{i+n+1} bestimmen muss. Hierbei stellt n die Größe des Fensters dar und somit auch die Länge der berücksichtigten Historie. Eine zweite mögliche Anwendung liegt darin, dem KNN jeweils nur ein Event e_i aus dem Referenztrace zu repräsentieren. Nach einer gewissen Initiierungsphase wird das KNN dann jeweils den Nachfolger e_{i+1} berechnen. Für diesen Fall müssen KNN mit Gedächtnisfunktion verwendet werden, so dass die Eventfolge $e_{i-n} \dots e_i$ genutzt wird, um ein mögliches e_{i+1} zu berechnen. Die Größe von n ergibt sich dabei aus der Gedächtniskapazität des KNN.

Die Anwendung des Sliding-Window entspricht in etwa dem Anwendungsszenario der Markov-Modelle (Kapitel 3.3.1). Im Zusammenhang mit KNN wird dies meist bei der Vorhersage von Zeitreihen verwendet. Häufig wird in diesem Kontext auch von der Walk-Forward Methode gesprochen (vgl. Kohzadi u. a. 1996; Hu u. a. 1999). Für die Anwendung des Sliding-Window, bei der sich das Gedächtnis ausschließlich aus der Breite des Sliding-Windows ergibt, ist die Verwendung eines Feedforward Netzes (FFN) eine naheliegende Auswahl.

Im Hinblick auf die Abbildungsgenauigkeit können die zu erwartenden Ergebnisse dieser Vorgehensweise nicht besser sein als die der Lernverfahren für Markov-Modelle (die Abbildungsgenauigkeit liegt dort, wie bereits beschrieben, bei 100 %). Weiterhin werden die Eigenschaften im Hinblick auf Langzeitabhängigkeiten verfahrensbedingt ebenfalls nicht besser sein. Somit kann das Sliding-Window-Verfahren unter der Anwendung durch FFN keine besseren Ergebnisse als das Markov-Modell-Lernverfahren produzieren.

Alternativ soll die Anwendung von KNN, welche in der Lage sind, durch ihre interne Struktur Langzeitabhängigkeiten zu lernen, an dieser Stelle näher betrachtet werden. Wegen des Verzichts auf ein Sliding-Window verspricht dieser Ansatz andere Ergebnisse im Vergleich zu Markov-Modellen.

Sowohl Rekurrente Netze (RNN) als auch Feedforward Netze (FFN) sind in der Lage, durch ihre Netzkonstruktion Langzeitabhängigkeiten durch eine Art „Gedächtnis" abzubilden. Die Anwendung von FFN auf das Erlernen einer ERG (Kapitel 3.2.1) haben (Schreiber, Cleeremans, McClelland 1988) demonstriert. Einige Vergleiche mit RNN finden sich u. a. in (Cleeremans, Servan-Schreiber, McClelland 1989; Ring 1993) und (Gers 2001). Dabei lässt sich grundlegend sagen, dass FFN durch ihre einfachere Lernprozedur in der Regel einen geringeren Lernaufwand und weniger Lerndaten benötigen. Demgegenüber steht aber die schlechtere Fehlalarmrate der FFN, wie (Ring 1993) in der Zusammenfassung konstatiert und was auch durch neuere Untersuchungen auf diesem Gebiet bestätigt wird (Brezak u. a. 2012).

Long short term memory Netze

Eine sehr vielversprechende Gruppe der RNN sind die sog. Long short term memory (LSTM) Netze, wie sie in (Hochreiter, Schmidhuber 1995) und (Hochreiter, Schmidhuber 1997) vorgestellt werden. Diese sind explizit für das Erlernen von Langzeitabhängigkeiten entwickelt worden. Ihr wichtigstes Merkmal bilden dabei die sogenannten Gedächtnisblöcke (engl. Memory Blocks), welche die herkömmlichen Neuronen des RNN ersetzen.

Das von (Hochreiter, Schmidhuber 1995) vorgestellte LSTM kann nur im klassischen Batchverfahren betrieben zu werden. Im Batchverfahren wird jeweils eine Datensequenz auf das KNN angewandt und danach das KNN neu initialisiert. Dies ist vor allem für die Gruppe der RNN, also auch dem LSTM, wichtig, um deren Rückkoppelungsknoten auf eine neue Datensequenz vorzubereiten. Das in dieser Arbeit entwickelte Verfahren wird aber auf eine unendliche Sequenz ohne Sequenzgrenzen angewendet. Daher ist die Anwendung des Batchmodus für das vorliegende Problem nicht möglich.

Für die Vorhersage von kontinuierlichen Sequenzströmen haben (Gers, Schmidhuber, Cummins 2000) eine Erweiterung vorgestellt. Dort wird der LSTM-Gedächtnisblock um ein sog. Forget-Gate erweitert. Somit ist es möglich, LSTM-Netze auf die gegebene Problemstellung einer unendlichen Sequenz anzuwenden. Weiterhin zeigt (Gers 2001), dass LSTM-Netze im Vergleich zu klassischen RNN wesentlich besser in Bezug auf Fehlererkennungen abschneiden.

Anwendbarkeit von LSTM

Eine erste Untersuchung der Anwendbarkeit auf die Problemstellung der Fehlererkennung durch Lernen aus Referenztraces im Sinne einer OoN-Erkennung wurde vom Autor selbst in (Langer, Eilers, Knorr 2009) vorgenommen. Im Vergleich zur ursprünglichen Anwendung von LSTM-Netzen in kontinuierlichen Strömen nach (Gers, Schmidhuber, Cummins 2000) wurde das Lernproblem mit der Einführung der ARG nochmals verschärft (Kapitel 3.2.1). Die Implementierung von (Gers, Schmidhuber, Cummins 2000) liefert teilweise sehr gute Ergebnisse in Bezug auf die Fehlerquote. Es wird allerdings von einer sehr großen Anzahl an Referenztraces ausgegangen (30.000 Training-Traces mit jeweils 100.000 Zeichen, was insgesamt $3*10^9$ Zeichen entspricht). In Bezug auf das in dieser Arbeit vorliegende Problem sind diese Zahlen nicht realistisch. Weiterhin geben (Gers, Schmidhuber, Cummins 2000) nur eine sehr vage Unterteilung der Qualität der Vorhersage in sehr gut, gut und schlecht an.

Daher wurde eine eigene Implementierung des vorgeschlagenen Verfahrens nach (Gers, Schmidhuber, Cummins 2000) vorgenommen. Die Ergebnisse der Evaluierung sind in Tabelle 3-1 dargestellt. Wie zu sehen ist, nähert sich das Verfahren mit zunehmender Länge des Referenztraces einer Fehlerrate von 0 an. Interessant dabei ist, dass die Fehlerrate fast umgekehrt logarithmisch im Verhältnis zur Länge des Referenztraces abnimmt (Abbildung 3-6).

Tabelle 3-1 Experimentelle Ergebnisse bei der Anwendung von LSTM auf die Problemstellung.

Länge des Referenztraces[10^3]	1. Fehler [10^3]	10. Fehler [10^3]	Fehlerrate [10^{-3}]
387,4	3,3	39,6	37,61
586,9	84,9	1.121,6	2,43
2.199,5	1.316,5	5.382,8	0,28
5.335,5	26.456,3	80.114,8	0,01

Weiterhin muss untersucht werden, wie die LSTM-Netze mit steigender Komplexität des Referenzsystems umgehen. Damit verbunden ist auch die Entfernung, welche Langzeitabhängigkeiten haben dürfen, um vom LSTM Netz abgebildet zu werden.

Zu diesem Zweck wurde in Kapitel 3.2.1 die ARG II-n eingeführt. Die ARG II-n bildet eine rekursiv erweiterbare ERG-KERG Kombination ab, wobei n die Rekursionstiefe

angibt. Es wurden dabei Untersuchungen bis ARG II-3 durchgeführt. Eine ARG II-3 enthält insgesamt 63 ERG mit unterschiedlichen Langzeitabhängigkeiten. Der minimale Abstand der größten Langzeitabhängigkeit, welche es zu erlernen gibt, beträgt somit ca. 130 Zeichen. Diese Anzahl bezieht sich auf die Annahme, dass keine der Schleifen einer ERG mehrmals durchlaufen wird, und ist damit die minimale Größe. Bei mehrmaliger Ausführung der Schleifen kann sich dieser Abstand noch vergrößern. In den durchgeführten Versuchen konnte für die ARG II-3 ein mittlerer Abstand von ca. 800 Zeichen festgestellt werden (Tabelle 3-2: bei 10^5 Durchläufen und $8*10^8$ Inputzeichen ergeben sich im Durchschnitt 800 Zeichen pro Durchlauf einer ARG II-3).

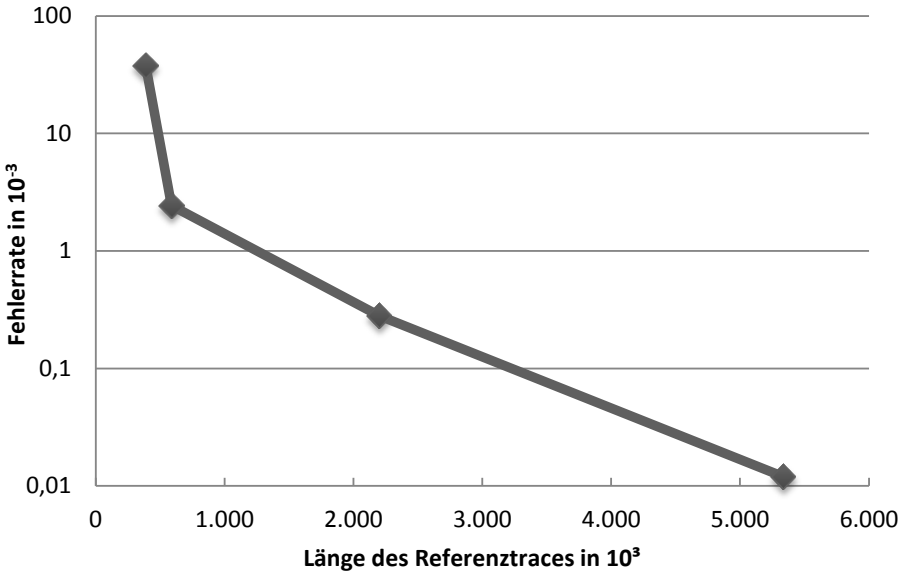

Abbildung 3-6 Entwicklung der Fehlerrate im Verhältnis zur Länge des Referenztraces bei der Anwendung von LSTM-Netzen auf KERG

Tabelle 3-2 Skalierung der LSTM Netze auf schwierigere Lernprobleme mit größeren Langzeitabhängigkeiten

ARG- Typ und LSTM-Struktur [Anzahl der Gedächtnisblöcke (Anzahl der Gedächtniszellen)]	Durchläufe ARG	Länge des Referenz-traces in 10^3	Position des 10. Fehlers in 10^3	Anteil der erfolgreichen Lernversuche in %
KERG [4(2)]	100.000	3.414	80.114	100
ARG I [4(2)]	100.000	3.653	34.092	60
ARG II-1 [4(2)]	100.000	15.081	38.743	48
ARG II-2 [5(2)]	100.000	35.552	12.141	10
ARG II-2 [8(2)]	100.000	35.472	100.000	10

ARG- Typ und LSTM-Struktur [Anzahl der Gedächtnisblöcke (Anzahl der Gedächtniszellen)]	Durchläufe ARG	Länge des Referenz-traces in 10^3	Position des 10. Fehlers in 10^3	Anteil der erfolgreichen Lernversuche in %
ARG II-3 [8(2)]	100.000	80.194	13.394	20

Die erste Spalte in Tabelle 3-2 zeigt sehr gut, dass mit wachsender Komplexität des Lernproblems die Größe des LSTM-Netzes angepasst werden muss, um einen adäquaten Lernerfolg zu erzielen. So konnten bei der ARG II-1 mit der Anwendung von vier Gedächtnisblöcken gute Ergebnisse erzielt werden, wohingegen bei der ARG II-3 bereits acht Gedächtnisblöcke benötigt wurden. Weiterhin ist festzuhalten, dass nicht jeder Lerndurchgang überhaupt brauchbare Ergebnisse liefert (vgl. Spalte 5, Tabelle 3-2).

Wachsende LSTM

Für die Anwendung von LSTM-Netzen auf das in dieser Arbeit zu lösende Problem müssen diese dem Umstand gerecht werden, ein System mit unbekannter Komplexität abbilden zu können. Wie in Tabelle 3-2 zu sehen ist, muss das LSTM-Netz dem Lernproblem angepasst werden. Da die gegebene Problemstellung ein Unsupervised-Learning darstellt, ist die Komplexität des Lernproblems aber a priori nicht bekannt.

Das Problem der Anpassung der Größe der KNN an die Komplexität des Lernproblems ist nicht neu und wird durch einige Ansätze bereits behandelt. Häufig geht es darum, die richtige Vernetzung zwischen den Neuronen zu finden oder auch die Anzahl der Hidden-Layer zu bestimmen. In der Literatur wird in diesem Zusammenhang von konstruktiven oder wachsenden Algorithmen gesprochen (Kwok, Yeung 1997). Die Vorgehensweise ist meist iterativ. Ausgehend von einfachen KNN-Strukturen werden die Lernergebnisse beobachtet und dann die Topologie des KNN verändert. Ein derartiges Vorgehen ist häufig dann notwendig, wenn entweder das KNN sehr komplex aufgebaut ist und optimiert werden soll oder wenn die Komplexität des Lernproblems stark schwankt und das KNN zur Lernzeit angepasst werden muss.

Die Anwendung von wachsenden LSTM-Netzen, welche sich an variable Problem-stellungen anpassen, wird in (Ribeiro, Alquézar 2002) beschrieben. Allerdings wird in (Ribeiro, Alquézar 2002) die Variante des LSTM genutzt, welche nicht mit sequenzlosen Traces umgehen kann und zudem als Referenzmodell nicht die KERG verwendet. Daher wurde im Rahmen dieser Arbeit die eigene LSTM-Implementierung um den Algorithmus von (Ribeiro, Alquézar 2002) erweitert und zur Evaluierung auf die Referenz der ARG angewendet.

Anwendung auf Problemstellung

Im Wesentlichen wurde das Vorgehen von (Ribeiro, Alquézar 2002) beibehalten. Dort wird vorgeschlagen, die Anzahl der Gedächtnisblöcke um eins zu erhöhen, wenn nach mehreren Epochen der normierte mittlere quadratische Gesamtfehler konstant bleibt. Der neue Gedächtnisblock wird dabei mit allen anderen Gedächtnisblöcken verknüpft und die Werte der bestehenden Gedächtnisblöcke werden eingefroren.

Im Ursprungszustand ist nur ein Gedächtnisblock im Netz vorhanden. Dieser ist mit dem Eingangs-, Ausgangs- und einem Bias-Neuron verbunden und besitzt wie alle neu hinzugefügten Blöcke zwei Gedächtniszellen. Ebenfalls wurden die Eingangs- sowie die Ausgangsneuronen miteinander verbunden. Zum Training wurde aus 1.000 Durchläufen der jeweiligen Grammatik ein Referenztrace erzeugt. Die Gewichtungen wurden nach einem aufgetretenen Fehler in der Vorhersage geändert. Weiter wurde das Training nach einer falschen Vorhersage nicht abgebrochen, sondern weitergeführt, um am Ende einer Epoche den Fehler des Netzes zu berechnen. Verringerte sich der Fehler innerhalb von 1.000 Epochen nicht, so wurde ein neuer Gedächtnisblock in das bestehende Netz eingefügt. Das Training wurde spätestens nach dem Erreichen von 10^5 Epochen abgebrochen und galt dann als Fehlschlag.

Die Ergebnisse der Anwendung der vorgestellten Implementierung wachsender LSTM Netze sind in Tabelle 3-3 dargestellt. Es wurden jeweils 1000 Durchläufe einer Grammatik vorgenommen. Die Anzahl der Epochen gibt an, wie lange es im Schnitt gedauert hat, eine Lösung zu finden, und kann demzufolge als Indikator für den benötigen Rechenaufwand gesehen werden (in den gezeigten Versuchen wird nach 10.000 Epochen mit einem Fehlschlag abgebrochen).

Tabelle 3-3 Ergebnisse aus Versuchen mit wachsenden LSTM-Netzen.

RG Typ	Gedächtnis-Blöcke Anzahl [min., max.] {bisher verwendet}	Epochen	Position des 10. Fehlers in 10^3	Anteil der erfolgreichen Lernversuche %
KERG	2,1 [2, 3] {4}	2.731	69.387	94
ARG I-2	2,5 [2, 4] {4}	4.085	50.093	80
ARG II-2	5,1 [4, 7] {8}	4.940	26.783	75

Es zeigt sich, dass das Verfahren der wachsenden LSTM-Netze zuverlässig die Anzahl der Gedächtnisblöcke bis zu der notwendigen Größe erhöhen kann. Es hat sich bei den Versuchen sogar herausgestellt, dass die in den vorhergehenden Versuchen verwendete Anzahl an Gedächtnisblöcken fast um den Faktor 2 zu groß war. Wurden in den statischen Versuchen 10.000 bis 100.000 Durchläufe der Grammatik als Referenztrace benötigt, so kann nach den hier gewonnenen Erkenntnissen bereits mit 1.000 Durchläufen der Grammatik ein LSTM-Netz trainiert werden, welches das Testtrace entsprechend vorhersagt. Somit kann auch sehr gut gezeigt werden, dass die richtige Dimensionierung des LSTM-Netzes entscheidenden Einfluss auf das Lernergebnis hat.

3.3.2.2 Evaluierung für die Anwendung im selbstlernenden OoN-Erkennungsverfahren

Allgemein ist festzuhalten, dass die Klasse der LSTM-Netze eine sehr vielversprechende Alternative für die Verwendung zur OoN-Erkennung darstellt. Vor allem die sehr gute Erkennung von Langzeitabhängigkeiten ist hervorzuheben. Mit der Einführung von wachsenden LSTM-Netzen kann auch das Verhalten unbekannter Systeme mit

unbekannter Komplexität erlernt werden, ohne dass ein manueller Eingriff in das Lernverfahren möglich ist. Somit ist die prinzipielle Anwendbarkeit auf das Unsupervised-Lernproblem gegeben.

Klassifizierungsgüte

Erkennung tatsächlicher Abweichungen

Die Evaluierung der LSTM-Netze wurde mittels der ERG bzw. der aufbauenden ARG vorgenommen. Diese besitzen, wie in Tabelle 3-4 dargestellt, acht Lösungsklassen. Eine Lösungsklasse beschreibt dabei jeweils eindeutig alle von einem Zustand abgehenden Transitionen. Das LSTM sagt jeweils eine der acht Lösungsklassen voraus. Stimmt die Vorhersage der Lösungsklasse, so kann davon ausgegangen werden, dass der Zustand in der ARG richtig bestimmt wurde. Somit wird das LSTM alle Abweichungen erkennen, welche keinen gültigen Ausführungsgraphen einer ARG darstellen.

Tabelle 3-4 Lösungsklassen einer ARG

Klasse	B	T	P	S	X	V	E
C1	1	0	0	0	0	0	0
C2	0	1	1	0	0	0	0
C3	0	0	0	1	1	0	0
C4	0	1	0	0	0	1	0
C5	0	0	1	0	0	1	0
C6	0	0	0	0	0	0	1
C7	0	1	0	0	0	0	0
C8	0	0	1	0	0	0	0

Fehlalarmrate

Bei der Klassifizierungsgüte von LSTM-Netzen für eine OoN-Erkennung ist vor allem die Fehlalarmrate als kritische Größe zu betrachten. Die Fehlalarmrate nimmt zwar schnell mit länger werdenden Referenztraces ab, allerdings ist die minimal notwendige Länge des Referenztraces im Vergleich zum erzeugenden Automaten immer noch sehr groß. Der Einsatz von wachsenden LSTM kann dies etwas reduzieren, da er die Netzgröße auf das nötige Minimum reduziert.

Langzeitabhängigkeiten

In punkto Erlernung von Langzeitabhängigkeiten stellen LSTM-Netze wohl eines der besten Verfahren auf diesem Gebiet dar. Der tatsächliche maximale Abstand zwischen zwei abhängigen Zeichen lässt sich nicht direkt berechnen und hängt sehr stark von der Datenbasis und dem zugrunde liegenden System ab. Die Versuche haben aber gezeigt, dass davon ausgegangen werden kann, dass LSTM-Netze in der Lage sind, ca. 100 Schritte zwischen abhängigen Zeichen aufzulösen.

Einbeziehung von Wissen aus vorhandener Spezifikation

Wie im unten aufgeführten Evaluierungskriterium Bootstrapping erläutert, ist ein Rückschluss vom erlernten Netz auf das zugrunde liegende System nicht möglich. Daher ist eine nachträgliche gezielte Modifikation des Netzes zur Einbeziehung von weiteren

Wissensquellen nicht möglich. Der einzig mögliche Weg, eine vorhandene Spezifikation einzubeziehen, ist die Erweiterung des vorliegenden Referenztraces. So könnte für den Lernprozess transparent neues Verhalten aufgenommen werden. Die einzig mögliche Variante, dies zu erreichen, wäre ein Eingriff in die Bewertungsfunktion des Lernprozesses. Es ist aber davon auszugehen, dass der wahrscheinlichkeitsbasierte Lernprozess durch das Hinzufügen einzelner Sequenzen, welche nicht im Kontext des Referenztraces liegen, ein nicht mehr nachvollziehbares Verhalten aufweist.

Für die Bewertung der untersuchten LSTM-Netze ist es mit der gezeigten Realisierung nicht möglich, weitere Daten außerhalb des Referenztraces miteinzubeziehen.

Overfitting

Ein generelles Problem bei der Anwendung der vorgestellten LSTM-Netze ist eine notwendige Initialisierungsphase, während der das Netz bei seiner Anwendung erst die richtigen internen Zustände finden muss. Dies resultiert daraus, dass das Netz erst ein paar Nachrichten benötigt, um den internen Status richtig zu setzen. Das Netz muss sich sozusagen erst auf das Trace synchronisieren. Durch die Variierung des Startpunktes im Referenztrace für unterschiedliche Epochen kann diese Eigenschaft etwas abgemildert werden. Trotzdem bleibt eine, wenn auch nicht sehr lange, Initialisierungsphase, in der fälschlicherweise ein Trace abgewiesen werden kann. Daher ist in Tabelle 3-1 die Position des 1. und des 10. Fehlers angegeben, da nach ca. 2 bis 6 Fehlern in der Vorhersage eine entsprechende Synchronisierung beobachtet werden konnte. Die Angabe des 10. Fehlers in Tabelle 3-2 f. gibt daher einen Aufschluss, wie gut die Fehlalarmrate nach der Synchronisierung des Netzes ist.

Diese Eigenschaft ist eine Overfitting-Eigenschaft, da sie nur bei der Bewertung ähnlicher Ströme auftritt und nicht beim Referenztrace.

Bootstrapping

Ein Rückschluss auf das zugrunde liegende System ist bei KNN in aller Regel nicht oder nur sehr schwer möglich. Bestenfalls kann aus der Anzahl der Gedächtnisblöcke eine sehr grobe Abschätzung über die Komplexität des zugrunde liegenden Systems erstellt werden.

Abbruchkriterien für Lernverfahren

Um das KNN anzulernen, wird im Falle der LSTM Netze ein Backpropagation-Verfahren angewendet. Dabei wird der tatsächliche Status der Ausgangsneuronen mit dem erwarteten Status verglichen. Im Falle einer Abweichung werden entsprechend die Gewichte des LSTM-Netzes neu berechnet. Zum Erlernen eines Referenztraces wird das LSTM so lange auf Basis dieses Traces trainiert, bis das LSTM-Netz das Referenztrace vollständig akzeptiert. Ist dies der Fall, war der Lernprozess erfolgreich. Da der Lernprozess nicht deterministisch ist, wird dieser nach Erreichen einer definierten Zeitgrenze abgebrochen.

Das Lernverfahren für LSTM-Netze besitzt dabei keine Erfolgsgarantie und kann selbst für ein und dasselbe Referenztrace teils mit Erfolg und teils mit Misserfolg beendet werden. Dies resultiert aus der zufälligen Initialisierung des LSTM-Netzes, wodurch der

Backpropagation-Algorithmus nicht bei allen Startkonstellationen des LSTM-Netzes zwangsläufig eine richtige Gewichtung findet.

Eine weitere Unschärfe im Lernprozess stellt das Wachstum dar. Für die gegebenen Testdaten, basierend auf der ERG, konnte nach einigen Versuchen die Anzahl der Lernversuche ermittelt werden, nach denen zusätzliche Zellen hinzugefügt werden müssen. Es zeigte sich aber bei weiteren Versuchen, dass bei anderen Testdaten (zufällige Automaten) diese Parameter keine optimalen Ergebnisse gebracht haben. Somit müsste der Lernprozess mehrmals mit unterschiedlichen Parametern für das Wachstum gestartet werden, um diesen auch für unbekannte Daten anwendbar zu machen.

Effizienz und Performance

Performance bzw. Rechenkomplexität des Lernverfahrens

Der benötigte Rechenaufwand für das Trainieren der LSTM Netze ist ausgehend von den durchgeführten Experimenten, verglichen mit den Markov-Modellen signifikant höher, hängt aber sehr stark von der Struktur des zu erlernenden Systems ab. Die Zeitgrenze bis zum Abbruch der Lernverfahren und zur Markierung eines erfolglosen Lernversuches wurde bei der ERG auf 5 Min und bei einer ARGII-3 auf 6 Stunden gesetzt.

Rechenkomplexität in der Anwendungsphase

Für die eigentliche OoN-Erkennung wird nur sehr wenig Rechenaufwand benötigt. Nach der Lernphase hat das Netz eine fixe Größe mit vergleichsweise wenig Zuständen. Der Rechenaufwand während der Anwendungsphase hängt am ehesten von den verwendeten Übergangsfunktionen ab, ist aber nur ein Bruchteil des Lernaufwandes.

3.3.3 Automaten-Lernverfahren – Angluin L^*

Das explizite Lernen von Automaten oder auch Modellen aus dem Verhalten eines Systems ist ein eigener Zweig im Bereich des maschinellen Lernens. Während sich andere Verfahren wie die hier vorgestellten Markov-Modelle (Kapitel 3.3.1) und KNN (Kapitel 3.3.2) damit befassen, das Verhalten eines unbekannten Systems nachzubilden, ohne dabei ergründen zu wollen, wie das System selbst aufgebaut sein könnte, beschäftigen sich Verfahren zum Lernen von Automaten meist damit, die Struktur oder Zusammenhänge des unbekannten Systems aufzuzeigen. Diese Verfahren können für eine OoN-Erkennung durchaus Vorteile bringen, da sie einen Aufschluss über das zugrunde liegende System geben können. Im besten Fall kann solch ein gelernter Automat als Spezifikation für das Netzwerkverhalten angesehen werden. Damit wäre der Nachweis, dass weitere zu bewertende Traces der Spezifikation entsprechen, auch rein formal durchführbar.

Die allermeisten Algorithmen zum Lernen von Automaten basieren auf bestärkenden Lernverfahren (Reinforcement Learning) (Narendra, Thathachar 1974; Kaelbling, Littman, Moore 1996). Welche Strategien dabei verfolgt werden, ist unterschiedlich, allen gemein ist aber, dass die Lernverfahren mit ihrer Umwelt oder dem zu lernenden Objekt interagieren. Der Lernalgorithmus stellt dabei Anfragen oder auch Eingaben an das zu erlernende System und beobachtet anschließend die Reaktion des Systems. Im Gegensatz zum überwachten Lernen (Supervised-Learning) stehen dabei die Eingabe- und Ausgabesequenzen nicht fest. Vielmehr stellt das Lernverfahren Hypothesen über

die Struktur auf und versucht, diese mit gezielten Anfragen an das zu lernende System zu bestätigen. Ein derartiges Vorgehen kann für die Problemstellung der Arbeit allerdings nicht angewendet werden, da eine Interaktion mit dem System nicht möglich ist (Kapitel 2.3.4).

Eine etwas andere Vorgehensweise, die nach (Settles 2009) als aktives Lernen von Automaten bezeichnet werden kann, beschreibt Angluin in (Angluin 1987). Der grundlegende Einsatzzweck des in (Angluin 1987) vorgestellten Verfahrens besteht darin, aus einer Anzahl von Sequenzen, welche das Verhalten eines Systems beschreiben, ein Verhaltensmodell zu extrahieren.

Auf die Ausdrucksweise eines Grammatiklernverfahrens angepasst, erlernt der Algorithmus anhand von Strings oder auch Sätzen eine Sprache und kann dann später entscheiden, ob ein anderer String der vorher erlernten Sprache entspricht (Angluin 1987).

Als aktiv kann das Verfahren deswegen bezeichnet werden, weil ein sogenannter Teacher benötigt wird, welcher ausgewählte Fragen des Lernverfahrens beantwortet. Die weitere Einordnung des Verfahrens in supervised/unsupervised und bestärkendes Lernen hängt davon ab, wie der Teacher implementiert ist. Das Verfahren wird häufig auch als induktiv bezeichnet, da es in seiner Wirkungsweise von einer sehr einfachen Beschreibung des Systems durch gezielte Schlussfolgerung aus den Antworten des Teachers eine immer speziellere Beschreibung des Systems erstellt. Dies ist ein wichtiger Unterschied zu vielen anderen Lernverfahren, welche häufig synthetisierend arbeiten, indem sie die Zustände mehrerer Beobachtungen vergleichen und zusammenführen (Biermann, Feldman 1972; Raman, Patrick, North und Werner 2010).

In (Broy u. a. 2005, Kapitel 19.4) wird ein ausführlicher Vergleich von Angluins Algorithmus mit ähnlichen Algorithmen präsentiert. Das Anwendungsszenario von (Broy u. a. 2005, Kapitel 19.4) ist dabei relativ ähnlich. Es soll aus der Beobachtung eines Black-Box-Systems ein Modell dessen Verhaltens abgeleitet werden. Die Autoren kommen zu dem Schluss, dass für diese Anwendung Angluins Algorithmus am besten geeignet ist. Diese These wird auch in der Promotionsschrift von (Berg 2006) bestätigt.

Der Algorithmus nach (Angluin 1987) scheint für das gegebene Problem gut geeignet zu sein. Angluin selbst hat gezeigt, dass sein Algorithmus, den er L^* nennt, in der Lage ist, in Polynomialzeit den kleinsten diskreten endlichen Automaten zu erlernen, welcher alle im präsentierten gegebenen Beispiele beschreibt.

In (Berg u. a. 2003) wird eine sehr ausführliche Evaluierung des L^*-Algorithmus vorgenommen. Aus den Ergebnissen kann man erste Rückschlüsse über das Verhalten des L^*-Algorithmus auf die hier vorliegende Problemstellung ziehen. Vor allem der Umgang mit komplexeren Zustandsautomaten und mit einem größeren Alphabet, als sie in einer ARG bisher definiert wurden, zeigt eine prinzipielle Vergleichbarkeit mit den Ergebnissen der LSTM-Netze.

Allerdings sind die Experimente nach (Berg u. a. 2003) nicht darauf ausgelegt, einen längeren Netzwerktrace als Eingangsdaten für den Lernalgorithmus zu verwenden, sondern vielmehr wurde eine direkte Interaktion mit dem zu erlernenden System angenommen. Daher geben die Ergebnisse zwar einen guten Anhaltspunkt über die

Performance des L^*-Algorithmus für das hier zu lösende Problem, sind aber für dieses Problem nicht repräsentativ. Zudem hat bei der Anwendung des L^*-Algorithmus die Realisierung des Teachers entscheidenden Einfluss auf die Ergebnisse. Um eine genaue Aussage über die Anwendbarkeit und die Performance des L^*-Algorithmus auf die Problemstellung zu tätigen, wurden eine eigene Implementierung erstellt und eine Evaluierung für die gegebenen Problemstellung durchgeführt. Die entsprechenden Ergebnisse werden im Folgenden präsentiert.

3.3.3.1 Anwendung auf Problemstellung

Funktionsweise und Nomenklatur von Angluin L^*

Um die Anwendung des L^*-Algorithmus besser zu verstehen, werden im Folgenden einige Funktionsmerkmale genauer erklärt und deren spezifische Auswirkungen auf die Problemstellung gezeigt.

Da der L^*-Algorithmus einen endlichen Automaten lernt, kann zunächst die Beschreibung eines Akzeptorautomaten A und eines Traces τ aus Kapitel 2.2 angewendet werden. Dabei kann ein Trace auch als Wort bezeichnet werden, wobei ein Wort aus Zeichen des Alphabetes Σ besteht. Die Menge aller Traces, welche durch ein Alphabet beschrieben werden können, ist dabei $\mathrm{T}, \tau \in \mathrm{T}$.

Neu eingeführt wird an dieser Stelle die Bezeichnung einer Sprache (eng. Language) $\mathcal{L}(A)$. Dabei akzeptiert ein Automat A jeweils eine Sprache $\mathcal{L}(A)$ (Definition 3-5).

Definition einer Sprache $\mathcal{L}(A)$

$$\mathcal{L}(A) = \{\ \tau \in \mathrm{T} \mid der\ Automat\ A\ akzeptiert\ \tau\} \qquad 3\text{-}5$$

Eine Sprache $\mathcal{L}$ wird als regulär bezeichnet, wenn es einen Automaten A (DEA) gibt, der alle Traces einer Sprache akzeptiert.

Der L^*-Algorithmus benötigt für die Interaktion mit seiner Umgebung oder auch zur Interpretation der Daten, aus denen ein Automat generiert werden soll, einen sog. Teacher. Dieser Teacher muss nach Angluin „minimal adequate“ sein und zwei Fragestellungen beantworten:

1. Zugehörigkeitsfrage
   Die Frage beantwortet, ob ein gegebenes Trace (oder Wort) Teil der Sprache sein kann - $\tau \in \mathrm{T}$ ist in $\mathcal{L}(A)$, oder anders ausgedrückt, ob der unbekannte Automat A das Trace τ akzeptieren würde.

2. Äquivalenzfrage
   Diese Frage soll beantworten, ob der von L^* gelernte Automat $\mathbb{A}$ dem unbekannten Automaten A, welcher dem System zugrunde liegt, gleicht. Da A nicht bekannt ist, nennt Angluin diese Frage auch „Conjecure Query“, also ob die Vermutung von L^* über das nicht bekannte System stimmt. Können wie im vorliegenden Fall Automaten nicht direkt miteinander verglichen werden, wird häufig der Vergleich über eine reguläre Sprache benutzt. Somit wird für die Äquivalenzfrage der Vergleich $\mathcal{L}(\mathbb{A}) = \mathcal{L}(A)$ angestellt. Wenn der Vergleich stimmt, wird die Frage mit „ja“ beantwortet. Stimmen die Sprachen nicht überein, so muss ein Gegenbeispiel als Antwort geliefert werden. Dieses

Gegenbeispiel ist entweder ein Wort, welches in $\mathcal{L}(\mathbb{A})$ enthalten ist, aber nicht in $\mathcal{L}(A)$ oder umgekehrt ($\mathcal{L}(\mathbb{A})\backslash\mathcal{L}(A)$ oder $\mathcal{L}(A)\backslash\mathcal{L}(\mathbb{A})$). Anders ausgedrückt ist das Gegenbeispiel entweder ein Trace, welches vom gelernten Automat generiert wird, vom unbekannten Automaten aber nicht akzeptiert wird oder umgekehrt.

Bei der Übertragung dieser beiden Fragen auf die vorliegende Aufgabenstellung ergibt sich das Problem, dass keine einzelnen Sequenzen im Trace identifizierbar sind. Wie bereits bei den LSTM-Netzen muss daher die Anwendbarkeit auf kontinuierlich arbeitende Systeme hergestellt werden.

Anwendbarkeit von Angluin L^*

Einer derartigen Problemstellung wird meist mit der Anwendung von Sampling-Verfahren begegnet (Settles 2011). Dabei werden aus dem Trace nacheinander Sequenzen entnommen und diese dem Lernverfahren als zu lernendes Wort präsentiert. Dabei gibt es verschiedene Herangehensweisen. Eines der einfachsten und häufig eingesetzten Verfahren ist das Random-Sampling. Hierbei wird von einer gleichmäßigen Verteilung oder auch einem statistischen Verhalten des abzubildenden Systems ausgegangen. Aus den Beobachtungen werden dann zufällig Teilsequenzen entnommen und dem Lernverfahren präsentiert. So lässt sich mit der Zeit eine Annäherung an das tatsächliche Systemverhalten erreichen. Die Random-Sampling-Methode funktioniert umso besser, je mehr redundante Informationen vorhanden sind und eine mehr oder weniger zufällige Verteilung der Information vorliegt. Etwas komplexere Verfahren versuchen meist mit Hilfe bisher erzielter Lernergebnisse die Suche bzw. Auswahl neuer Samples zu beschleunigen oder zu verbessern. Diese Verfahren sind aber in der Regel sehr anwendungsspezifisch.

Da Random-Sampling zwangsläufig eine Unschärfe besitzt, wird an dieser Stelle die Möglichkeit durch die Implementierung mittels einer vollständigen Suche vorgestellt.

Zugehörigkeitsfrage

Bei der gegebenen Problemstellung wird davon ausgegangen, dass ein Trace $\tau = e_0 \dots e_i, i \to \infty$ das Verhalten eines Systems beschreibt. Somit wird durch dieses Trace der nicht bekannte Automat A abgebildet. Ist dies so, so enthält dieses Trace alle Worte der Sprache $\mathcal{L}(A)$. Da in dem sequenzlosen Trace keine Wortgrenzen existieren, kann die Suche, ob ein gegebenes Wort im Trace enthalten ist, kein absolut eindeutiges Ergebnis liefern.

Eine Minimalbedingung dafür, dass ein Wort im sequenzlosen Trace enthalten ist, ist, dass die Zeichenfolge des Wortes im Trace auftaucht. Wären die Wortgrenzen bekannt, so könnte entschieden werden, ob es sich tatsächlich um das gesuchte Wort handelt. In diesem Fall müsste vor und nach der Fundstelle eine Wortgrenze existieren. Da eine derartige Prüfung nicht möglich ist, muss sich auf die Minimalbedingung beschränkt werden.

Die Größenordnung der Anzahl der zu stellenden Zugehörigkeitsfragen für den L^*-Algorithmus wird von Angluin auf $\mathcal{O}(n|\Sigma||\mathrm{Z}|^2)$ angegeben. Hierbei gibt n die Länge des Gegenbeispiels an, welches die Äquivalenzfrage zurückliefert. Wird die Länge i des Traces für die Suche nach dem Wort mit einbezogen, so ergibt sich ein Rechenaufwand von $\mathcal{O}(in|\Sigma||\mathrm{Z}|^2)$ pro geliefertem Gegenbeispiel der Äquivalenzfrage (Definition 3-10).

Dabei ist zu beachten, dass die Anzahl der Zustände |Z| des Automaten nicht konstant ist, sondern meist mit der Länge der gelieferten Gegenbeispiele zunimmt.

Anzahl der Zugehörigkeitsfragen nach Angluin $\mathcal{O}(in|\Sigma||Z|^2)$ 3-6

Die Erfahrung bei den Experimenten hat gezeigt, dass die Anzahl der erlernten Zustände in der Regel die Länge des Gegenbeispiels überschreitet, aber in derselben Größenordnung bleibt. Somit kann in erster Näherung $\mathcal{O}(in^3|\Sigma|)$ als Komplexitätsmaß für die Anzahl der Zugehörigkeitsfragen pro Gegenbeispiel angegeben werden[5].

Da die Anzahl der Suchvorgänge im Wesentlichen von der Länge des Gegenbeispiels abhängt, ist eine Realisierung der Zugehörigkeitsfrage mittels einer vollständigen Suche als möglich anzusehen.

Äquivalenzfrage

Der Nachweis der Äquivalenz zweier Sprachen wird gewöhnlich über den Nachweis der Äquivalenz der Automaten, welche die Sprachen beschreiben (Almeida, Moreira, Reis 2009), geführt. Da im vorliegenden Fall nur ein Automat (der von L^* erlernte Automat) bekannt ist, müssen andere Mittel gefunden werden, um diese Frage zu beantworten.

Prinzipiell kann die Äquivalenz zweier Automaten getestet werden, indem geprüft wird, ob ein Automat Worte erzeugen kann, welche durch den zweiten Automat nicht erzeugbar wären und umgekehrt. Gibt es ein solches Wort, so sind die Automaten nicht gleich. Dies entspricht im Wesentlichen dem Äquivalenztest zweier Sprachen (Gleichung 3-7):

Äquivalenz Beschreibung zweier Sprachen $\left(\mathcal{L}(\mathbb{A}) \bigcap \overline{\mathcal{L}(A)}\right) \bigcup \left(\mathcal{L}(A) \bigcap \overline{\mathcal{L}(\mathbb{A})}\right) = \emptyset$ 3-7

Ist davon auszugehen, dass ein Trace alle Worte enthält, welche der nicht bekannte Automat A generieren kann, so gilt es in der Äquivalenzfrage zu prüfen, ob:

a) der gelernte Automat $\mathbb{A}$ Worte generieren kann, welche nicht im Trace enthalten sind und

b) er alle Worte, welche im Trace enthalten sind, auch selbst generieren könnte.

Im Folgenden wird dargestellt, wie die Fragen zu (a) und zu (b) beantwortet werden können und welche Komplexität diese Fragen besitzen können. Die wesentlichen Einflussfaktoren dafür sind (i) die Größe des Alphabetes |Σ|, (ii) die Anzahl der Zustände $|Z|$ des Automaten, (iii) die Gesamtlänge des Traces i und (iv) die maximale Wortlänge n $(1 > n \leq i)$, welche geprüft wird.

Da im vorliegenden Fall keine Wortgrenzen vorhanden sind, kann für die Fragen (a) und (b) keine sinnvolle maximale Wortlänge im Voraus angegeben werden. Die theoretisch

[5] Diese Näherung ist empirisch begründet. Die Anzahl der Zustände hängt in erster Linie vom zugrunde liegenden System ab, dessen Eigenschaften aber nicht bekannt sind.

maximale Wortlänge ist die Länge i des Traces τ, wenn das Trace als ein Wort betrachtet wird. Es ist aber davon auszugehen, dass in einem Trace mehrere Wörter beliebiger Länge enthalten sein können. Somit müssten alle Wörter der Länge 1 bis i, welche ein Trace potentiell abbilden kann, geprüft werden.

Fragestellung (a)

Der naheliegende Ansatz, diese Frage zu beantworten, ist es tatsächlich, alle Worte des gelernten Automaten $\mathbb{A}$ mit der Länge n zu generieren. Die Frage, wie viele verschiedene Worte der gelernte Automat $\mathbb{A}$ generieren kann, hängt sehr stark vom Aufbau des Automaten ab. Wird davon ausgegangen, dass die Zeichen rein stochastisch verteilt sind und ein Automat, wie in Abbildung 3-8 dargestellt geprüft werden soll, so müssen im schlimmsten Fall alle Zeichen des Alphabetes Σ in allen Variationen einschließlich der Wiederholung für ein Wort der Länge n geprüft werden. Geprüft wird, indem der gelernte Automat das Wort generiert und für dieses Wort die Zugehörigkeitsfrage gestellt wird. Die Anzahl der möglichen Variationen mit Wiederholung ergibt sich aus Gleichung 3-8:

$$\left(\binom{n}{|\Sigma|}\right) = n^{|\Sigma|} \qquad 3\text{-}8$$

Somit ergibt sich bei der Suche der möglichen Variationen im Trace mit der Länge i, die Rechenkomplexität $\mathcal{O}(in^{|\Sigma|})$ als obere Grenze (Gleichung 3-9 f.).

Maximale Rechenkomplexität für Äquivalenzfrage Teil (a) $\mathcal{O}(in^{|\Sigma|})$ 3-9

Minimale Rechenkomplexität für Äquivalenzfrage Teil (a) $\mathcal{O}(i|Z|)$ 3-10

An dieser Stelle wird zur Bestimmung auch die Struktur der Automaten herangezogen. So ist der am meisten spezialisierte Automat für einen Trace, bestehend aus den Zeichen ABCD, der Automat in Abbildung 3-7, da er genau eine Kombination, nämlich ABCD in zyklischer Wiederholung erzeugen kann. Der Automat in Abbildung 3-8 hingegen kann als der allgemeinste Automat angesehen werden, da er jede beliebige Kombination der Zeichen ABCD in jeder beliebigen Länge erzeugen kann.

Sind in den vorliegenden Traces Abhängigkeiten zwischen den Zeichen vorhanden, so ergibt sich eine etwas andere Konstellation für Frage (a). Soll z. B. die Frage (a) für den Automaten aus Abbildung 3-7 beantwortet werden, so ergibt sich die maximale Anzahl der erzeugbaren Wörter aus der Anzahl der Zustände $|Z|$. Hierbei hat die Länge der Wörter keinen Einfluss, da jeweils nur der Zeichenstrom „ABCD…" mit unterschiedlichen Präfixen erzeugt werden kann. Ein derartiger Automat kann auch als der speziellste Automat angesehen werden, da er keine Wortkombinationen zulässt. Somit ergibt sich für den speziellsten Automaten eine Rechenkomplexität von $\mathcal{O}(i|Z|)$ zur Beantwortung der Frage (a).

Die Frage (a) wird mit einem der Wörter, welches der gelernte Automat $\mathbb{A}$ generieren kann, welches aber nicht im Trace enthalten ist, beantwortet. Aufgrund der Tatsache,

dass dieses Wort nicht im Trace enthalten ist, wird dieses Wort im Folgenden negatives Gegenbeispiel genannt.

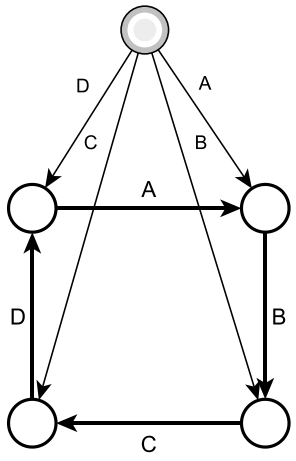

Abbildung 3-7 Automat mit einer Transition pro Zustand (ohne Startzustand)

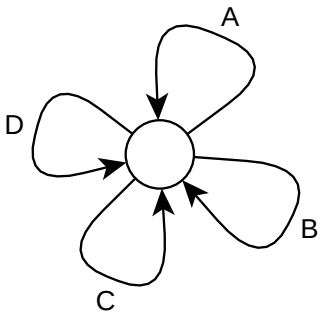

Abbildung 3-8 Automat mit einem Zustand und der maximalen Komplexität

Fragestellung (b)

Die Komplexität der Frage (b) ist auf den ersten Blick unabhängig vom gelernten Automaten, da nicht die Wörter vom gelernten Automaten generiert, sondern im Trace gesucht werden. Die Frage, ob der gelernte Automat das Wort selbst generieren kann, kann auch umgedeutet werden in die Frage, ob der Automat das Wort akzeptiert. Diese Umdeutung ist zulässig, wenn die Ausgabefunktion in eine Eingabefunktion umgewandelt wird. Dazu müssen lediglich die Zeichen, welche zur Transition gehören, als Übergangsbedingung für diese Transition verwendet werden. Somit kann das Ausgabealphabet gleich dem Eingabealphabet des Automaten gesetzt werden. Die Prüfung der Frage (b) wird durch die Anwendung der Akzeptanzfrage stark vereinfacht. Es müssen nicht alle erzeugbaren Wörter des erlernten Automaten bekannt sein und durchsucht werden. Die Prüfung der Akzeptanz des Wortes durch den erlernten Automaten ist von der Komplexität $\mathcal{O}(\mathrm{n})$ und eine hinreichende Bedingung für die Beantwortung der Frage (b). Um die Komplexität der Frage (b) zu zeigen, muss bekannt sein, wie viele Wörter ein Trace beinhalten kann. Da auch für diese Frage keine Wortgrenzen im Trace bekannt sind, müssen alle möglichen Wörter der Länge n aus

dem Trace der Länge i geprüft werden. Somit ergibt sich die obere Grenze der zu prüfenden Wörter nach Gleichung 3-11.

$$G = \sum_{k=i-n+1}^{i} k = \frac{1}{2}\,(2in - n^2 + n) \qquad 3\text{-}11$$

Wird bis zur maximalen Wortlänge $n = i$ geprüft, so ergibt sich Gleichung 3-12 als Obergrenze der zu prüfenden Worte.

$$G = \left(\frac{1}{2}\,(i^2 + i)\right) \qquad 3\text{-}12$$

Wird weiterhin beachtet, dass jedes Wort im Trace mit der Länge i gesucht werden muss, so ergibt sich die maximale Rechenkomplexität für Frage (b) nach Gleichung 3-13

Maximale Rechenkomplexität für Äquivalenzfrage, Teil (b) $\mathcal{O}(i^2 n)$ 3-13

Ist der zugrunde liegende Automat bekannt, so ist es möglich, redundante Prüfungen zu unterlassen. Wiederum stellt ein zugrunde liegender Automat wie in Abbildung 3-7 den einfachsten Fall dar. Hier ist es nur notwendig, dass dieser Automat ein Wort der Länge $n = i$ (also das gesamte Trace) akzeptiert. Dies ist ausreichend, da aufgrund der Struktur des Automaten nachgewiesen werden kann, dass in diesem Fall alle Worte $n < i$ ebenfalls akzeptiert werden müssen. Somit ergibt sich $\mathcal{O}(i)$ als mögliche Untergrenze für die Frage (b) (Gleichung 3-14).

Minimale Rechenkomplexität für Äquivalenzfrage Teil (b) $\mathcal{O}(i)$ 3-14

Wird im Trace ein Wort gefunden, welches nicht vom erlernten Automaten generierbar ist, so wird dieses Wort als Gegenbeispiel zurückgeliefert. Dieses Gegenbeispiel wird im Folgenden positives Gegenbeispiel genannt.

Liefern weder Frage (a) noch Frage (b) Gegenbeispiele, so kann die Hypothese L^* als richtig angesehen werden. Der erlernte Automat $\mathbb{A}$ stimmt also mit dem zugrunde liegenden Automaten A überein.

Die angegebenen Grenzen für die Komplexität der Überprüfung der Äquivalenzfrage beziehen sich auf die lineare Suche nach Gegenbeispielen, bis alle Alternativen durchsucht wurden.

Implementierung des Lernverfahrens nach Angluin L^*

Im vorhergehenden Kapitel wurde die prinzipielle Anwendbarkeit von L^* auf die Problemstellung und die zu erwartende Komplexität aufgezeigt. Hierdurch lässt sich bereits ein Polynomialzeitverhalten des Verfahrens gegenüber der Wortlänge n zeigen. Weiterhin kann schon vorhergesagt werden, dass die Komplexität des Lernvorganges abnehmen wird, je regelmäßiger bzw. abhängiger die zugrunde liegenden Daten sind.

In diesem Kapitel werden anhand einer eigenen Implementierung die Anwendbarkeit und die Eigenschaften des L^* für die vorliegende Problemstellung evaluiert. Dazu werden zunächst die Realisierung der Zugehörigkeitsfrage, der Äquivalenzfrage und einige Details der Algorithmus diskutiert. Anschließend werden Ergebnisse bezüglich der Lern- und Klassifizierungsgüte vorgestellt.

Realisierungsdetails L^*

Als Datenformat für die Beschreibung von Wörtern und Traces bieten sich für diesen Anwendungsfall sog. Strings an. Ein String ist in diesem Fall eine Zeichenkette einer definierten Länge. So wird ein Trace durch einen String mit i Zeichen repräsentiert. Ein Zeichen entspricht dabei bei der ID eines Events im Trace. Die Suche nach Wörtern im Trace entspricht in diesem Fall der Suche nach einen Teilstring. Damit lässt sich sehr einfach die Zugehörigkeitsfrage realisieren.

Arbeitsweise

Die prinzipielle Arbeitsweise des Angluin L^*-Algorithmus ist als Pseudocode in Abbildung 3-9 dargestellt. Zunächst wird stets eine initiale Hypothese über das System aufgestellt. Diese entspricht immer dem allgemeinsten Automaten (ähnlich Abbildung 3-8) mit einem Zustand und gilt für jedes vorkommende Zeichen einer Transition. Es werden dann jeweils über die Äquivalenzfrage Gegenbeispiele in die Hypothese aufgenommen. Dieser Vorgang wird so lange wiederholt, bis keine Gegenbeispiele die aufgestellte Hypothese widerlegen können.

Die Hypothese des L^*-Algorithmus wird nicht auf Basis von Automaten repräsentiert. Vielmehr wird eine Beobachtungstabelle erstellt, welche alle möglichen Zustände und Transitionskombinationen enthält. Die Beobachtungstabelle ist vorstellbar als eine Tabelle mit Präfixen und Suffixen. Die Präfixe indexieren die Reihen und die Suffixe die Spalten. Die Zusammensetzung aus Präfix und Suffix ergibt jeweils eine Abfolge von Zeichen. Diese Abfolge kann je nach Eintrag in der Tabelle gültig (wahr) oder nicht gültig (falsch) sein. Die Einträge in der Tabelle können dabei $falsch$ oder $wahr$ (äquivalent zu wahr/falsch) sein. Ist ein Eintrag $falsch$, so ist der String aus Präfix und Suffix nicht Teil des Automaten. Ist ein Eintrag $wahr$, so kann er vom Automaten beschrieben werden. Diese Werte werden mittels der Zugehörigkeitsfrage ermittelt. Weiterhin wird die Tabelle horizontal in zwei Teile unterteilt. Der obere Teil identifiziert potenzielle Zustände des Automaten und der untere Teil potenzielle Transitionen des Automaten. Wird von der Äquivalenzfrage ein Gegenbeispiel geliefert, so werden damit neue Zustände im oberen Bereich der Tabelle eingeführt.

Diese Tabelle muss nach (Angluin 1987) geschlossen und konsistent sein. Ist diese geschlossen und konsistent, so kann daraus ein Automat erzeugt werden.

Eine Tabelle ist konsistent, wenn alle Zeilen der Tabelle, welche einen identischen Zustand beschreiben, diesen Zustand auch jeweils gleich als vorhanden bzw. nicht vorhanden beschreiben. Wird ein Zustand unterschiedlich bewertet (inkonsistente Tabelle), wird eine neue Spalte eingeführt, welche die beschriebenen Zustände der Tabelle entsprechend konkretisiert.

Eine Tabelle ist geschlossen, wenn jede Transition zu einem in der Tabelle beschriebenen gültigen Zustand führt. Ist dies nicht der Fall, wird ein neuer Zustand in die Tabelle

eingeführt. Dieser Zustand setzt sich aus der Sequenz bis zur Transition und der Sequenz aus der Transition ohne Zielzustand zusammen.

```
Erstelle initiale Beobachtungstabelle (initiale Hypothese für
Automaten)
Wiederhole solange Äquivalenzfrage ein Gegenbeispiel liefert
{
      Füge Gegenbeispiel in Beobachtungstabelle ein (Erweiterung der
      Beobachtungstabelle durch neuen Zustand)
      Wiederhole bis Beobachtungstabelle konsistent und geschlossen
      {
            Prüfe, ob Beobachtungstabelle geschlossen (alle
            Transitionen eindeutig)
            {
                  Füge neue Reihe bei den Zuständen ein bis
                  Beobachtungstabelle geschlossen
                  Stelle Zugehörigkeitsfrage für neu eingefügte Reihen
                  der Beobachtungstabelle
            }
            Prüfe, ob Beobachtungstabelle konsistent (alle Zustände
            eindeutig)
            {
                  Füge neue Spalte ein bis Beobachtungstabelle
                  konsistent
                  Stelle Zugehörigkeitsfrage für alle Reihen der
                  Beobachtungstabelle
            }
      }
      Stelle Äquivalenzfrage
}
Erstelle Automat aus Beobachtungstabelle
Der erlernte Automat ist äquivalent dem tatsächlichen Automaten
```

Abbildung 3-9 Arbeitsweise Angluin L^*-Algorithmus (Pseudocode)

Eine beispielhafte Anwendung des Angluin L^*-Algorithmus inklusive der Beschreibung der Manipulation der Beobachtungstabellen findet sich in Anhang A.

Realisierung der Äquivalenzfrage

Sehr wichtig für das schnelle Erreichen einer guten Abbildung des gelernten Automaten aus dem Trace sind die präsentierten Gegenbeispiele. Angluin selbst gibt hierbei den Hinweis, dass es sehr hilfreich für L^* ist, wenn jeweils das kürzeste Gegenbeispiel präsentiert wird, welches die Hypothese von L^* widerlegt.

Im Folgenden werden die Realisierung der Fragestellung (a) und (b) der Äquivalenzfrage näher erläutert und deren Bedeutung für die Ergebnisse von L^* aufgezeigt.

Fragestellung (a)

Um ein negatives Gegenbeispiel zu finden, muss versucht werden möglichst alle vom erlernten Automaten generierbaren Worte gegenüber dem Referenztrace zu prüfen. Um die Forderung nach dem kürzesten Gegenbeispiel zu erfüllen, wird die Wortlänge von $n = 2$ beginnend schrittweise erhöht. Die Wortlänge wird erst dann erhöht, wenn mit der gegebenen Wortlänge keine Gegenbeispiele im Referenztrace gefunden werden können. Somit wird garantiert, dass immer die kürzesten Gegenbeispiele zuerst gefunden werden. Das angewendete Verfahren entspricht damit einer Breitensuche auf dem Ausführungsbaum des erlernten Automaten.

Optimierung

Wie weiter oben in diesem Kapitel gezeigt, hängt die Anzahl der zu untersuchenden Wörter von der Struktur des erlernten Automaten ab. Vor allem Transitionen von Zuständen auf sich selbst erhöhen die Anzahl der zu suchenden Worte sehr stark. Daher wird bei der Ausführung des erlernten Automaten der Durchführungszweig gestoppt, wenn ein Zustand mehr als zweimal durchlaufen worden ist. Durch diese Maßnahme wird zum einen die zu prüfende Anzahl an Worten stark reduziert und zum anderen eine Überspezialisierung des erlernten Automaten verhindert. Da die hier verwendeten Automaten keine Begrenzung von Schleifen kennen, würden sonst aufeinanderfolgende gleiche Zeichen als Reihe in den zu erlernenden Automaten einfließen und die allgemeinere Form der unbegrenzten Schleife ablösen.

Fragestellung (b)

Die Realisierung für das Finden von positiven Gegenbeispielen wird bei der vollständigen Suche mit $\mathcal{O}(i^2 n)$ für große i ($i \sim 10^3 - 10^5$) zu umfangreich und somit nicht mehr durchführbar. Daher wird für diese Suche angenommen, dass es ausreichend ist, wenn der gelernte Automat das Referenztrace als komplettes Wort akzeptiert. Diese Annahme ist hinreichend für Automaten der Form nach Abbildung 3-7 und reduziert die Komplexität des Problems auf $\mathcal{O}(i)$. Dies kann in abgeschwächter Form auch für Automaten angenommen werden, welche nicht, wie in Abbildung 3-7 dargestellt, aufgebaut sind, wenn ein entsprechend längeres Referenztrace gegeben ist. Diese Annahme hat sich bei den durchgeführten Experimenten, welche weiter unten in diesem Kapitel vorgestellt werden, bestätigt, so dass für die Suche nach positiven Gegenbeispielen auf das Sampling von einzelnen Wörtern verzichtet werden kann.

Eine Minimalanforderung an den gelernten Automaten $\mathbb{A}$ ist, dass er das vorliegende Referenztrace vollständig akzeptiert. Wird das Referenztrace nicht akzeptiert, so ist ein positives Gegenbeispiel gefunden. Nach der gegebenen Definition beginnt das Gegenbeispiel bei Position $i = 0$ und endet mit der Position $i = n$ im Trace, welches das Zeichen enthält, das vom Automaten nicht akzeptiert wurde. Um die Forderung nach dem kürzesten Gegenbeispiel zu gewährleisten, wird von Position $i = n$ ein Teiltrace $\tau = e_{n-x} \dots e_n$ generiert und geprüft, ob dieses vom erlernten Automaten akzeptiert wird. Dabei wird x so lange erhöht, bis das Teiltrace nicht mehr akzeptiert wird, womit x die Länge des Gegenbeispiels darstellt. Mit diesem Vorgehen werden die Teile vom Referenztrace entfernt, welche quasi redundant vorhanden sind. Somit werden frühere

Schleifendurchläufe des erlernten Automaten entfernt, welche keine Auswirkung auf spätere Zustände haben.

Auswirkungen von Gegenbeispielen

Auswirkung negativer Gegenbeispiele.

Negative Gegenbeispiele sind dafür verantwortlich, dass der gelernte Automat $\mathbb{A}$ keine Sequenzen/Wörter generieren kann, welche nicht im Referenztrace enthalten sind. L^* wird mit dem allgemeinsten Automaten mit einem Zustand und einer Transition pro Zeichen des Alphabetes initialisiert (Abbildung 3-8). Dieser Automat kann jede beliebige Zeichenfolge generieren. Negative Gegenbeispiele schränken die Variationsmöglichkeiten des erlernten Automaten für die Generierung von Wörtern ein. In diesem Zusammenhang hat sich herausgestellt, dass die maximal geprüfte Länge n der generierbaren Wörter eine wichtige Größe für viele Eigenschaften des gelernten Automaten $\mathbb{A}$, aber auch für die Laufzeit von L^* darstellt.

Der für das Erlernen von Langzeitabhängigkeiten bestimmende Faktor ist die getestete Wortlänge n. Je größer die getestete Wortlänge n ist, umso mehr „Allgemeinheit" verliert der gelernte Automat $\mathbb{A}$. Der Einfluss von n ist sehr gut bei dem Erlernen einer ERG zu beobachten, wie in Abbildung 3-10 gezeigt wird. Erst wenn die getestete Wortlänge über die Anzahl der Transitionen zwischen den abhängigen Punkten im zugrunde liegenden Referenzautomaten hinausgeht, kann L^* die Abhängigkeiten zwischen den rot markierten Transitionen auch auflösen. Die schrittweise Erhöhung der Wortlänge n für Gegenbeispiele ist dennoch beizubehalten, da sonst die Forderung der minimalen Länge des Gegenbeispiels sehr häufig verletzt wird.

Mit der Präsentation von negativen Gegenbeispielen werden sukzessive Zeichenkombinationen im gelernten Automat $\mathbb{A}$ ausgeschlossen. Somit verbessert sich die Klassifizierungsgüte des erlernten Automaten im Sinne der OoN-Erkennung. Genauer verbessert sich die Richtig-Negativ-Rate, also die Anzahl der vom erlernten Automaten erkennbaren Abweichungen vom Normalverhalten.

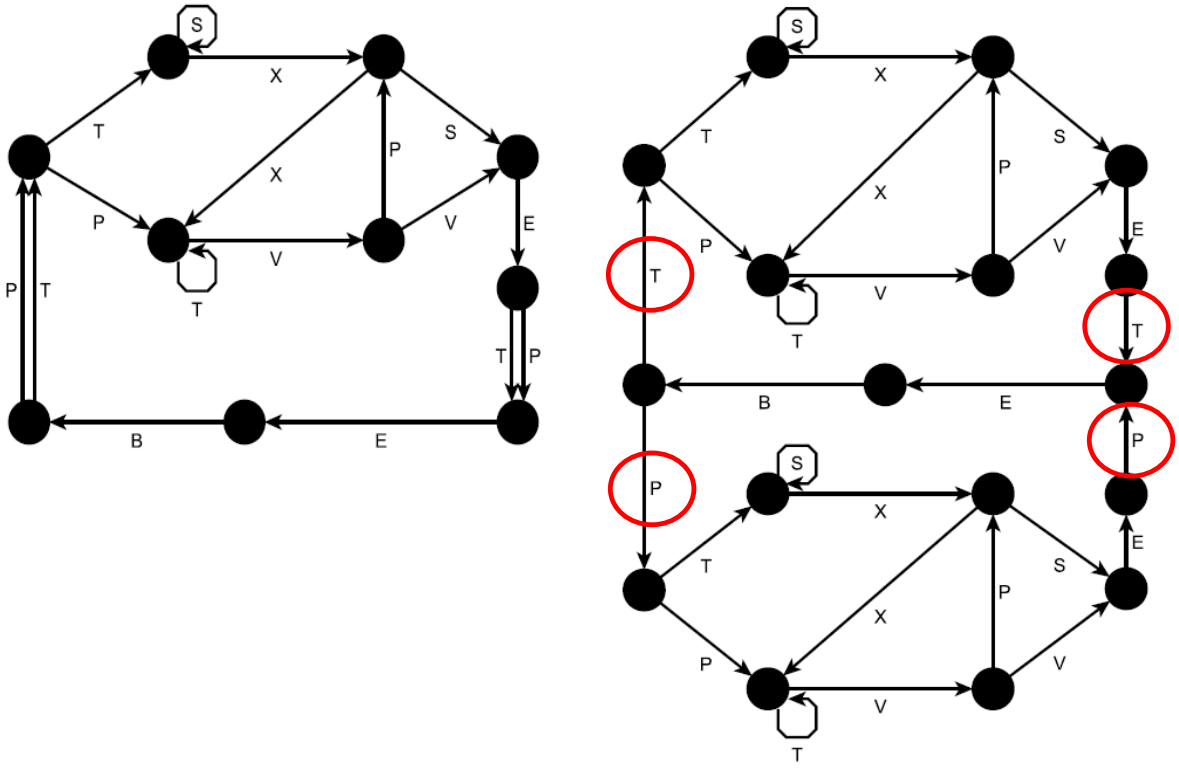

Abbildung 3-10 Lernergebnisse einer KERG
Links: gelernter Automat, welcher mit einer Wortlänge von n=6 getestet wurde, rechts: gelernter Automat, welcher mit einer Wortlänge von n=10 getestet wurde.

Auswirkung positiver Gegenbeispiele

Positive Gegenbeispiele stellen Verhalten im Referenztrace dar, welches der gelernte Automat nicht akzeptiert, aber akzeptieren sollte. Durch positive Gegenbeispiele werden dem gelernten Automaten in der Regel neue Transitionen hinzugefügt, so dass er an einem bestimmten Zustand ein weiteres Zeichen akzeptiert. Positive Gegenbeispiele verbessern vor allem die Fehlalarmrate (False-Positive-Rate) des gelernten Automaten $\mathbb{A}$ im Sinne der OoN-Erkennung.

Wie oben bei der Äquivalenzfrage beschrieben, hat es sich gezeigt, dass es nicht zwangsläufig notwendig ist, alle Wörter des Referenztraces auf ihre Akzeptanz vom gelernten Automaten zu testen. Positive Gegenbeispiele werden mittels des Durchlaufs des Referenztraces durch den gelernten Automaten gefunden. Dabei wird geprüft, ob das Referenztrace vom gelernten Automaten akzeptiert wird. Wird an einer Stelle das Referenztrace abgewiesen, so wird diese Stelle als positives Gegenbeispiel dem Lernverfahren präsentiert. Weiterhin konnte gezeigt werden, dass der Verzicht auf das Sampling aller Wörter aus dem Referenztrace keine signifikant schlechteren Ergebnisse liefert.

Evaluierung mittels Reber-Grammatik

Die im Kapitel 3.2.1 beschriebenen Referenzmodelle KRG, KERG und ARG, deren Nutzung bereits bei der Untersuchung KNN in Kapitel 3.3.2 vorgestellt wurde, werden

zur Evaluierung der Lernverfahren genutzt. Ein wichtiger Fokus liegt dabei auf der Evaluierung der Abbildung von Langzeitabhängigkeiten. In diesem Kapitel werden die Ergebnisse der Anwendung der in dieser Arbeit präsentierten Realisierung des L^*-Algorithmus mittels der Referenzmodelle KRG, KERG und ARG vorgestellt.

Getestet wurde, mit welcher Länge des Referenztraces die vorgestellte Realisierung des L^*-Algorithmus die jeweilige Reber-Grammatik erlernen kann. Hierzu wurden Tracelängen von 10^2 bis 10^8 Zeichen verwendet[6]. Wurde eine Reber-Grammatik korrekt erlernt, so wurde der Lernprozess abgebrochen. Die Durchführungszeit für einen Lernvorgang wurde auf 24 Stunden begrenzt. Wird nach dieser Zeit nicht der korrekte bzw. vollständige Automat gelernt, so werden die maximal getestete Wortlänge und der mit dieser Wortlänge erlernte Automat beschrieben. In diesem Fall wird der erlernte Automat alle Ausgaben der zu erlernenden Reber-Grammatik akzeptieren, aber nicht alle Abhängigkeiten bzw. Langzeitabhängigkeiten aufgelöst haben. Das bedeutet, dass der gelernte Automat zwar das Referenztrace akzeptiert, aber gleichzeitig auch Wörter generieren kann, welche nicht im Referenztrace enthalten sind. Aufgrund der rekursiven Einbettung der Reber-Grammatik ergibt sich die Anzahl der aufgelösten Langzeitabhängigkeiten aus der Anzahl der Reber-Grammatiken im erlernten Automaten $\mathbb{A}$. Der Abstand der Langzeitabhängigkeiten ist bei den ARG-Grammatiken größer als bei den KERG-Grammatiken.

Die Ergebnisse der durchgeführten Evaluierung sind in Tabelle 3-5 zu sehen. Da der Referenztrace zufällig erzeugt wurde, sind alle Versuche mit mehreren Referenztraces durchgeführt worden. Die angegeben Werte geben den Durchschnitt der Versuche wieder. Abweichungen ergeben sich fast ausschließlich in der Länge des benötigten Referenztraces bzw. der maximal getesteten Wortlänge.

[6] Der Nachweis, dass korrekt erlernt wird, wird in diesem Fall über den direkten Vergleich der erlernten Automaten mit dem zugrunde liegenden Referenzautomaten des Referenztraces geführt.

Tabelle 3-5 Ergebnisse der Evaluierung L^*-Algorithmus mittels Reber-Grammatiken

Grammatik Typ	Anz.-RG	Zustände	gelernt	Länge des Referenztraces	Lernzeit	Getestete Wortlänge
RG	1	7	x	1.000	<1 s	4
KERG1	2	19	x	10.000	20 s	7
KERG2	4	43	x	100.000	1 min	11
KERG3_N	6	19	x	100.000	2 min	4
KERG3	8	91		100.000.000	24 h	6
ARG	3	26	x	10.000	20 s	12
ARGII-1_N	3	28	x	10.000	1 min	11
ARGII-1	9	82		100.000.000	24 h	11
ARGII-2_N	3	30	x	10.000.000	90 min	7
ARGII-2	27	250		100.000.000	24 h	10
ARGII-3_N	1	16	x	1.000.000	1 min	5
ARGII-3	81	754		100.000.000	24 h	6

Im Ergebnis besitzen erlernte Automaten einer vorgegebenen Reber-Grammatik mehr Zustände als die zu zugrunde liegende Grammatik. Wie in Abbildung 3-11 zu sehen ist, besitzt der erlernte Automat einen Startzustand (links oben), von dem jeweils alle Zustände der zugrunde liegenden Reber-Grammatik erreicht werden. Die nicht ausgefüllten Zustände in Abbildung 3-11 besitzen nur abgehende Transitionen, so dass diese vom Startzustand aus maximal einmal durchlaufen werden können. Die ausgefüllten Zustände sind die zur Reber-Grammatik gehörenden Zustände. Die nicht ausgefüllten Zustände können auch als Initialisierungszustände bezeichnet werden, welche den Eintritt in die bzw. die Synchronisierung mit der Reber-Grammatik von jedem beliebigen Punkt im Trace erlauben.

Diese Eigenschaft wirkt sich sehr positiv auf den Einsatz für die OoN-Erkennung aus. Der erlernte Automat kann von jedem beliebigen Ausgangspunkt ein zu prüfendendes Trace beurteilen. Dadurch wird eine Initialisierungsphase überflüssig.

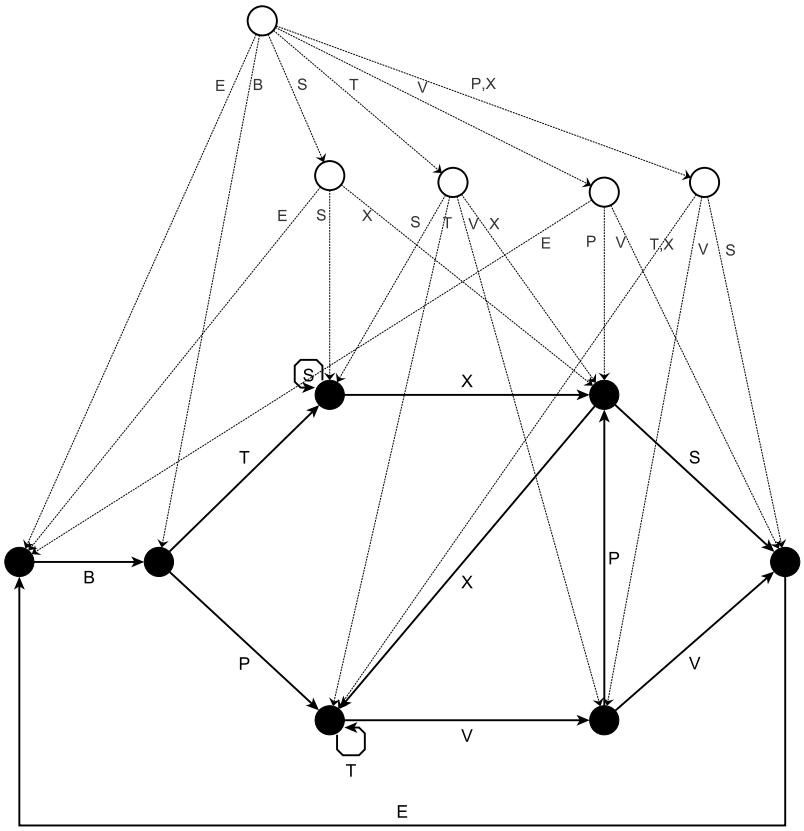

Abbildung 3-11 Mit L^* erlernter Automat, ausgehend von einem mittels einer RG erzeugten Referenztrace

Bei der Auswertung der Lernergebnisse kann Folgendes konstatiert werden:

Gültige Lösung. Die Evaluierung hat gezeigt, dass die hier vorgestellte Realisierung des L^*-Algorithmus für alle Grammatiken eine gültige Lösung findet. Eine gültige Lösung ist in diesem Fall ein Automat, der alle Ausgaben der verwendeten Referenzgrammatik akzeptiert und bis zur getesteten Wortlänge keine Ausgaben erzeugen kann, welche nicht von der Referenzgrammatik akzeptiert würden.

Vollständige Lösung. Eine vollständige Lösung wird genau dann erzielt, wenn exakt der gleiche Automat wie die Referenzgrammatik erlernt wurde. Nach den vorliegenden Versuchen konnten für die RG, die KERG1, die KERG2 und die ARG eine vollständige Lösung des Lernproblems erzielt werden. Für die KERG3, die ARGII-1, die ARGII-2 und die ARGII-3 konnten zwar gültige, aber keine vollständigen Lösungen erreicht werden. Die jeweiligen gültigen Lösungen sind in Tabelle 3-5 mit XXX_N gekennzeichnet. Die maximale Anzahl der RG beträgt 4 für die KERG- und 3 für die ARG-Typen der Reber-Grammatik. Die maximal aufgelöste Anzahl an Zuständen beträgt 43 bei der KERG2.

Lerndauer. Die Lerndauer zur Erstellung gültiger Lösungen ist in allen Fällen kleiner als 60 Sekunden. Da in Tabelle 3-5 immer die gültige Lösung mit der größten getesteten Wortlänge enthalten ist, weicht dies für die ARGII-2_N mit 90 Sekunden und n=7 ab. Die ersten gültigen Lösungen konnten bei der ARGII-2_N aber mit n=6 in unter 60 Sekunden erstellt werden.

Langzeitabhängigkeiten. Die Langzeitabhängigkeiten konnten für die RG, die KERG1, die KERG2 und die ARG-I vollständig aufgelöst werden. D. h. im Falle der KERG2 konnte ein Abstand von minimal neun Zeichen zwischen zwei abhängigen Transitionen aufgelöst werden (n=11, da Start- und Endzeichen der Sequenz erkannt werden müssen).

Effizienz der Lösungsfindung. Die Evaluierung wurde mit zufällig erzeugten Referenztraces durchgeführt. D. h. für größere Grammatiken sind zwangsläufig längere Referenztraces notwendig, um sicherzustellen, dass alle möglichen Ausführungsgraphen im Referenztrace enthalten sind. Daher wird für größere Grammatiken zwangsläufig ein längeres Trace benötigt. Es ist festzustellen, dass die benötigte Länge der Referenztraces bis zur ARGI, welche vollständig gelernt werden konnte, relativ linear verläuft (10^4 Events). Für komplexere Grammatiken ist ein signifikanter Anstieg der benötigten Länge des Referenztraces zu beobachten (> 10^6 Events). Bis zur ARGI bzw. zur KERG2 skaliert der Algorithmus in der benötigten Länge des Referenztraces, also auch in der benötigten Rechenzeit, sehr gut. Dies korreliert mit der maximal getesteten Wortlänge. So kann für einfache Grammatiken diese Wortlänge sehr schnell erhöht werden, wobei bei komplexeren Grammatiken auch nach längerer Lernzeit eine Wortlänge von sechs nicht überschritten wird.

Evaluierung mittels zufällig generierter Automaten

Neben der Evaluierung des L^*-Algorithmus auf Basis von Tests mit Reber-Grammatiken werden hier Ergebnisse aus der Evaluierung mittels zufällig erzeugter Referenzautomaten vorgestellt.

Durch die Nutzung zufällig erzeugter Referenzautomaten soll das Verhalten des L^*-Algorithmus im Hinblick auf sein Verhalten bei unterschiedlich komplexen Referenzautomaten (vgl. Definition der Komplexität eines Automaten in Kapitel 4.2.1) und unterschiedlichen Alphabetgrößen untersucht werden. Das Vorgehen ähnelt dabei der Evaluierung mittels Reber-Grammatiken, nur mit dem Unterschied, dass die Grammatik bzw. der zugrunde liegende Referenzautomat zufällig geniert wurden. Bei der Generierung der Referenzautomaten wurden die Alphabetgröße, die Anzahl der Zustände und die Anzahl der Transitionen variiert (Kapitel 3.2.1).

Abschätzung der Effizienz und maximaler Komplexität

Um eine Abschätzung geben zu können, wie effizient der L^*-Algorithmus aus einem Referenztrace die Abhängigkeiten extrahieren kann, wurden die Länge der Referenztrace für jeweils unterschiedlich komplexe Referenzautomaten variiert.

In Abbildung 3-12 ist dargestellt, wie lang ein Referenztrace sein müsste, um mittels des L^*-Algorithmus den jeweils korrekten Automaten lernen zu können. Es ist gut zu sehen, dass mit steigender Anzahl an Zuständen die Länge des Referenztraces jeweils vergrößert werden muss, um den korrekten Referenzautomaten erlernen zu können.

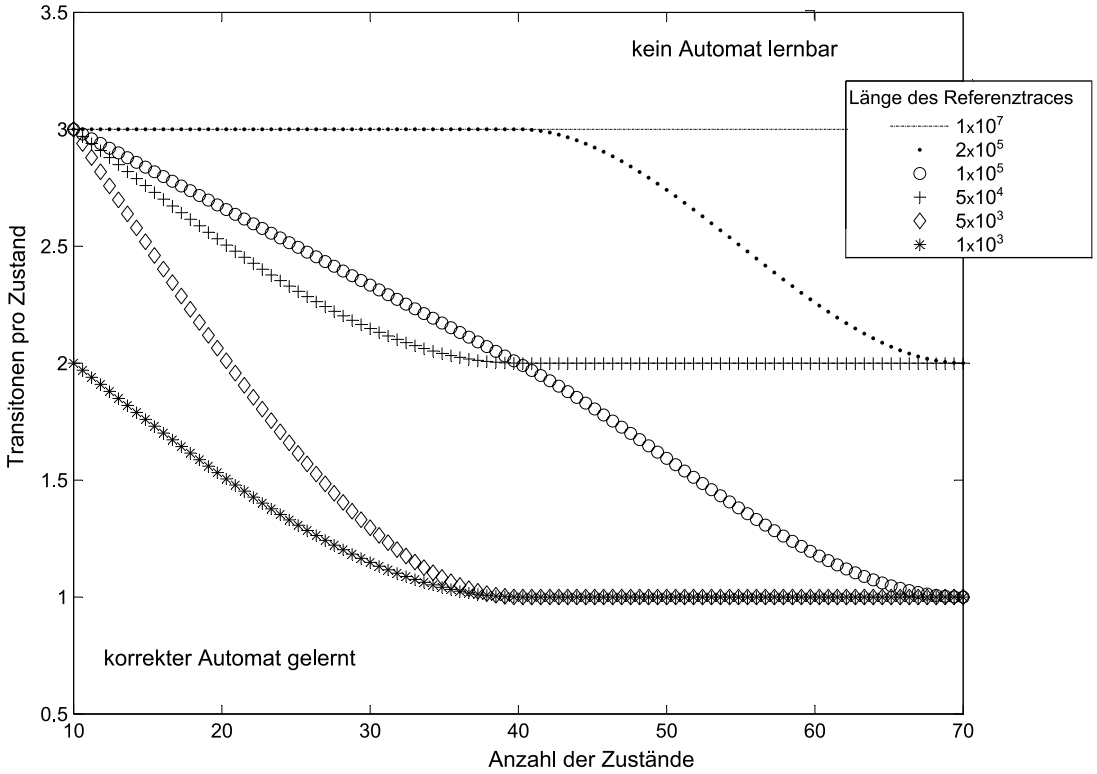

Abbildung 3-12 Evaluierung des L^*-Algorithmus mittels zufällig erzeugter Automaten

Es hat sich dabei gezeigt, dass schon mit einer kleinen Tracelänge von ca. 10^3 Zeichen Referenzautomaten mit bis zu 20 Zuständen und mehr als einer Transition pro Zustand erlernbar sind. Die notwendige Länge des Referenztraces zum korrekten Erlernen des Referenzautomaten nimmt vor allem mit der Anzahl der Zustände des Referenzautomaten zu. Um den komplexesten der erstellten Referenzautomaten mit drei Transitionen pro Zustand und insgesamt 70 Zuständen zu erlernen, wurde ein Referenztrace von 10^7 Zeichen benötigt.

Es zeigt sich, dass es ein mindestens quadratisches Wachstum der benötigten Länge des Referenztraces in Bezug auf die Komplexität des zu erlernenden Referenzautomaten gibt. Allerdings ist positiv festzuhalten, dass dieses Wachstum erst bei mehr als 20 Zuständen signifikant zunimmt.

Auswirkung der Alphabetgröße auf den L^*-Algorithmus

Neben der Komplexität des zugrunde liegenden Referenzautomaten kann die Alphabetgröße eine Auswirkung auf das Verhalten bzw. die Ergebnisse des Lernverfahrens haben. Eine Vergrößerung der Alphabetgröße geht mit der Vergrößerung des möglichen Lösungsraumes einher, da bei gleicher Wortlänge mehr Variationen, also mögliche Wörter, denkbar sind.

Zur Evaluierung des Einflusses der Wortlänge auf das Verhalten des L^*-Algorithmus werden Referenzautomaten mit jeweils zehn Zuständen, 20 Transitionen und einer Alphabetgröße von 3 bis 20 Zeichen zufällig generiert. Da alle Referenzautomaten dieser Form zuverlässig gelernt werden, wird die benötigte Zeit als Indikator für den Einfluss der Alphabetgröße herangezogen.

Wie in Abbildung 3-13 zu sehen ist, kann kein signifikanter Zusammenhang zwischen der Lernzeit und der Alphabetgröße festgestellt werden. Durch den weitaus größeren Einfluss der Komplexiät der Referenzautomaten sind etwaige Effekte, ausgehend von der Alphabetgröße, nicht erkennbar.

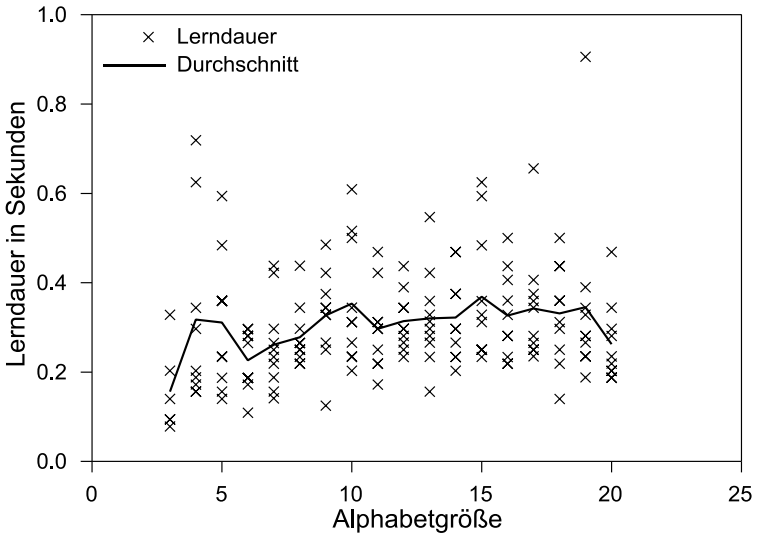

Abbildung 3-13 Einfluss der Alphabetgröße auf die Lernergebnisse des L^*-Algorithmus

Performance

Bei der Evaluierung der Performance der Lernphase ergeben sich keine großen Überraschungen. Wie in Abbildung 3-14 zu sehen ist, wächst die benötigte Zeit zum Lernen in etwa linear mit der Länge des Referenztraces.

Abbildung 3-15 zeigt die Abhängigkeit der Lerndauer von der Anzahl der Zustände des zugrunde liegenden Referenzautomaten. Hier ist festzustellen, dass die Lerndauer ungefähr quadratisch mit der Anzahl der zu erlernenden Zustände wächst.

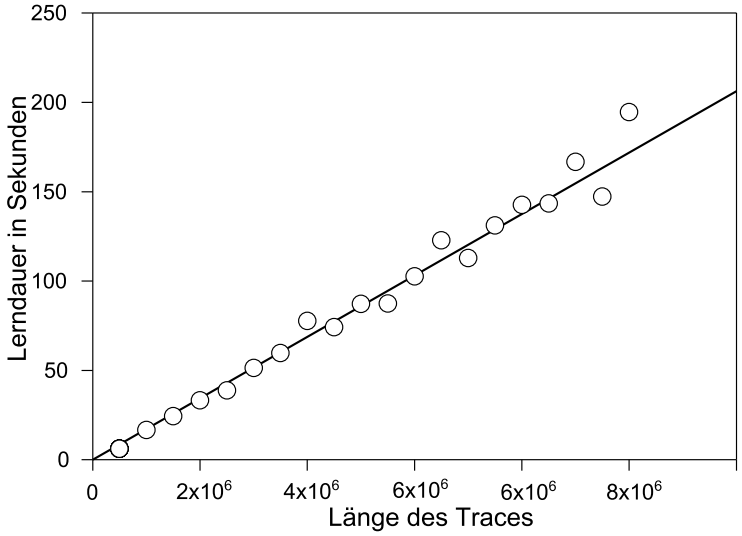

Abbildung 3-14 Lerndauer von Referenzautomaten mit zehn Zuständen und zehn Transitionen mit verschiedenen Referenztracelängen

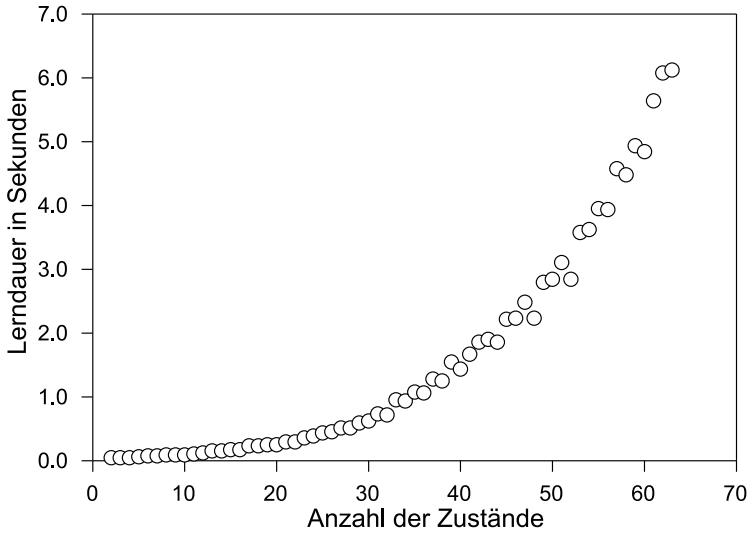

Abbildung 3-15 Lerndauer von Referenzautomaten mit 3 bis 60 Zuständen

3.3.3.2 Evaluierung für die Anwendung im selbstlernenden OoN-Erkennungsverfahren

Allgemein ist festzuhalten, dass der L^*-Algorithmus eines der transparentesten Verfahren zum Lernen von Abhängigkeiten darstellt. Auch die Realisierung des Teachers, welcher gemeinhin als Schwachpunkt des Verfahrens gilt, wurde, wie im vorherigen Kapitel gezeigt, gelöst.

Durch die relativ kurzen Laufzeiten beim Lernen konnte auf ein statistisches Sampling des Referenztraces verzichtet und eine lineare Suche über alle möglichen Wörter im Referenztrace angewendet werden. Durch die vollständige Suche und das bekannte

Verhalten des L^*-Algorithmus, stets den kleinsten korrekten Automaten zu erstellen, kann der erlernte Automat als die kleinste mögliche Abbildung der Zusammenhänge des gegebenen Referenztraces angesehen werden.

Klassifizierungsgüte

Erkennung tatsächlicher Abweichungen

Die Klassifizierungsgüte der von L^* gelernten Automaten hängt fast ausschließlich von der maximal geprüften Wortlänge n und der Beschaffenheit des Referenztraces ab.

Gemäß der hier vorgestellten Implementierung wird L^* einen Automaten erlernen, der alle mit der Wortlänge n beschreibbaren Zustände enthält, welche auch im Referenztrace enthalten sind.

Ob eine Abweichung im Sinne der OoN-Erkennung erkannt wird, hängt von der Struktur des gelernten Automaten und wiederum von der getesteten Wortlänge n ab. Abweichungen, welche nur mit größeren Wortlängen als n beschrieben werden können (Langzeitabhängigkeit), werden nicht zwangsläufig erkannt. Für Abweichungen bis zur getesteten Wortlänge kann die Erkennung garantiert werden.

Abhängigkeiten darüber hinaus hängen vom Lernverlauf und der daraus resultierenden Struktur des erlernten Automaten ab. Führt eine Abweichung nicht dazu, dass ein möglicher anderer Systemzustand beschrieben wird (Abweichung ist kein gültiges Wort), so kann dies ebenfalls sicher erkannt werden. Somit hängt die Erkennung von Abweichungen von der Hamming-Distanz der Wörter der Sprache $\mathcal{L}(\mathbb{A})$ ab.

Fehlalarmrate

Rein formal wird der erlernte Automat keine Fehlalarme auslösen, da das Verhalten des Referenztraces bis zur getesteten Wortlänge exakt abgebildet werden kann. Eine weite mögliche Ursachen für Fehlalarme stellt zudem das Overfitting dar (siehe Overfitting in Kapitel 3.2.2.4 und nicht abhängige Alphabete in Kapitel 5.3). Der Zusammenhang zwischen der Fehlalarmrate und der Erkennung von Abweichungen ist in Abbildung 3-16 dargestellt.

Langzeitabhängigkeiten

Die abgebildete Langzeitabhängigkeit hat ihre untere Grenze in der maximal getesteten Wortlänge n; d. h. in diesem Intervall werden alle Langzeitabhängigkeiten aufgelöst. Das bedeutet aber nicht zwangsläufig, dass bei einem Abstand von mehr als n Zeichen die Langzeitabhängigkeit nicht aufgelöst werden kann.

Ausgehend von diesen Beobachtungstabellen werden die sogenannten Zugehörigkeits-fragen gestellt. Diese können durchaus Zeichensequenzen abfragen, welche bedeutend länger als die Wortlänge sind. Zudem kann der Test, ob das Referenztrace vom gelernten Automaten akzeptiert wird (Äquivalenzfrage Teil (b), Kapitel 3.3.3.1), auch längere Subtraces zurückliefern. Somit werden teilweise bereits mit der getesteten Wortlänge 6 bei einer ARGII-1 Langzeitabhängigkeiten der Entfernung 10 aufgelöst.

Sind weiterhin im erlernten Automaten Schleifen vorhanden, können diese beliebig oft durchlaufen werden. So wird die Anzahl der Zeichen zwischen den relevanten

Zuständen außerhalb der Schleife signifikant größer sein als die getestete Wortlänge. Dies ist der wesentliche Unterschied zu einer Sliding-Window-Technik. Bei der Sliding-Window-Technik sind die obere und untere Grenze gleich der Wortlänge n, Schleifen können somit mit der Sliding-Window-Technik nicht aufgelöst werden.

Einbeziehung von Wissen aus vorhandener Spezifikation

Dass es möglich ist, bestehende Spezifikationen, welche anhand von Message Sequence Charts vorliegen, mittels des L^*-Algorithmus zu erlernen, haben bereits (Bollig u. a. 2010) gezeigt. Weiterhin wurde in (Berg u. a. 2005) demonstriert, dass es möglich ist, bestehende Spezifikation mit Hilfe des L^*-Algorithmus an abweichende Implementierungen anzupassen.

Durch die Anwendung der Äquivalenzfrage kann beinahe jedes positive und negative Verhaltensmuster in den Lernprozess aufgenommen werden. Das bedeutet, dass es möglich ist, explizit gewolltes (positives) Verhalten aufzunehmen oder explizit nicht erlaubtes (negatives) Verhalten aus dem erlernten Automaten zu entfernen. Solange dies jeweils in Sequenzen bzw. Traces abbildbar ist, kann das Lernverfahren mit beliebig vielen zusätzlichen positiven und negativen Beispielen erweitert werden, ohne dabei signifikante Änderungen im Lernprozess vornehmen zu müssen.

Overfitting

Overfitting kann entstehen, wenn die Länge des Referenztraces nur unwesentlich größer ist als die maximal getestete Wortlänge n. In diesem Fall würde ein Automat mit nur einem Handlungsstrang ähnlich zu Automat (ii) in Abbildung 3-16 erlernt werden. Ein solcher erlernter Automat wäre zu speziell auf das vorliegende Referenztrace angepasst und würde alle zu prüfenden Traces verwerfen.

Vermeiden lässt sich Overfitting durch die Anpassung der Wortlänge n an die zur Verfügung stehende Länge des Referenztraces. Für die Evaluierung des selbstlernenden OoN-Erkennungsverfahrens mit realen Daten (Kapitel 5) wurde die Wortlänge $n = 6$ gewählt und es wurden nur Subtraces mit mehr als 100 Events zugelassen. Diese Größenordnung hat bei den vorliegenden Daten keine Overfitting-Effekte hervorgebracht.

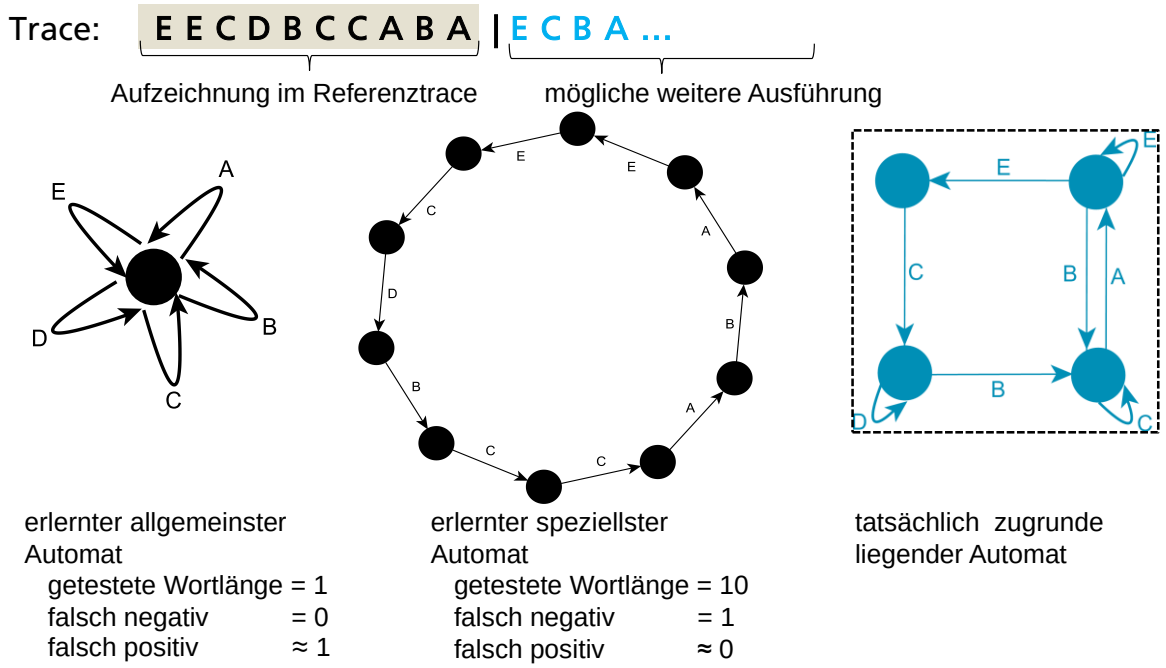

Abbildung 3-16 Zusammenhang zwischen fp- und fa-Rate im Verhältnis zu den gelernten Automaten und dem zugrunde liegenden Automaten

Bootstrapping

Der L^*-Algorithmus liefert einen DEA (Akzeptorautomat) als Hypothese über das evaluierte System. Somit lassen sich relativ genaue Rückschlüsse über die Komplexität und den Aufbau des evaluierten Systems ziehen.

Abbruchkriterien für Lernverfahren

Der Lernvorgang der hier vorgestellten Anwendung und Implementierung des L^*-Algorithmus kann mit jeder beliebigen Wortlänge abgebrochen werden, wenn für diese Wortlänge die Äquivalenzfrage positiv beantwortet worden ist. D. h., dass jeweils zu diesem Zeitpunkt ein DEA erlernt worden ist, der die oben genannten Kriterien bis zur Wortlänge n erfüllt. Ein Abbruch mit einer negativ beantworteten Äquivalenzfrage ist nicht sinnvoll, da in diesem Fall die Hypothese nicht bestätigt worden ist und keine verlässlichen Aussagen über den gelernten Automaten gemacht werden können. In diesem Fall muss die nächstkleinere Wortlänge $n-1$ als gültige Lösung herangezogen werden.

Es ist möglich, das Lernverfahren nach dem Erreichen einer festgelegten Wortlänge abzubrechen. Die Rechenzeit bis zu einer festen Wortlänge kann allerdings (wie weiter unten beschrieben) sehr unterschiedlich ausfallen. Aufgrund der Abhängigkeit von der zugrunde liegenden Datenstruktur, was bis zu einem exponentiellen Anstieg der Rechenzeit führen kann, sind sehr lange Berechnungszeiten bis hin zu quasi nicht berechenbaren Problemen möglich.

Die Berechnung einer sinnvollen Grenze für die maximal zu testende Wortlänge wird vom Autor in (Langer, Bertulies, Hoffmann 2011) beschrieben. Hier wird die Struktur des gelernten Automaten genutzt, um eine statistische Abschätzung für die maximal zu testende Wortlänge zu erhalten. Dieser Ansatz funktioniert gut für stochastisch verteilte

Daten, ist aber nur begrenzt für Netzwerktraces einsetzbar (Langer, Bertulies, Hoffmann 2011).

Alternativ kann das Lernverfahren nach einer maximal verstrichenen Zeit zu jedem Zeitpunkt abgebrochen werden. In diesem Fall muss, wie in Kapitel 3.3.3.1 beschrieben, auf die zuletzt positiv getestete Wortlänge $n-1$ zurückgegriffen werden. Kann ein Automat mit der minimalen Wortlänge $n=2$ nicht erlernt werden, so kann für dieses Referenztrace kein gültiger Automat erlernt werden.

Effizienz und Performance

Performance bzw. Rechenkomplexität des Lernverfahrens

Die Performance der Realisierung des L^* hängt sehr stark vom Referenztrace bzw. dem zugrunde liegenden System ab. Besitzt das Referenztrace starke Abhängigkeiten, werden sehr schnell die zugehörigen Automaten mit meist wenigen Zuständen pro Transition erlernt.

Es zeigt sich, dass vor allem die Anzahl der Transitionen pro Zustand ein gutes Maß für die Abhängigkeit der Daten ist. Für einen zu erlernenden Automaten mit weniger Transitionen pro Zustand (Abbildung 3-8 mit zwei Transitionen pro Zustand) sind in der Regel bedeutend weniger Äquivalenzfragen notwendig als für einen Automaten mit mehr Transitionen pro Zustand (Abbildung 3-7 mit vier Transitionen pro Zustand). Dieses Verhalten verschärft sich zunehmend mit größer werdender Wortlänge n. Angluin gibt für das Hinzufügen eines Gegenbeispiels eine Anzahl von $\mathcal{O}(n|\mathrm{Z}|)$ Zugehörigkeitsfragen an.

Es lässt sich zusammenfassen, dass die Performance von L^* bei der Anwendung auf die Problemstellung sehr stark von der Komplexität der Eingangsdaten und dem resultierenden erlernten Automaten abhängt. Positiv zu bemerken ist, dass Eingangsdaten mit hoher Abhängigkeit (einfacher Automat) viel schneller verarbeitet werden können als Daten mit keiner oder nur geringer Abhängigkeit (komplexer Automat).

Rechenkomplexität in der Anwendungsphase

Für die Anwendung im selbstlernenden OoN-Erkennungsverfahren werden die erlernten DEA als Akzeptorautomaten verwendet. Der Rechenaufwand dabei ist sehr gering, da jeweils nur geprüft werden muss, ob von dem jeweiligen Zustand eine Transition mit einem entsprechenden Zeichen abgeht oder nicht. Es hat sich gezeigt, dass die Größe der erlernten Automaten meist unter 1000 Zuständen bei weniger als zwei Zuständen pro Transition liegt. Somit ist der nötige Speicherbedarf für die erlernten Automaten vergleichsweise gering.

3.3.4 Zusammenfassender Vergleich der Lernverfahren

Nach der Evaluierung der drei vorgestellten Lernverfahren werden an diese Stelle ein Vergleich der Verfahren und eine abschließende Bewertung der Ergebnisse durchgeführt. Dazu werden die Ergebnisse in Bezug auf die eingeführten Evaluierungskriterien gegenübergestellt, so dass der am besten geeignete Algorithmus für weitere Untersuchungen gewählt werden kann.

Grundlegend konnte für alle drei in Betracht gezogenen Lernverfahren die prinzipielle Anwendbarkeit auf das selbstlernende OoN-Erkennungsverfahren gezeigt werden. Da es große Unterschiede in der Wirkungsweise und der Anwendung auf die Problemstellung gibt, sind die Ergebnisse der einzelnen Lernverfahren nicht direkt miteinander vergleichbar. Um dieses Problem zu lösen, wurden die in Tabelle 3-6 aufgelisteten, für das selbstlernende OoN-Erkennungverfahren wichtigen Evaluierungskriterien definiert. Obwohl für viele Kriterien genaue Zahlen aus den durchgeführten Versuchen vorliegen, wurde eine relative Bewertung vorgenommen. Dies resultiert vor allem daraus, dass die Evaluierung anhand von künstlich erzeugten Daten vorgenommen wurde und somit die ermittelten Zahlenwerte eher eine Tendenz abbilden als eine genaue Gewichtung.

Zusammenfassend lässt sich feststellen, dass Markov-Modelle sehr einfach anzuwenden sind, aber sehr viel Speicherbedarf für die erstellten Modelle benötigen. Die LSTM-Netze, als Spezialfall der künstlichen neuronalen Netze, besitzen zweifelsohne die besten Lerneigenschaften bei den Langzeitabhängigkeiten. Allerdings sind LSTM-Netze aufgrund ihrer intransparenten Abbruchkriterien und der hohen Fehlalarmrate sehr schwierig anzuwenden.

Obwohl der L^*-Algorithmus nicht in allen Kriterien die beste Alternative darstellt, scheint er vor allem aufgrund seiner hohen Transparenz des Lernprozesses und der sehr guten Klassifizierungsleistung (Fehlalarmrate und Erkennung tatsächlicher Abweichung) die beste Wahl zu sein. Zudem hat die zusätzlich durchgeführte Evaluierung in Bezug auf das Skalierungsverhalten bei unterschiedlichen Alphabetgrößen und Referenztracelängen gute Werte geliefert. Deshalb wird in den folgenden Kapiteln der L^*-Algorithmus in der vorgestellten Implementierung für weitere Untersuchungen verwendet.

Tabelle 3-6 Vergleich der Eignung der Lernverfahren für die Problemstellung der OoN-Erkennung

Bewertungskriterium	Markov	KNN	Angluin L^*
Erkennung tatsächlicher Abweichungen	+	+	++
Fehlalarmrate	+	–	++
Langzeitabhängigkeiten	o	++	–
Einbeziehung von Wissen aus vorhandener Spezifikation	+	– –	++
Overfitting	++	–	++
Bootstrapping	+	– –	++
Abbruchkriterien für Lernverfahren	++	–	+
Performance bzw. Rechenkomplexität des Lernverfahrens	+	–	+
Rechenkomplexität in der Anwendungsphase	–	++	++

Legende: -- sehr schlecht; - schlecht; o mittel; + gut; ++ sehr gut

3.4 Anwendung auf reale Datensätze

Die vorgestellten Lernverfahren wurden bisher durchweg mit künstlichen Datensätzen evaluiert. Wichtige Eigenschaften wie die Klassifizierungsgüte können nur mit Referenzdaten bestimmt werden, deren Eigenschaften bekannt sind. Daher sind die Versuche mit künstlich erzeugten Daten, bei denen alle Zusammenhänge bekannt sind, sehr wichtig für eine vergleichbare Evaluierung unterschiedlicher Lernverfahren. Die künstlich erzeugten Traces folgen der aufgestellten Vermutung über das zugrunde liegende Systemverhalten (Kapitel 2.2).

Um herauszufinden, inwieweit die Systemhypothese der Realität entspricht und die evaluierten Verfahren tatsächlich in der Lage sind, die vorgegebene Problemstellung zu lösen, ist die Anwendung auf reale Fahrzeugdaten unerlässlich. Daher wird im Folgenden eine Evaluierung mittels tatsächlicher Netzwerkaufzeichnungen eines Fahrzeug-CAN durchgeführt. Mit den so gewonnenen Erkenntnissen können weitere Optimierungen durchgeführt und auftretende Unzulänglichkeiten identifiziert werden.

Als realer Referenzdatensatz steht eine Netzwerkaufzeichnung des Antriebs-CAN des Frecc0-Versuchsträgers in der Variante mit Verbrennungsmotor zur Verfügung. Der Versuchsträger wurde innerhalb des Projektes Fraunhofer Systemforschung

Elektromobilität (Buller, Hanselka 2012) verwendet. Die getestete Variante des Versuchsträgers entspricht dem Serienprodukt Artega GT des Kleinserienherstellers Artega GmbH & Co. KG. Da es sich in diesem Fall um ein seriennahes Fahrzeug handelt, kann für die erstellten Netzwerkaufzeichnungen von einem fehlerfreien und produktnahen Verhalten ausgegangen werden.

3.4.1 Eigenschaften eines Netzwerktraces

Die Aufzeichnung des Kommunikationsverhaltens eines Netzwerkes wird als Netzwerktrace bezeichnet. Die allgemeine Definition eines Traces wird in Kapitel 2.2.2 in Definition 2-13 erstellt. Ein Netzwerktrace hat hiervon abweichend einige zusätzliche bzw. spezielle Merkmale, welche in diesem Kapitel beschrieben werden. Eine Umwandlung in die allgemeine Form nach Definition 2-13 wird im nächsten Kapitel beschrieben. Da es sich bei den vorliegenden Daten um Aufzeichnungen eines Fahrzeug-CAN-Busses handelt, werden an dieser Stelle schwerpunktmäßig die wichtigsten Eigenschaften des dort verwendeten CAN-Busses beschrieben.

Der CAN-Bus ist, wie der Name schon sagt, ein Bus. Das bedeutet, dass die dort gesendeten Nachrichten von jedem Teilnehmer gelesen werden können und es keine Verbindungsmerkmale wie Sender und Empfängeradresse gibt. Eine Nachricht auf dem CAN-Bus ist maximal 8 Byte lang und wird durch eine eindeutige CAN-ID identifiziert. Im Fahrzeugbereich wird üblicherweise für jede Botschaft (Nachricht mit eindeutiger CAN-ID) ein festes Belegungsschema der beinhaltenden Daten definiert. So werden pro CAN-ID meist mehrere sog. Signale definiert. Deren Bit-Position, Länge und Codierung werden in einem sog. Nachrichtenkatalog beschrieben. Somit kann jeder Empfänger anhand der CAN-ID und des Nachrichtenkataloges die Bedeutung und die einzelnen Werte der Signale bestimmen. Ein Signal bestimmt dabei meist eine gewisse Eigenschaft oder einen bestimmten Messwert. Häufig wird in diesem Zusammenhang auch von Variablen gesprochen, welche übertragen werden.

Weiterhin werden CAN-Botschaften sehr häufig zyklisch versendet. Das bedeutet, dass eine CAN-Botschaft in einer festgelegten Frequenz versendet wird. Dies ist zum einen eine Art Aktivitätskennzeichnung des Senders und zum anderen wird dadurch sichergestellt, dass gesendete Daten auch beim Empfänger ankommen. Durch das zyklische Senden enthält nicht jede CAN-Botschaft neue Werte, da immer der letzte bekannte Wert einer Variablen gesendet wird, auch wenn dieser sich nicht geändert hat.

Bei der Aufzeichnung der CAN-Botschaften mit einem Analysewerkzeug wird jeder gespeicherten Botschaft dessen Empfangszeit (Zeitstempel) hinzugefügt.

3.4.2 Datenvorverarbeitung

Ein Event nach Definition 2-13 ist ein eindeutiges Ereignis, welches die Änderung des Systemzustandes markiert. Um dieser Definition gerecht zu werden, müssen die vorliegenden Netzwerkaufzeichnungen entsprechend vorverarbeitet werden. Daher werden in der Datenvorverarbeitung aus dem aufgezeichneten Netzwerktrace eindeutige Events extrahiert.

Die Abbildung von Netzwerkverkehr auf eindeutige Events hängt sehr stark sowohl von der Form des Netzwerkverkehrs als auch von der Anwendungsebene, auf der die spätere Analyse stattfinden soll, ab. So können beispielsweise für die TCP/IP-Kommunikation

Informationen aus dem Übertragungsprotokoll wie z. B. IP-Adresse, Port und weitere protokollinterne Informationen genutzt werden, um die Zustände einer TCP/IP-Kommunikation entsprechend aufzubereiten. Sollen höher liegende Schichten oder auch Informationen aus Applikationsdaten genutzt werden, so müssen jeweils deren Kommunikationsschemata oder die Datenbelegung bekannt sein.

Im Fall der Verarbeitung von CAN-Daten stehen außer der CAN-ID keine weiteren Verbindungsdaten zur Verfügung. Die Auswertung muss daher auf der Ebene der versendeten Applikationsdaten aufsetzen. Die Datenbelegung der aufgezeichneten Nachrichten ergibt sich aus dem Nachrichtenkatalog, wie im vorhergehenden Kapitel beschrieben. Daher werden in einem ersten Schritt mit Hilfe des Nachrichtenkataloges die Rohdaten der CAN-Botschaften in Signale mit den zugehörigen Werten übersetzt. Von den so gewonnenen Signalen werden diejenigen mit kontinuierlichen Messgrößen wie z. B. Geschwindigkeit verworfen, da diese keine direkt verwertbaren Zustandsinformationen enthalten. In die weitere Verarbeitung werden daher nur Signale mit diskreten ganzzahligen Werten übernommen.

Nachdem die entsprechenden verwertbaren Signale für die OoN-Erkennung identifiziert sind, werden anhand der jeweiligen Datenwerte entsprechende Events generiert. Dabei ergibt jeweils ein bestimmter Wert eines Signals ein eindeutiges Event. Es wird demnach jeweils aus der Kombination Signalname und Signalwert eine Event-ID generiert. So ergibt beispielsweise das Signal „*Blinker-Links"* mit den Werten 0 für „*aus*" und 1 für „*ein*" zwei verschiedene Events. Die Erzeugung der Event-IDs muss eindeutig pro Name/Wert-Kombination erfolgen. In der vorliegenden Implementierung werden die Event-IDs fortlaufend vergeben.

Zur Erzeugung des Traces wird eine Zuordnungstabelle erstellt, welche das Mapping von Signalname und Signalwert auf die Event-ID zulässt und umgekehrt. Mit Hilfe dieser Tabelle werden nun die vorliegende Netzwerkaufzeichnung verarbeitet und für jeden Fund der Kombination Signalname und Signalwert die entsprechende Event-ID in das für die OoN-Erkennung zu verwendende Trace geschrieben. Zusätzlich zur Event-ID wird der Zeitstempel des Events in das Trace übernommen.

Nach Definition 2-13 muss ein Event eine Änderung des Systemzustandes kennzeichnen. Diese Forderung wird durch das zyklisch wiederholte Senden der gleichen Zustandsinformation in CAN-Netzwerken verletzt. Daher werden bei der Erzeugung des Traces nur Event-IDs in das Trace übernommen, wenn es sich beim jeweiligen Signal tatsächlich um einen neuen Wert handelt. Somit werden Wiederholungen des gleichen Events verhindert.

3.4.3 Ergebnisse der Anwendung des L^*-Algorithmus auf reale Datensätze

Für einen ersten Test der Anwendung eines Lernverfahrens auf reale Referenztraces wird an dieser Stelle der L^*-Algorithmus gewählt. Dieser schneidet in der Evaluierung der Lernverfahren sehr gut ab (Kapitel 3.3.4). Er bietet vor allem durch seine sehr guten Bootstrapping-Eigenschaften einen guten Einblick, welche Qualität das Lernergebnis hat. Durch die Analyse des erzeugten Automaten und der maximalen Testlänge lässt sich ein guter Rückschluss auf die tatsächliche Lernleistung ableiten.

Aus den Versuchen mit der vorgestellten L^*-Implementierung ist bekannt, dass dieser bis zu einer Anzahl von maximal 100 Zuständen recht gut skaliert und die Alphabetgröße keinen signifikanten Einfluss auf die Lernergebnisse hat (Kapitel 3.3.3.1). Mit diesem Hintergrund wurden daher aus dem CAN-Referenztrace des Versuchsträgers mit einer Aufzeichnungsdauer von ca. 13 min 40 Signale manuell ausgewählt, von denen zu vermuten ist, dass es zwischen diesen einen Zusammenhang bzw. Abhängigkeiten gibt.

Nach der Datenvorverarbeitung wurden aus diesen 40 Signalen 85 verschiedene Events (Alphabetgröße) ermittelt, welche ein resultierendes Referenztrace mit einer Länge von insgesamt 401.660 Events beschreiben. Im Folgenden wird der Begriff Zeichen als Äquivalent für Event verwendet. Die Ergebnisse der Anwendung des L^*-Algorithmus auf dieses Trace wurden vom Autor bereits in (Langer, Bertulies, Hoffmann 2011) beschrieben. Der L^*-Algorithmus erlernt demnach für das beschriebene Referenztrace den in Abbildung 3-17 gezeigten Automaten.

Der Lernvorgang erreicht nach 48 Minuten Rechenzeit eine maximale Testlänge von lediglich $n = 2$ Zeichen und kann diese auch nach mehrstündiger weiterer Ausführung des Algorithmus nicht erhöhen. Der resultierende Automat mit der Testlänge von $n = 2$ Zeichen besitzt 123 Zustände und 1.023 Transitionen. Es ergibt sich demnach eine Komplexität von 8,5 Transitionen pro Zustand. Durch die geringe maximale Testlänge von $n = 2$ Zeichen werden nur sehr wenige Abhängigkeiten aufgelöst. Dies bedeutet, dass bei der Verwendung in einem selbstlernenden OoN-Erkennungsverfahren nur sehr wenige Abweichungen des Verhaltens erkannt werden würden.

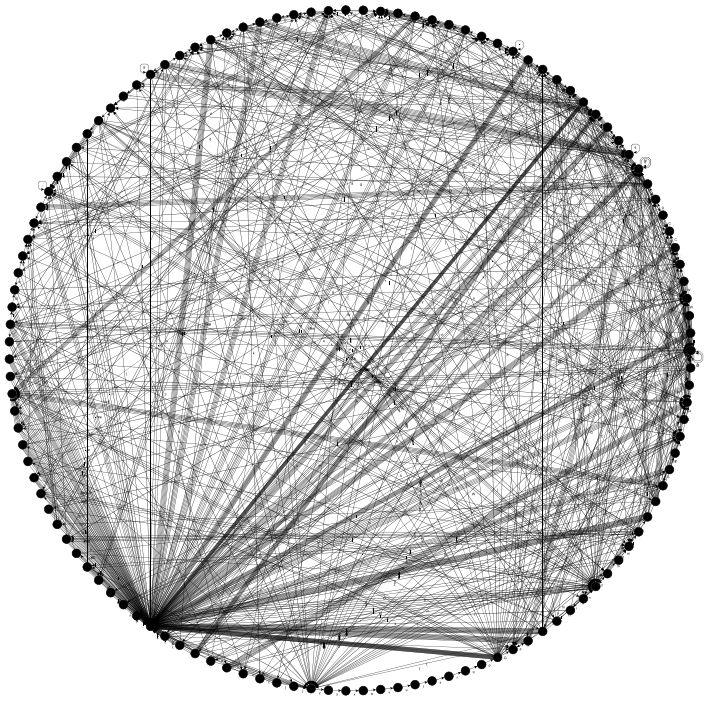

Abbildung 3-17 Gelernter Automat aus realen Daten nach (Langer, Bertulies, Hoffmann 2011)

Das Experiment zeigt somit, dass die Zusammenhänge der manuell ausgewählten Signale wohl zu komplex sind, als dass sie vom angewandten Lernverfahren mit den zur Verfügung stehenden Daten aufgelöst werden könnten.

Es ist davon auszugehen, dass trotz der manuellen Auswahl der Signale noch zu viele nicht voneinander abhängige Ereignisse im zu lernenden Referenztrace enthalten sind. Wie bereits in der Systemvermutung (Kapitel 2.2) geäußert, werden diese unabhängigen Ereignisse von nebenläufigen Automaten generiert und das Produkt von nebenläufigen unabhängigen Automaten wird entsprechend komplex.

Es muss demnach eine bessere Auswahl der Events getroffen werden, welche jeweils durch einen einzelnen Lernvorgang bearbeitet werden. Um dies zu erreichen, kann eine existierende Systemspezifikation herangezogen werden, um entsprechend abhängige Ereignisse zu finden. Häufig ist eine genaue Beschreibung der Abhängigkeiten von Ereignissen auf Netzwerkebene aber nicht vorhanden, da, wie in dieser Arbeit einleitend erwähnt (Kapitel 1), in diesem Bereich die Systemspezifikationen häufig lückenhaft ist. Daher wird im Kapitel 4 eine Lösung vorgestellt, welche es ermöglicht, diesen Vorgang ohne Zuhilfenahme einer Spezifikation zu bearbeiten.

3.5 Fazit

Kapitel 3 untersucht eingehend die Anwendung möglicher Lernverfahren für das Problem der selbstlernenden OoN-Erkennung. Untersucht wurden dabei das Lernen von Markov-Modellen, von künstlichen neuronalen Netze in Form von Long short term memory (LSTM) Netzen und das Angluin L^*-Lernverfahren zum Erlernen von Akzeptorautomaten.

Es konnte gezeigt werden, dass diese drei Verfahren auf die Problemstellung anwendbar sind. Sowohl für die LSTM-Netze als auch den L^*-Algorithmus wurde jeweils eine eigene Implementierung erstellt. Zudem wurden für die LSTM-Netze als auch den L^*-Algorithmus Erweiterungen vorgenommen, um sie für die Problemstellung anwendbar zu machen.

Anhand der aufgezeigten Evaluierungskriterien wurden die Lernverfahren ausführlich verglichen und bewertet. Hierbei zeigt sich, dass auch nach einigen Optimierungen kein Algorithmus ein für alle Kriterien optimales Ergebnis liefert. Aufgrund der besseren Transparenz und Beherrschbarkeit wird aber die Realisierung des L^*-Algorithmus als tragfähigste Lösung für die gegebene Problemstellung angesehen.

Die Evaluierung wurde für eine bessere Vergleichbarkeit auf Basis künstlicher Datensätze vorgenommen. Es zeigt sich bei einer abschließenden Anwendung des L^*-Algorithmus auf reale Datensätze, dass die zu erwartende Komplexität des realen Lernproblems die Fähigkeiten des Lernverfahrens überschreitet. Daher werden im nächsten Kapitel 4 Verfahren untersucht, mit deren Hilfe es möglich ist, das Lernproblem so weit zu vereinfachen, dass es für Lernverfahren handhabbar wird und nutzbare Abhängigkeitsmodelle für ein selbstlernendes OoN-Erkennungsverfahren liefert.

4 Identifizierung von parallelen Handlungssträngen in Netzwerktraces

Bei der Evaluierung der Lernverfahren konnte gezeigt werden, dass es möglich ist, aus einem Referenztrace Abhängigkeitsmodelle zu extrahieren. Die Evaluierung an realen Datensätzen hat aber gezeigt, dass die Komplexität der Problemstellung die Fähigkeiten der angewendeten Lernverfahren deutlich übersteigt. Selbst eine manuelle Auswahl bestimmter Netzwerkbotschaften führt schon zu einer nicht mehr handhabbaren Größenordnung von verschiedenen Events im aufgezeichneten Referenztrace.

Dieses Kapitel widmet sich der Fragestellung, wie die Komplexität der von den Lernverfahren verarbeiteten Daten verringert werden kann. Die Lösung dieses Problems ist nach den bisherigen Untersuchungen die Voraussetzung für die Machbarkeit des selbstlernenden OoN-Erkennungsverfahrens. Grundlegend besteht die Lösungsidee in der Identifizierung von parallelen Handlungssträngen im Referenztrace. Dadurch werden einzelne Subtraces extrahiert, welche eine signifikant geringere Komplexität als das Referenztrace aufweisen, so dass sinnvolle Abhängigkeitsmodelle extrahiert werden können.

Das Kapitel 4 gliedert sich dabei wie folgt:

Zunächst werden in Kapitel 4.1 eine genaue Problembeschreibung und ein daraus resultierender möglicher Weg zur Problemlösung aufgezeigt. Hierbei wird, ausgehend von der in Kapitel 2.2 aufgestellten Systemvermutung, die Lösungsmöglichkeit beschrieben, welche eine exponentiell abnehmende Komplexität des zu lernenden Verhaltens verspricht. Es wird aber auch gezeigt, dass eine arithmetische Lösung des gezeigten Problems aufgrund zu wenig bekannter Parameter nicht möglich ist. Daher basieren die später vorgestellten Lösungsansätze auf heuristischen Methoden. Ausgehend von den bekannten Systemeigenschaften wird in Kapitel 4.2 eine Evaluierungsmethodik für die Bewertung der Verfahren zur Verringerung der Komplexität aufgezeigt und vorgestellt.

In den beiden folgenden Kapiteln 4.3 und 4.4 werden zwei speziell für diese Problemstellung entworfene Clustering-Verfahren vorgestellt. So wird in Kapitel 4.3 ein neues Verfahren gezeigt, welches auf Basis des zeitlichen Verhaltens mögliche parallele Handlungsstränge identifiziert. Hierzu wird das Frequenzspektrum der einzelnen Events berechnet und die entsprechenden Frequenzanteile werden als Merkmale für die Gruppierung bzw. das Clustering der Events herangezogen. Ein zweites Clustering-Verfahren wird in Kapitel 4.4 vorgestellt. Dieses nutzt die informationstheoretische Entropie als Eigenschaft, um das Clustering-Problem zu beschreiben und zu lösen. Nach der Vorstellung der zwei neu entwickelten Clustering-Verfahren wird eine Optimierung, welche auf der Anwendung der Ergebnisse aus mehreren Clustering-Verfahren beruht, vorgestellt und evaluiert.

4.1 Problembeschreibung und Lösungsansatz

Wird der Argumentation aus Gleichung 2-14 gefolgt, kann eine wesentliche Reduzierung der abzubildenden Zustände erreicht werden, wenn es gelingt, die beobachteten Events eines Referenztraces so zu gruppieren, dass diese jeweils zu einem unabhängigen Ausführungsgraphen gehören. Zu einem unabhängigen Ausführungs-

graphen gehören Events, wenn sie von einem einzelnen zugrunde liegenden Automaten generiert worden sind. Somit soll das Lernverfahren nicht mehr A^{**} lernen, sondern ${A_n}^*$ (mit $A^{**} = A_0^* \times A_1^* \times A_2^* \times \dots A_i^*$), was einer sehr starken Reduzierung der Komplexität des zu lernenden Verhaltens gleichkommt.

Da sich nach Gleichung 2-15 die Anzahl der möglichen Zustände eines resultierenden Automaten A^{**} durch das Produkt aller Zustände der einzelnen Automaten ${A_n}^*$ ergibt, wächst die Anzahl der Zustände von A^{**} exponentiell im Vergleich zu der Anzahl der Zustände in ${A_n}^*$. Somit ergibt sich auch ein exponentielles Wachstum des Lernaufwandes für das anzuwendende Lernverfahren. Gelingt es, die Automaten ${A_n}^*$ zu identifizieren, so kann der Aufwand für ein Lernverfahren exponentiell reduziert werden.

Das Problem stellt sich mathematisch wie folgt dar: Gegeben ist ein Alphabet Σ^{**} mit allen Zeichen eines Traces. Es gilt, alle Subalphabete Σ_i^* zu identifizieren, so dass: $\Sigma^{**} = \biguplus \Sigma_i^*$ und $A^{**} = A_0^* \times A_1^* \times A_2^* \times \dots A_i^*$ (Gleichung 2-14). Es sollen somit disjunkte Subalphabete gefunden werden, welche zu unabhängig voneinander arbeitenden Automaten gehören. Automaten werden als unabhängig bezeichnet, wenn der resultierende Automat der Vereinigung der Automaten durch das Produkt der Automaten abgebildet wird (Gleichung 2-14). Die Auflösung eines Automaten A^{**}, welcher das Produkt mehrerer Automaten ${A_n}^*$ darstellt, ist daher verlustlos. D. h., es gehen keine Informationen über bestimmte Verhaltensweisen verloren. Dies trifft nur für das Produkt von Automaten zu, nicht aber für die Komposition mehrerer Automaten.

Da aber alle A^* unbekannt sind, sind auch alle Σ^* unbekannt. Es müssen also Zuordnungsmerkmale gefunden werden, so dass es möglich ist, Zeichen des Alphabetes Σ^{**} jeweils einem Subalphabet Σ_i^* zuzuordnen. Dieses Problem kann auch als Clustering bezeichnet werden, wobei die Anzahl und die Größe der Subalphabete nicht bekannt sind.

Der Cluster-Algorithmus muss die Zeichen so in Gruppen bzw. Cluster aufteilen, dass das Verhalten jeweils eines Clusters nur noch durch eine Komposition, aber nicht durch das Produkt mehrerer Automaten abbildbar ist. Ist dies der Fall, so wird durch die resultierenden Subtraces jeweils ein unabhängiger Ausführungsgraph eines solchen Automaten beschrieben.

Ein Beispiel dafür kann wie folgt aussehen:

Das Alphabet $\Sigma^{**} = \{a, b, c, d\}$ mit $\tau \rightarrow a, b, c, d, a, b, c, d$ kann in die Subalphabete $\Sigma_1^* = \{a, b\}$ mit $\tau_1 \rightarrow a, b, a, b$ und $\Sigma_2^* = \{c, d\}$ mit $\tau_2 \rightarrow c, d, c, d$ geclustert werden (Abbildung 4-1), wobei die Forderung gestellt wird, dass die Automaten, welche jeweils τ_1 und τ_2 abbilden können (A_1^* und A_2^*), voneinander unabhängig sein sollen. Unabhängig heißt in diesem Fall, dass der Automat, welcher τ abbildet (also A^{**}), sich aus der unabhängigen Vereinigung, also dem Produkt der Automaten A_1^* und A_2^* ($A^{**} = A_1^* \times A_2^*$), bildet.

Ist dies der Fall, kann zu jedem Zustand in A_1^* gleichzeitig jeder beliebige Zustand in A_2^* eingenommen werden und umgekehrt. Würde das obige Beispielalphabet Σ^{**} den Automaten in Abbildung 3-7 abbilden, so wäre diese Bedingung nicht erfüllt. Der Automat A^{**} wäre die Komposition aus den Automaten A_1^* und A_2^* ($A^{**} = A_1^* \circ A_2^*$) und

somit ist das Auftreten der Zeichen der Subalphabete Σ_1^* und Σ_2^* im Trace τ nicht voneinander unabhängig.

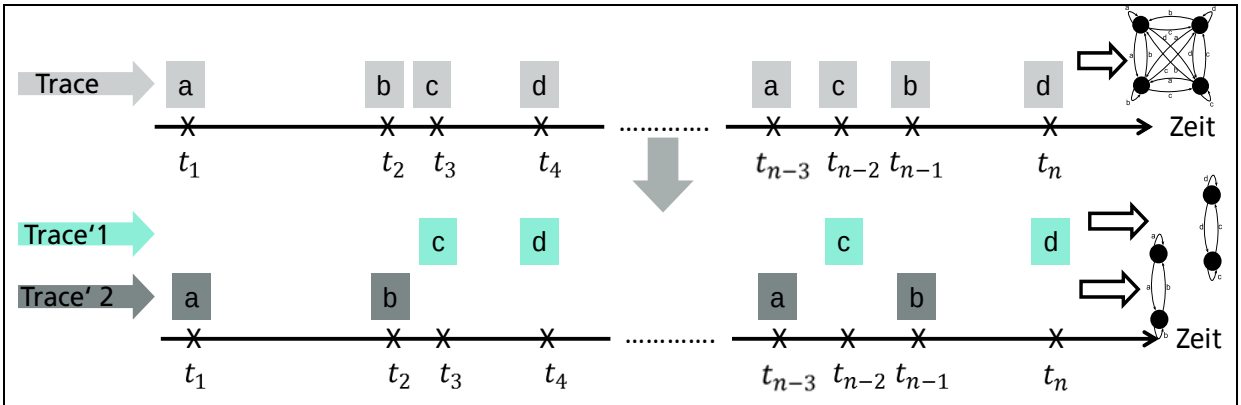

Abbildung 4-1 Beispiel eines möglichen Clusterings für das Alphabet $\boldsymbol{\Sigma^{**}} = \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}\}$ mit $\boldsymbol{\tau \rightarrow a, b, c, d, a, b, c, d}$

Könnten zu allen möglichen Kombinationen an Subalphabeten die zugehörigen Automaten bestimmt werden, so könnte geprüft werden, ob diese der Komposition ($A^{**} = A_1^* \circ A_2^*$) oder dem Produkt ($A^{**} = A_1^* \times A_2^*$) genügen. Da die zugrunde liegenden Automaten a priori nicht bekannt sind und zudem die Anzahl der möglichen Variationen an Subalphabeten sehr groß werden kann, ist eine Berechnung oder auch ein Ausprobieren aller möglichen Lösungen nicht möglich.

Der Clustering-Algorithmus stellt daher eine Heuristik dar, welche eine möglichst gute Annäherung an die Forderung disjunkter Automaten liefern soll. Um einen Clustering-Algorithmus anwenden zu können, müssen aus den Subalphabeten, den Subtraces oder auch einzelnen Zeichen des Alphabetes Merkmale extrahiert werden, welche es erlauben, sinnvolle Gruppen an Subalphabeten zu bilden.

4.1.1 Umgang mit unvollständigen Lösungen

Bei der Anwendung eines heuristischen Verfahrens wie in diesem Fall wird eine exakte Trennung der Subalphabete zur Auflösung der des Produktes $A^{**} = A_1^* \times A_2^*$ nach A_1^* und A_2^* nicht zwangsläufig für alle Automaten möglich sein. Eine exakte Trennung würde das Optimum der Lösung darstellen. Daher wird die heuristische Lösung eine gewisse Unschärfe aufweisen, was dazu führen kann, dass die gefundenen Automaten auch teilweise der Kompostion $A^{**} = A_1^* \circ A_2^*$ genügen.

Für die Erstellung von Abhängigkeitsbeschreibungen aus einem Referenztrace geht bei der Auflösung einer Komposition ein kleiner Teil der tatsächlich vorhandenen Abhängigkeitsbeziehungen verloren, da nur die Auflösung des Produktes verlustfrei durchführbar ist. Diesem Effekt kann entgegenwirkt werden, indem die Forderung disjunkter Alphabete für die Clustering-Verfahren reduziert wird. Die Überschneidung zweier abhängiger Alphabete bedeutet im Wesentlichen, dass diese Alphabete auch zu einem größeren Alphabet zusammengeführt werden könnten und dass das resultierende Alphabet ebenfalls ein abhängiges Alphabet ergibt. Da Clustering-Verfahren normalerweise keine doppelte Zuordnung eines Punktes zu mehreren Clustern erlauben,

wird ein derartiger Fall bei der Anwendung nur eines Clustering-Verfahrens nicht auftreten. Bei der Anwendung verschiedener Clustering-Verfahren kann eine Überschneidung der identifizierten Cluster durchaus vorkommen.

Somit ergibt sich für die Anwendung der Clustering-Verfahren nicht zwangsläufig die Anforderung, ein bestimmtes Clustering-Verfahren auszuwählen, sondern die Ergebnisse der Clustering-Verfahren zu kombinieren. Wichtig ist das Finden qualitativ guter Cluster, welche eine möglichst große Anzahl des Alphabetes abdecken, wobei Überschneidungen der Cluster erlaubt bzw. sogar erwünscht sind.

Im Folgenden werden die Begriffe „abhängiges Alphabet“ oder „abhängige Zeichen“ verwendet, wenn damit ausgedrückt werden soll, dass die zugehörigen Subtraces durch einen Automaten beschrieben werden können, der nicht das Ergebnis des Produktes mehrerer Automaten ist. Somit identifiziert ein abhängiges Alphabet einen Teil des Traces, der einen einzelnen parallelen Handlungsstrang beschreibt.

4.1.2 Nutzbare Merkmale für Clustering-Verfahren

Clustering-Verfahren gruppieren Elemente so, dass in einer Gruppe Elemente mit möglichst ähnlichen Eigenschaften enthalten sind. Um dies zu erreichen, müssen für die entsprechenden Elemente zunächst die relevanten Eigenschaften festgelegt werden und für jede Eigenschaft muss ein Abstandsmaß definiert werden.

Das Ziel der Anwendung des Clustering-Verfahrens in dem hier vorliegenden Fall ist mit dem Finden von abhängigen Subalphabeten klar definiert. Daher müssen die entsprechenden Eigenschaften und Abstandsmetriken so gewählt werden, dass diese auch einen Rückschluss auf das Anwendungsziel zulassen.

Bei der Betrachtung der Events eines Traces gibt es a priori keine direkt verwertbaren Eigenschaften, welche für ein Clustering im Sinne der Aufgabenstellung in Frage kommen. Erschwerend kommt hinzu, dass Events nicht einzeln verglichen werden können, sondern ein Event in der Regel mehrmals in einem Referenztrace vorkommt. Somit müssen Eventfolgen verglichen werden.

Die Mehrzahl bekannter Clustering-Verfahren basiert auf der Nutzung einer großen Anzahl an Eigenschaften der zu gruppierenden Elemente. Daher müssen in der Regel einzelne dieser Eigenschaften explizit ausgewählt werden, um das Clustering-Verfahren anwenden zu können. Dieser Prozess wird Feature Selection genannt (Salem Alelyani, Jiliang Tang, Huan Liu 2013). Im vorliegenden Fall sind allerdings nur wenige bis gar keine direkt verwertbaren Eigenschaften vorhanden. Somit müssen Eigenschaften (sog. Features) quasi generiert werden. Diese eher selten angewendete Methode wird in der Literatur als Feature Construction oder auch Feature Extraction bezeichnet (Liu, Motoda 1998 ; Guyon 2006).

Für die Konstruktion von Eigenschaften wird meist auf ein Verhaltens- oder Umgebungsmodell der zugrunde liegenden Daten zurückgegriffen. Mit der Hilfe eines derartigen Modells ist es möglich, die aufgezeichneten Eigenschaften entweder umzurechnen oder auch neue Eigenschaften zu generieren, welche besser auf das Clustering-Problem anwendbar sind.

Bezogen auf die vorliegende Problemstellung kommen für diese Art der Feature Construction zwei Eigenschaften in Betracht: (i) das zeitliche Verhalten der Events und (ii) die informationstheoretischen Eigenschaften des Events. Auf die Verarbeitung beider Eigenschaften wird in den folgenden Kapiteln näher eingegangen.

Das zeitliche Verhalten eines Events ergibt sich aus den Auftretenszeiten des einzelnen Events im Netzwerktrace. Eine Auftretenszeit wird durch einen sog. Zeitstempel dokumentiert. Informationstheoretische Eigenschaften sind etwas schwieriger direkt zu benennen. Im Wesentlichen sind damit aber Eigenschaften gemeint, welche sich aus den Grundlagen der Kanalcodierung nach (Shannon 1948) und der algorithmischen Komplexität nach (Solomonoff 1964) ergeben.

4.2 Evaluierungsmethodik

Ein Clustering-Verfahren für die hier gestellte Problemstellung muss potenziell unabhängige Subalphabete mit zugehörigen Subtraces identifizieren. Dabei müssen die jeweiligen zugrunde liegenden Automaten nur noch nach Gleichung 2-14 (Produkt) kombinierbar sein, um das Verhalten des Referenztraces zu beschreiben. Die Bewertung der Clustering-Ergebnisse ist nicht ohne weiteres möglich, da die beschreibenden Automaten nicht bekannt sind. Demzufolge ist auch die Einhaltung der Gleichung 2-14 nicht direkt überprüfbar.

Somit ist keine direkte Überprüfung der Clustering-Ergebnisse in Bezug auf ihre Qualität möglich. Wie in Kapitel 4.1 bereits erwähnt, sind die Ergebnisse des Clustering-Verfahrens als heuristisch anzusehen. Das bedeutet, es wird nicht die optimale Lösung gesucht, sondern eine möglichst gute Lösung (eine optimale Lösung würde Gleichung 2-14 entsprechen). Eine möglichst gute Lösung muss einen möglichst großen Anteil des Alphabetes in möglichst unabhängige Alphabete gruppieren.

Zur Evaluierung der Clustering-Ergebnisse wird daher eine Methodik benötigt, mit der es möglich ist zu bestimmen, wie gut die Clustering-Ergebnisse in Bezug auf die Identifikation paralleler Handlungsstränge sind. Die Schwierigkeit bei der Evaluierung der Clustering-Ergebnisse besteht darin, dass die entsprechenden Automaten, welche das Referenztrace beschreiben können, a priori nicht bekannt sind. Somit muss ein Verfahren eingesetzt werden, welches es ermöglicht, Rückschlüsse auf mögliche Eigenschaften der zugrunde liegenden Automaten eines Referenztraces zu ziehen. Entsprechende Verfahren zum Erlernen von Abhängigkeitsmodellen bzw. Automaten wurden bereits bei der Vorstellung der Lernverfahren in Kapitel 3 präsentiert.

Der in Kapitel 3.3.3 beschriebene L^*-Algorithmus bringt für eine derartige Auswertung die besten Voraussetzungen mit, da er für diesen Zweck sehr gut auswertbare Automaten (DEA) erlernt. Die von L^* erlernten Automaten können als Hypothese für den zugrunde liegenden Automaten eines erstellten Clusters genutzt werden und so eine Bewertung der Clustering-Ergebnisse ermöglichen.

Es ist allerdings auch mittels des L^*-Algorithmus nicht möglich den resultierenden Automaten A^{**} zu erlernen. Somit kann, selbst wenn alle Subautomaten A^* genau bekannt sind, die Einhaltung der Gleichung 2-14 nicht direkt ermittelt werden. Auch in diesem Fall ist es notwendig, das Ziel des Clustering-Verfahrens, die Reduzierung der Komplexität der Systembeschreibung, indirekt zu bewerten.

Diese indirekte Bewertung wird in diesem Fall über die resultierende Komplexität der erlernten Automaten erstellt. Die Komplexität ergibt sich naheliegend aus der Struktur der erlernten Automaten mittels L^*. Im Folgenden wird eine Definition hierfür erarbeitet.

4.2.1 Komplexitätsmaß erlernter Automaten

Es gibt sehr unterschiedliche Ansätze, die Komplexität eines Automaten zu beschreiben. Häufig wird dies vom Gesichtspunkt der Softwareentwicklung getan. Somit spielen dort die Anzahl der Zustände und die Anzahl der Verzweigungen im Automaten eine wichtige Rolle. Die hier zu bewertende Komplexität soll keine Aussage über die Verständlichkeit eines Automaten geben, sondern vielmehr eine Aussage über die informationstheoretische Komplexität des Automaten zulassen.

In Bezug auf das beschriebene Lernproblem aus Kapitel 3 wird aus den dort gewonnenen Ergebnissen ein eigenes Komplexitätsmaß definiert. Es hat sich gezeigt, dass es in Bezug auf die Erlernbarkeit eine starke Signifikanz aus dem Verhältnis von Transitionen und Zuständen gibt. Dies resultiert vor allem daraus, dass es bei einer größeren Anzahl von Transitionen pro Zustand auch eine größere Anzahl von erzeugbaren Zeichenkombinationen geben kann.

Die Komplexität eines Automaten wird daher für diese Arbeit definiert durch die Anzahl an Transitionen pro Zustand eines Automaten. Die Definition des Komplexitätsmaßes eines DEA beschreibt Gleichung 4-1.

Definition Komplexitätsmaß eines DEA:

$$\mathbb{K} = \frac{|T|}{|Z|} \qquad 4\text{-}1$$

Die obere Grenze für $\mathbb{K}$ ist dabei die Alphabetgröße $|\Sigma|$, die untere Grenze ist 1. Die obere Grenze von $\mathbb{K} = |\Sigma|$ stellt die Komplexität dar, welche maximal bei dem Produkt von $A^{**} = A_1^* \times A_2^*$ entstehen kann. Der resultierende Automat ist dann ein Automat mit einem Zustand und $|\Sigma|$ Transitionen (Abbildung 3-8). Hat $\mathbb{K}$ einen Wert für einen Automaten an der oberen Grenze, so kann der Automat jedes beliebige Trace akzeptieren. Ist ein Automat in seiner Komplexität an der oberen Grenze, so ist dieser sehr wahrscheinlich das Produkt mehrerer unabhängiger Automaten.

Die untere Grenze von $\mathbb{K} = 1$ stellt einen Automaten ohne Verzweigung dar, wie er in Abbildung 3-7 zu sehen ist (ein Kreis bei Entfernung des Startzustandes). Dieser Automat bildet einen nicht verzweigten Handlungsablauf ab und kann genau eine Zeichenfolge der Länge $|T|$ akzeptieren.

Es ist davon auszugehen, dass Automaten mit hoher Komplexität sehr wahrscheinlich das Produkt unabhängig ausgeführter Automaten sind. Hingegen kann davon ausgegangen werden, dass bei kleineren Komplexitäten die Wahrscheinlichkeit zunimmt, nicht mehr „teilbare“ Automaten vorzufinden.

4.2.2 Anwendung des L^*-Algorithmus zur Evaluierung

Wie in Kapitel 3.3.3 erläutert ist es mittels des L^* möglich, aus einem Trace beliebiger Länge einen Automaten zu erlernen, der zumindest das gegebene Trace bis zu einer gegebenen maximalen Testlänge vollständig akzeptiert. Im Falle der Evaluierung von Clustering-Ergebnissen wird von jedem Subtrace ein einzelner Automat gelernt.

Die hier vorgestellte Evaluierungsmethode nutzt die Komplexität $\mathbb{K}$ der erlernten Automaten. So sollten Automaten mit einer Komplexität nahe dem Maximum verworfen werden, da diese offensichtlich nicht dem Ziel der Verringerung der Komplexität gerecht werden. Eine weitergehende Bewertung der Clustering-Ergebnisse auf Basis der Komplexität kann nur in Relation zu anderen Automaten vorgenommen werden. Das Vorgeben fester Grenzen ist schwierig, da dies sehr stark vom tatsächlichen Systemverhalten abhängt. Im Folgenden wird daher eine Methode erläutert, welche eine differenzierte Bewertung der Automaten zulässt.

Die Evaluierung des L^*-Algorithmus in Kapitel 3.3.3 hat gezeigt, dass der Algorithmus für Automaten mit niedriger Komplexität sehr schnell sehr gute Hypothesen für die zugrunde liegenden Automaten aufstellen kann. Somit kann auch die benötigte Lernzeit als Indiz für abhängige Alphabete und deren Komplexität dienen. Durch die schnelle Konvergenz des Algorithmus bei abhängigen Alphabeten ergibt sich eine fast schon exponentiell steigende Rechenzeit im Verhältnis zur Komplexität der Automaten. Daher wurde für die Evaluierung eine feste Zeitschranke eingeführt, nach der der L^*-Algorithmus abgebrochen wird, wenn er nicht schon früher selbst terminiert ist. Die Zeitschranke muss abhängig von der Rechenleistung und der Länge des Referenztraces gewählt werden. Für das vorliegende Referenztrace mit einer Aufzeichnungsdauer von ca. 12 Minuten, wurde eine Zeitschranke von 3 Minuten gewählt.

Zusätzlich zur Komplexität des gelernten Automaten ergibt sich somit als Evaluierungskriterium auch die Zeit, die L^* benötigt, um den Automaten zu erlernen, oder auch die Information, dass L^* innerhalb der Zeitschranke keinen Automaten erlernen konnte.

Ein weiteres Merkmal, welches nach der Ausführung des L^* genutzt werden kann, um die Clustering-Ergebnisse zu evaluieren, ist die maximal getestete Wortlänge. Hierbei gibt es zwei Möglichkeiten: (i) die Ausführung von L^* immer bis zur Zeitgrenze und dann die maximal getestete Wortlänge als Gütekriterium verwenden und (ii) die gleichzeitige Verwendung der maximal getesteten Wortlänge als Abbruch- und Gütekriterium.

Während der durchgeführten Experimente mit realen Datensätzen hat sich gezeigt, dass sich die Verwendung einer kleinen maximalen Wortlänge sehr günstig auf die benötigte Zeit zur Evaluierung auswirkt. Dabei wird die Aussagekraft der gelernten Automaten in Bezug auf die hier durchzuführende Evaluierung nicht signifikant verringert, wenn die maximale Wortlänge so gewählt wird, dass ein Großteil der Lernaufgaben sehr schnell (innerhalb weniger Sekunden) beendet wird.

Konkret haben die durchgeführten Versuche gezeigt, dass eine Zeitgrenze von wenigen Minuten (3 bis 5 min) für Wortlängen von fünf bis sieben Zeichen als guter Wert angesehen werden kann. Nur ein sehr geringer Prozentsatz (ca. 5 %) der Lernvorgänge, die nach dem Erreichen der Zeitschranke abgebrochen wurden, konnte nach längerer Lernzeit (hier 20 min) die entsprechende Wortlänge noch erreichen und einen gültigen Automaten lernen.

4.2.3 Datensätze für Evaluierung

Zur Evaluierung der entwickelten Clustering-Verfahren werden Datensätze eines realen Fahrzeug-CAN-Busses verwendet (Kapitel 3.4). Als Referenzdatensatz wird ein Netzwerktrace mit einer Dauer von ca. 13 Minuten verwendet. Das resultierende

Referenztrace weist insgesamt ca. 7.500 verschiedene Events (Zeichen) auf und hat eine Länge von insgesamt ca. $2{,}5 \cdot 10^6$ Zeichen.

4.2.4 Nutzung als Bewertungsverfahren für Auswahl der Cluster zur OoN-Erkennung

Die im Kapitel 4.2 vorgestellte Evaluierungsmethodik war ursprünglich dazu gedacht, die Ergebnisse einzelner Clustering-Verfahren miteinander vergleichen zu können. Bei der Anwendung der Clustering-Ergebnisse im selbstlernenden OoN-Erkennungsverfahren hat sich jedoch gezeigt, dass entschieden werden muss, welche Cluster eine gute Qualität besitzen und für die OoN-Erkennung eingesetzt werden können.

Da dieser Vorgang sehr ähnlich zur gezeigten Evaluierungsmethodik ist, werden die vorgestellten Evaluierungskriterien auch als Bewertungskriterien zur Auswahl guter Cluster für die Verwendung im selbstlernenden OoN-Erkennungsverfahren genutzt. In diesem Zusammenhang wird aus der Evaluierungsmethodik ein Bewertungsverfahren, welches Teil des selbstlernenden OoN-Erkennungsverfahrens ist.

Mit Hilfe des Bewertungsverfahrens werden nur die Subalphabete für das selbstlernende OoN-Erkennungsverfahren genutzt, welche die Forderung nach abhängigen Alphabeten erfüllen. Als Hauptkriterien für die Selektion der Clustering-Ergebnisse werden das Erreichen der maximalen Komplexität und das Lernen eines Automaten innerhalb der Zeitgrenze herangezogen.

4.3 Clustering auf Basis des zeitlichen Verhaltens

Eine sehr häufig genutzte Methode zum Clustering großer Datenmengen ist das Sortieren nach der Häufigkeit des jeweiligen Ereignisses. Häufig wird in diesem Zusammenhang von der Frequenz des Ereignisses gesprochen. In vielen Fällen ist auch von einer statistischen Analyse des zeitlichen Verhaltens oder von Auftretenshäufigkeiten die Rede. Die grundlegende Annahme liegt darin, dass Ereignisse mit einer ähnlichen Häufigkeit mit großer Wahrscheinlichkeit aus der gleichen Ursache resultieren.

Diese These ist auch für die Analyse von Computerprogrammen anwendbar. Die Auswertung von Log-Dateien oder Debug-Traces mittels einer statistischen Analyse, um mögliche Ausführungspfade zu extrahieren, wird z. B. in (Nickolayev, Roth, Reed 1997) oder (Kranzmüller 2000) beschrieben. Zudem wird eine Analyse von Programmcodes anhand von Aufruffrequenzen zum besseren Programmverständnis oder zur Erkennung von böswilligen Änderungen des Programms von einigen Autoren vorgeschlagen (z. B. Ball 1999, Hangal, Lam 2002).

Es liegt daher nahe, dass abhängige Alphabete auch ein ähnliches zeitliches Verhalten aufweisen. Deshalb wird im Folgenden die Möglichkeit untersucht, Referenztraces mit Hilfe des zeitlichen Verhaltens der Events in abhängige Subalphabete zu gruppieren.

Die Eigenschaften für das zeitliche Verhalten werden jeweils pro Zeichen (d. h. ein Event mit einer eindeutigen ID) des Alphabetes erstellt. Ein zeitlich markiertes Event ε ergibt sich aus dem Event E und dem zugehörigen Zeitstempel t nach Definition 4-2.

Definition zeitlich markiertes Event

$$\varepsilon_i = (e, t_i) \text{ mit } e \in E \quad (4\text{-}2)$$

Ein zeitlich markiertes Trace φ ergibt sich aus der Folge von zeitlich markierten Events.

Definition zeitlich markiertes Trace

$$\varphi = \varepsilon_1, \varepsilon_2 \dots \varepsilon_n \quad (4\text{-}3)$$

Für das zeitliche Verhalten eines einzelnen Events werden aus dem Trace die Zeitstempel dieses Events herangezogen. Somit ergibt sich für jedes Event eine Folge von Auftretenszeitpunkten nach Gleichung 4-4. Wird diese Folge von Zeitstempeln auf der Zeitachse dargestellt, so ergibt sich der zeitliche Verlauf eines einzelnen Events.

Zeitlicher Verlauf eines Events

$$e \rightarrow t_1, t_2 \dots t_n \quad (4\text{-}4)$$

4.3.1 Clustering auf Basis der Frequenz

Wie einleitend bereits beschrieben, ist die statistische Analyse des zeitlichen Verhaltens eines der einfachsten Mittel, um eventuelle Abhängigkeiten im Referenztrace aufzuzeigen. Im Grunde werden mittlere Frequenzen der Events bestimmt und Events mit ähnlichen mittleren Frequenzen werden der gleichen Gruppe zugeordnet.

Die mittlere Frequenz kann aus dem Mittelwert der Differenzen der Auftretenszeiten eines Events berechnet werden (Gleichung 4-5).

Mittlere Frequenz eines Events

$$\overline{t_a} = \frac{\sum_{i=1}^{n-1}(t_{i+1} - t_i)}{n} \quad (4\text{-}5)$$

Da ein Event nicht zwangsläufig zeitlich gleich verteilt vorkommen muss, kann auch der Erwartungswert der mittleren Zeit zwischen zwei Events verwendet werden.

Es erscheint weiterhin nicht sinnvoll, Events, deren Auftretenszeiten zufällig oder auch sehr ungleichmäßig sind, mit zyklisch auftretenden Events zu vergleichen. Daher ist das Vorhandensein einer gewissen Periodizität im Auftreten des Events ein weiteres Kriterium für diese Art des Clusterings.

Die Periodizität eines Events kann durch den Varianzkoeffizienten c_v der Auftretenszeiten beschrieben werden (Gleichung 4-6).

Varianzkoeffizient eines Events

$$c_v = \frac{Standardabweichung(\overline{t_a})}{Erwartungswert(\overline{t_a})} \quad (4\text{-}6)$$

Der Varianzkoeffizient ist in diesem Fall ein Maß für die Streuung der Zeitwerte zwischen den Events. Ein kleinerer Varianzkoeffizient entspricht dabei einer kleineren Streuung.

Zur Evaluierung der Anwendbarkeit des Clusterings auf Basis der Frequenz wurden die Events des verwendeten Referenztraces (Kapitel 4.2.3) zunächst auf ihre Periodizität überprüft. Bei der Anwendung eines c_v von 0,2 (d. h. 20 % Streuung) konnten lediglich 6 % der Events des Referenztraces als nennenswert periodisch eingestuft werden.

Aufgrund dieses sehr niedrigen Anteils periodischer Events scheint Clustering auf Basis der Frequenz für die gegebene Problemstellung nicht Erfolg versprechend und wird daher nicht weiter untersucht.

4.3.2 Clustering auf Basis des Frequenzspektrums

Wie im vorhergehenden Kapitel gezeigt wurde, genügt es für den Großteil der Events des Referenzdatensatzes nicht, die mittlere Frequenz eines Events zu verwenden, um sinnvolle Clustering-Ergebnisse zu erhalten. Entweder die Hypothese der Abhängigkeit im zeitlichen Bereich ist nur sehr selten zutreffend oder die Abhängigkeiten gestalten sich komplexer, als dies eine einzelne Frequenz abbilden kann.

Bei einer Analyse des Auftretens einzelner Events in einem Trace mittels eines Histogramms ist gut erkennbar, dass Events zwar in gewisser Regelmäßigkeit, aber nicht gleich verteilt über den Beobachtungszeitraum auftreten. Allerdings zeigt sich auch, dass die Darstellung der Peaks in einem Histogramm sehr von der gewählten Klassenbreite abhängt. Das macht auch die direkte Verwendung der Histogrammdarstellung für ein Clustering schwierig (Abbildung 4-2).

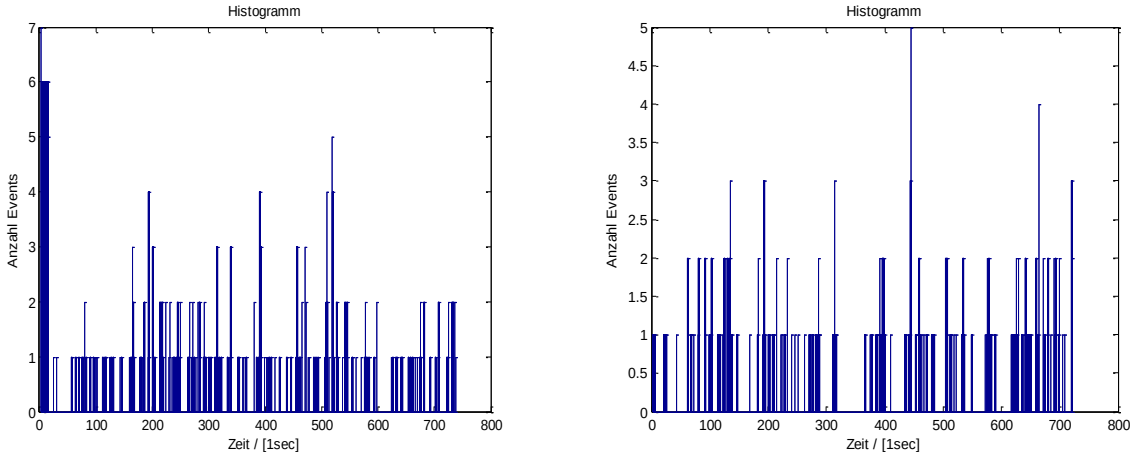

Abbildung 4-2 Beispiel zweier Histogramme von Events des Referenztraces

Um das komplexere zeitliche Verhalten der Events eines Traces für Clustering-Verfahren zugänglich zu machen, wird im nächsten Kapitel zunächst eine erweiterte Hypothese über das zu erwartende Systemverhalten aufgestellt. Mit Hilfe dieser Hypothese wird dann im nächsten Schritt ein Verfahren gezeigt, welches aus dem zeitlichen Verlauf eines Events entsprechende Eigenschaften generieren kann, die für ein Clustering-Verfahren verwertbar sind. Einige der hier gezeigten Erkenntnisse bzw. Ergebnisse wurden vom Autor selbst in (Langer, Oswald 2014 a) beschrieben.

4.3.2.1 Erweiterung Systemhypothese

In Kapitel 2.2 wurde einleitend die Hypothese aufgestellt, dass die Events des Traces durch mehrere, parallel ausgeführte diskrete endliche Automaten (DEA) abgebildet werden können (Gleichung 2-14). Die dazu vorgeschlagenen Automaten nach Gleichung 2-2 f. besitzen keine Möglichkeit, das zeitliche Verhalten abzubilden. Um das zeitliche Verhalten des Traces erklären zu können, muss diese Hypothese erweitert werden.

Als geeignetes Darstellungsmittel für derartige Probleme bietet sich die Theorie der Zeitautomaten (engl. Timed Automata) nach (Alur, Dill 1994) an. Als Erweiterung zum DEA besitzt hier jeder Automat eine beliebige Anzahl an Uhren und die Transitionen des Automaten werden jeweils mit einem Guard (Ausführungsbedingung) versehen. Ein Guard prüft vor jedem Zustandsübergang, ob die jeweilige Uhr einen entsprechend vorgesehen Wert hat und der Zustandsübergang durchgeführt werden darf.

Definition eines Zeitautomaten (ZDEA) : 4-7

Der Akzeptorzeitautomat ist definiert durch das 6-Tupel $A_t := (\Sigma, Z, z_0, X, \phi, F)$, wobei

- Σ das Eingabealphabet, bestehend aus Zeichen $s \in \Sigma$, ist,
- Z die Menge an Zuständen, welche A annehmen kann, ist,
- z_0 der Startzustand von A; $z_0 \in Z$ ist,
- X eine endliche Menge von Uhren $x \in X$ ist,
- ϕ die Übergangsfunktion zwischen den Zuständen mit $\phi: \Sigma \times Z \times G(X) \mapsto Z$ ist, wobei $G(X)$ den zeitlichen Guard der Transition beschreibt und
- F die Menge der finalen Zustände $F \subseteq Z$ ist.

Mit dieser Definition eines ZDEA lässt sich das zeitliche Verhalten von eventbasierten Systemen modellieren. Um Rückschlüsse aus dem zeitlichen Verhalten eines Referenztraces auf die Zugehörigkeit eines Events zu einem bestimmten Automaten zu ziehen, ist es allerdings notwendig, diese sehr allgemeine Form der Darstellung etwas zu vereinfachen. Diese Vereinfachung dient im Weiteren dazu, ein einfaches Systemmodell aufzustellen, mit dessen Hilfe es möglich ist, aus dem zeitlichen Verhalten eines Events weitere Eigenschaften zu extrahieren. Diese Eigenschaften sollten dann für ein Clustering-Verfahren eingesetzt werden können.

Echtzeitsysteme weisen in der Regel ein zeitlich konstantes bzw. vorhersagbares Verhalten auf. Daher wird im Folgenden davon ausgegangen, dass eine Transition nach einer fest vorgegebenen Zeitspanne ausgeführt wird. Werden in der einfachen Version eines solchen Automaten für jede Transition ein fester Zeitwert verwendet und die Uhr in jedem Zustand zurückgesetzt, so ergibt sich ein Automat, wie er beispielhaft in Abbildung 4-3 dargestellt ist.

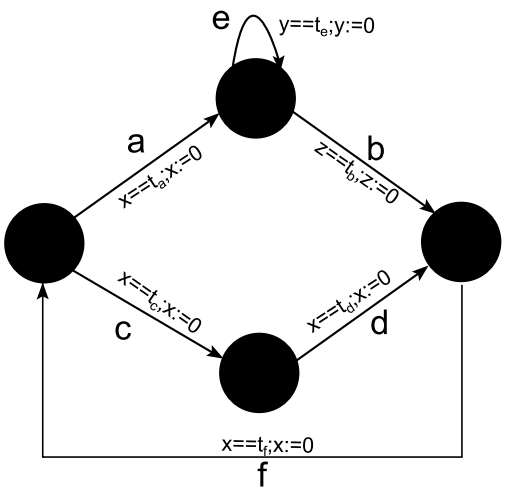

Abbildung 4-3 Vereinfachte Form eines ZDEA mit fixen Zeitgrenzen

Wird diese einfache Form der Übergangsfunktion bzw. des Guards zur Modellierung des zeitlichen Verhaltens des Systems zugrunde gelegt, so können, ausgehend vom modellierbaren Verhalten, folgende charakteristische Eigenschaften im Zeitverlauf eines Events vorhergesagt werden.

Die Zeitdauer zwischen dem wiederholten Auftreten eines bestimmten Events ergibt sich aus der Summe der Zeiten der Transitionen auf dem Pfad durch den Automaten. Da es verschiedene Pfade zwischen Transitionen mit dem gleichen Event geben kann, ergeben sich auch verschiedene mögliche Zeiten zwischen den Events. Durch Schleifen innerhalb des Automaten kann es zu zyklischen Wiederholungen eines Teilgraphen kommen, was in etwa der Realisierung von Schleifen eines Programmes entspricht. Ein Programm enthält in der Regel mehrerer geschachtelte Schleifen.

Bei Betrachtung des zeitlichen Verhaltens von geschachtelten Schleifen ergeben sich mehrere überlagerte Frequenzen, wobei die inneren Schleifen in der Regel eine höhere Eigenfrequenz aufweisen als die äußeren Schleifen. Die inneren Schleifen enthalten zusätzlich zu ihren eigenen Frequenzanteilen die Frequenzanteile der äußeren Schleifen. In einer groben Näherung könnte das Frequenzverhalten mit einer multiplikativen Überlagerung der einzelnen Frequenzen beschrieben werden (Abbildung 4-4). Eine multiplikative Überlagerung der Frequenzen wird hauptsächlich durch die Verwendung verschiedener Uhren verursacht. Eine innere Schleife kann so abrupt abgebrochen werden, wenn die Uhr einer angrenzenden Transition abgelaufen ist.

Beispiel im Anwendungskontext: Eine Funktion A erzeugt mit einer Frequenz von 0,1 Hz ein Event a. Durch Event a wird die Ausführung einer Funktion B gestartet und in Funktion B jeweils mehrmals kurz hintereinander eine Schleife ausgeführt, welche mit 0,5 Hz ein Event b generiert. In der Folge hat das Event b eine Eigenfrequenz von 0,5 Hz, durch die zyklische Aktivierung hat es aber auch einen Frequenzanteil von ~0,1 Hz, welcher ungefähr der Frequenz des auslösenden Events a entspricht.

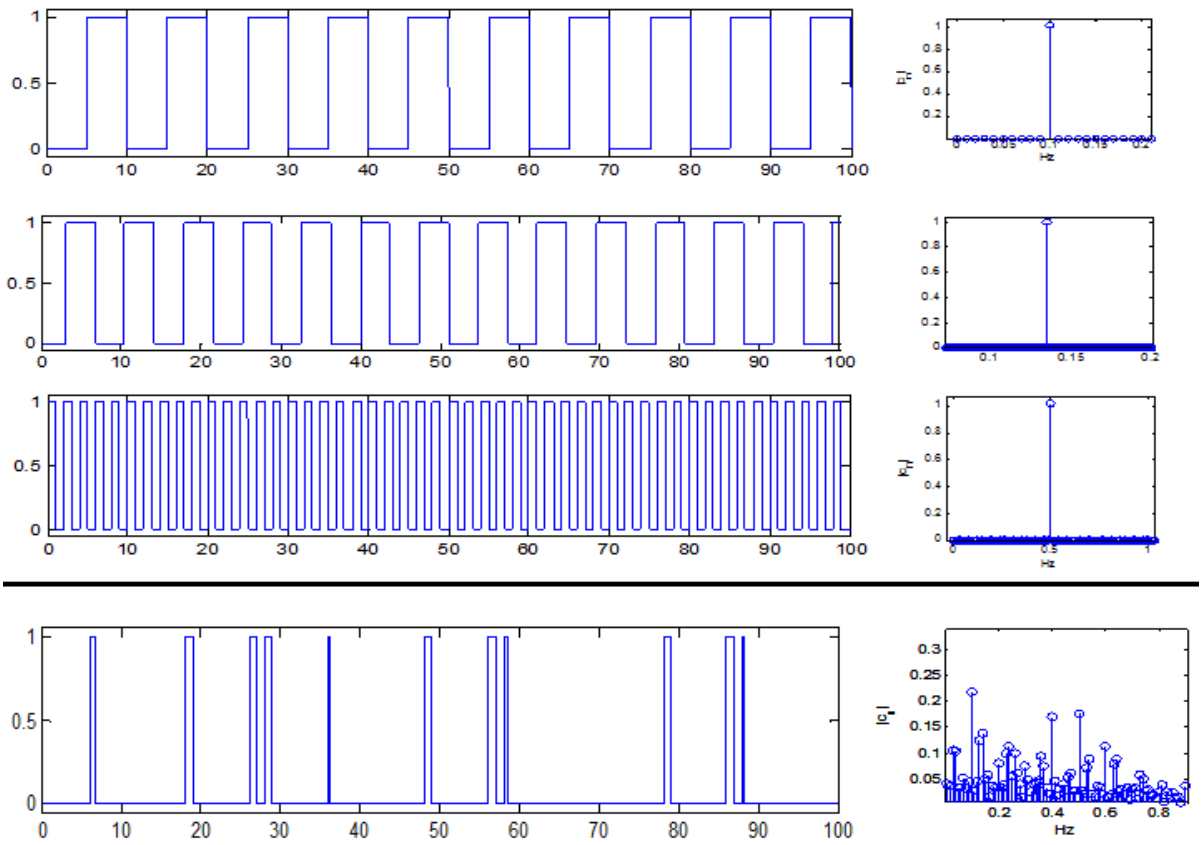

Abbildung 4-4 Simulation eines multiplikativen Frequenzverlaufes eines Events mit der Überlagerung von mehreren Aktivierungsfrequenzen (rechts: Frequenzgang, links: das zugehörige Frequenzspektrum)

Resultierend aus diesem Systemmodell ist davon auszugehen, dass im zeitlichen Verhalten von Events in vielen Fällen mehrere Frequenzanteile enthalten sind. Weiterhin kann davon ausgegangen werden, dass Events, welche dem gleichen ZDEA zuzuordnen sind, ähnliche oder gleiche Frequenzanteile aufweisen werden. Somit bilden die verschiedenen Frequenzanteile eines Events eine gute Basis für die Anwendung eines Clustering-Verfahrens.

Ist es demzufolge möglich, die Frequenzanteile im Zeitverhalten eines Events zu bestimmen, so können diese für die Berechnung der Ähnlichkeit oder des Abstandes in einem Clustering-Verfahren eingesetzt werden.

4.3.2.2 Realisierung von Clustering-Verfahren auf Basis des Frequenzsprektrums

Spektralanalyse eines Events

Im Folgenden wird eine Methode vorgestellt, die es erlaubt, aus einem Event und dessen Auftretenszeiten $e \rightarrow t_1, t_2 \ldots t_n$ die Frequenzanteile für ein Clustering-Verfahren zu erzeugen.

Eine einfache Möglichkeit, um Frequenzanteile eines Signals zu berechnen, ist die Fourier-Analyse. Unter der Annahme, dass alle Zeitpunkte $t_1, t_2 \ldots t_n$ den Null-Durchgang eines Sinus-Signales darstellen, lässt sich mittels der diskreten Fourier-Analyse ein Frequenzspektrum aus diesem Signal berechnen. Dabei stellt die zeitliche Länge des

Traces die Länge des Samples dar, welches analysiert werden muss. Da die Berechnung des resultierenden Sinus-Signals selbst schon die Kenntnis der entsprechenden Frequenzanteile voraussetzt, ist auch eine näherungsweise Berechnung der resultierenden Schwingung sehr aufwendig. Um den Aufwand für die Berechnung des resultierenden Signals niedrig zu halten, wird für die Fourier-Analyse eine Rechteckfunktion durch die Zeitpunkte $t_1, t_2 \dots t_n$ gelegt (Abbildung 4-5). Aus dieser Rechteckfunktion wird anschließend mittels einer Fast-Fourier-Transformation (FFT) ein entsprechendes Spektrum abgeleitet.

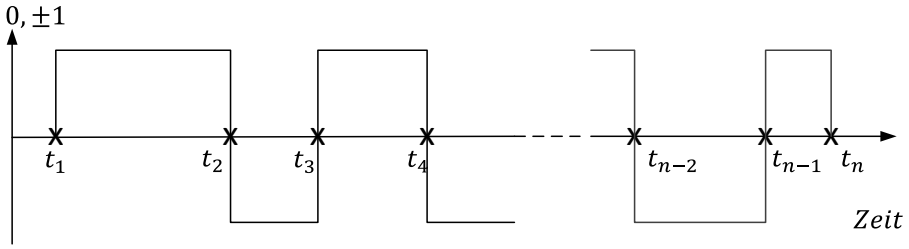

Abbildung 4-5 Rechteckfunktion mit Null-Durchgang bei den t_n eines Events e

Die FFT ist für die Analyse zeitdiskreter Signalverläufe ausgelegt und kann entsprechend jeweils ein Signalsample einer bestimmten Zeitdauer verarbeiten. Daher ist die erstellte Rechteckfunktion über die Länge bzw. Zeitdauer des Referenztraces als ein Signalsample anzusehen. Die Sample-Frequenz ergibt sich dabei aus der kleinsten Zeitspanne zwischen zwei Zeitstempeln. Nach dem Nyquist-Shannon-Abtasttheorem muss die Abtastfrequenz f_s größer als die doppelte größte Eigenfrequenz sein ($f_s > 2 \cdot f_{MAX}$).

Die mit Hilfe der FFT erstellten Spektren sind auf den ersten Blick sehr verrauscht (Abbildung 4-6, links). Dies ist hauptsächlich auf die Verwendung einer Rechteckfunktion als Eingangssignal und die nicht zwangsläufige Periodizität zurückzuführen.

Für die Verwendung in einem Clustering-Verfahren muss das Spektrum auf seine Hauptbestandteile reduziert werden. Aus diesem Grund wird eine weitere Filterung bzw. Aufarbeitung des Spektrums durchgeführt. Wichtig für die Aufarbeitung ist die Generierung eines charakteristischen Spektrums, nicht zwangsläufig die Extrahierung aller bzw. ausschließlich der Hauptfrequenzen. Die Methodik zur Aufarbeitung des Spektrums ist daher eher empirisch und weniger signaltechnisch begründet und wird im Folgenden kurz vorgestellt.

Das zu lösende Problem entspricht in etwa der Suche nach lokalen Maxima in verrauschten Daten. Einen Überblick und ähnliche Ansätze zur Aufbereitung derartiger Daten findet man z. B. in (Coombes u. a. 2005) und (Du, Kibbe, Lin 2006).

In einem ersten Schritt werden mittels der Bestimmung des Maximums in einem gleitenden Fenster naheliegende Maxima zusammengeführt (Abbildung 4-6, rechts). In einem zweiten Schritt wird die Matlab-Funktion *FindPeaks* angewandt. Diese vergleicht jeweils die Vorgänger und Nachfolger eines Datums. Sind diese niedriger als das untersuchte Datum, wird dieses als Peak ausgewählt. Dieser Schritt wird so lange ausgeführt, bis die Anzahl der gefundenen Peaks des resultierenden Spektrums weniger als acht, aber maximal drei Peaks aufweist. Das aufgearbeitete Frequenzspektrum eines

Events hat damit mehrere differenzierende Merkmale und kann im nächsten Schritt für eine Cluster-Analyse verwendet werden.

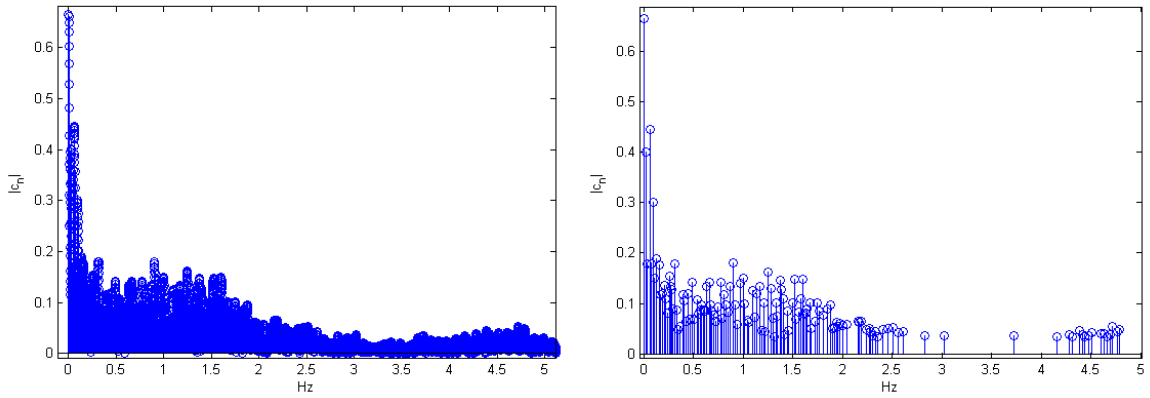

Abbildung 4-6 Ergebnis der FFT der Rechteckfunktion des Zeitverlaufes eines Events (links); Spektrum nach erstem Aufbereitungsschritt (rechts)

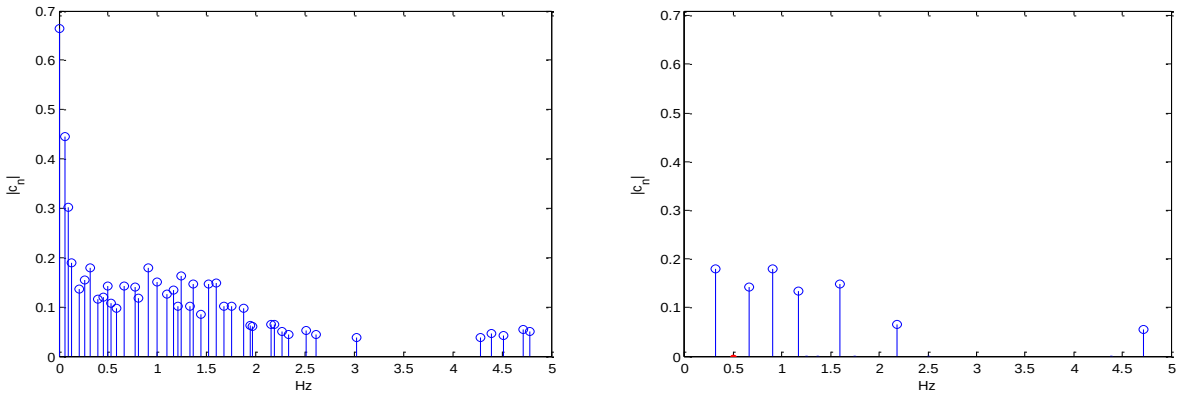

Abbildung 4-7 Aufbereitetes Spektrum nach dem ersten (links) und zweiten Durchlauf (rechts) der Matlab Funktion *FindPeaks*

Abstandsfunktion für Frequenzspektrum

Eine wichtige Voraussetzung für die Anwendung von Clustering-Verfahren ist die Bestimmung der Ähnlichkeit oder des Abstandes zweier Events (in diesem Fall deren Frequenzspektren). Das extrahierte Frequenzspektrum hat zwei Merkmale: (i) die jeweiligen Frequenzen und (ii) die zugehörigen Amplituden. Mit diesen Eigenschaften soll die Ähnlichkeit im Sinne der Systemhypothese eines zeitlichen Automaten (Gleichung 4-7) beschrieben werden.

Sehr häufig wird beim Vergleich des Spektrums die Frequenz mit der größten Amplitude als wichtigstes Merkmal angesehen, da bei einer harmonischen Schwingung diese Frequenz gewöhnlicherweise die Hauptfrequenz darstellt. Dies ist im vorliegenden Fall nicht anwendbar, da die Ausgangsdaten keine harmonische Schwingung darstellen.

Eine einfache Vermutung der Ähnlichkeit kann über den Vergleich der Frequenzanteile erstellt werden. Je mehr gleiche Frequenzanteile die Spektren zweier Events aufweisen, umso größer ist die Wahrscheinlichkeit, dass diese Events vom gleichen Automaten

erzeugt wurden. Für diese Vermutung hat die Größe der Amplitude keine Auswirkung auf die Ähnlichkeit. Inwieweit durch die Zugehörigkeit zum gleichen Automaten auch die Amplitude eines Spektrums miteinbezogen werden kann, lässt sich nicht zwangsläufig aus der Systemhypothese ableiten. Daher wird zunächst nur die Position der Frequenzanteile in die Berechnung des Ähnlichkeitsmaßes einbezogen.

Das Ähnlichkeitsmaß kann auch aus dem Abstand zweier Frequenzspektren interpretiert werden und wird im Folgenden als Abstandsmaß bezeichnet. Somit muss eine Berechnungsvorschrift für den Abstand der Frequenzanteile angegeben werden. Da die Anzahl der Frequenzanteile auf acht reduziert wurde, ergibt sich ein acht-dimensionaler Raum, in dem jedes Frequenzspektrum einen Punkt darstellt. Für die Bestimmung des Abstandes zwischen diesen Punkten können nun bekannte Abstandsmaße angewendet werden.

Das Abstandsmaß muss nicht zwangsläufig den geometrischen Abstand zwischen den Punkten darstellen. Die aufgestellte Systemhypothese lässt vermuten, dass das Vorkommen gleicher Frequenzanteile höher zu bewerten ist als der geometrische Abstand der Spektren. Fehlen z. B. einzelne Frequenzanteile in einem Eventspektrum, verschiebt dies die Lage des Spektrums im Raum signifikant. Dies sollte bei der Abstandsberechnung berücksichtigt werden.

Um diesen Effekt zu verhindern, wird ein eigenes Abstandsmaß eingeführt, welches das Vorkommen gleicher Frequenzanteile stärker bewertet. Dazu wird eine neue Abstandsfunktion definiert. Um das Fehlen einzelner Spektralanteile nicht zu stark zu gewichten, wird eine Bewertung des Abstandes nur anhand der übereinstimmenden Frequenzanteile $freq_{x,i}$ erstellt. Das neu definierte Abstandsmaß wird durch die Anzahl der übereinstimmenden Frequenzanteile bestimmt. Zu diesem Zweck wird als maximaler Abstand die maximale Anzahl der Frequenzanteile max_{freq} bestimmt. Besitzen zwei Spektren einen gleichen Frequenzanteil, wird jeweils ein Zähler vom maximalen Abstand abgezogen.

Zur Bestimmung der Gleichheit eines einzelnen Frequenzanteils wird dabei eine Toleranzschwelle ε eingeführt, um kleinere Ungenauigkeiten auszugleichen. Da nicht alle Spektren die gleiche Anzahl an Frequenzen aufweisen, werden die Spektren mit weniger Anteilen jeweils bis zur maximalen Anzahl mit leeren Frequenzen aufgefüllt. Den geringsten Abstand besitzen Spektren, deren Frequenzen komplett übereinstimmen. Den größten Abstand (max. Anzahl an Frequenzen) besitzen Spektren, die keine gemeinsamen Frequenzanteile aufweisen.

Definition Abstandsmaß d_{freq} (Frequenzabstand) für Vergleich zweier Spektren (Gleichung 4-9)

$$sum\_eqal\_freq_{x,y} = \sum_{|freq_{x,i} - freq_{y,j}| > \varepsilon} 1; \qquad i,j = 1, \dots, max_{freq} \qquad 4\text{-}8$$

Berechnungsvorschrift $d_{freq}(x,y) = max_{freq} - sum\_eqal\_freq_{x,y}$ 4-9

Maximum $\max(d_{freq}) = max_{freq}$ 4-10

Minimum $min(d_{freq}) = 0$ 4-11

Um einen Vergleich des neu eingeführten Abstandsmaßes mit bekannten Abstandsmaßen ziehen zu können, werden im Folgenden weitere bereits bekannte Abstandsmaße für den Vergleich zweier Spektren vorgestellt. Später werden diese Abstände einer Evaluierung im Hinblick auf ihre Eignung im Clustering-Verfahren unterzogen.

Die bereits bekannten und zur Evaluierung herangezogenen Abstände sind:

(i) der euklidische Abstand mit:

$$d_E(x,y) = \sqrt{\sum_{i=1}^{max_{freq}} (freq_{x,i} - freq_{y,i})^2} \quad 4\text{-}12$$

(ii) der Manhattan-Abstand mit:

$$d_M(x,y) = \sum_{i=1}^{max_{freq}} |freq_{x,i} - freq_{y,i}| \quad 4\text{-}13$$

(iii) der Hamming-Abstand (sehr ähnlich zu d_{freq}, aber mit Berücksichtigung der Reihenfolge) mit:

$$d_H = \sum_{|freq_{x,i} - freq_{y,i}| > \varepsilon} 1 \,; i = 1, \dots, max_{freq} \quad 4\text{-}14$$

Verwendeter Clustering-Algorithmus

Auf Basis des generierten Frequenzspektrums (Kapitel 4.3.2.2) und der beschriebenen Abstandsfunktionen (euklidische-, Manhattan-, Hamming- und Frequenz-Abstand) können nun bekannte Clustering-Algorithmen angewendet werden. Bei der Auswahl der Clustering-Algorithmen muss berücksichtigt werden, dass die Anzahl der Cluster nicht bekannt ist. Weiterhin darf die maximale Anzahl von 127 Events pro Cluster nicht überschritten werden. Die Einschränkung auf 127 Events pro Cluster rührt aus der Verwendung des L^*-Algorithmus zur Evaluierung der Clustering-Ergebnisse, da die eingesetzte Implementierung dieses Algorithmus eine maximale Anzahl von 127 Zeichen erlaubt.

Zur Evaluierung des frequenzspektrumbasierten Clustering-Verfahrens und der zugehörigen Abstandsmaße wird ein hierarchischer Clustering-Algorithmus gewählt. Dies ist zwar nicht der schnellste Algorithmus, er bietet aber die Möglichkeit, sehr einfach und schnell die Anzahl der Cluster und die maximale Alphabetgröße pro Cluster zu variieren.

Alternativ könnten z. B. DBSCAN (Ester u. a. 1996) oder auch der OPTICS-Algorithmus (Ankerst u. a. 1999) eingesetzt werden.

Bei einem hierarchischen Clustering wird der Cluster-Baum (Dendrogramm) auf einer bestimmten Höhe (Level bzw. Hierarchieebene) durchtrennt (Abbildung 4-8). Die entstehenden Sub-Bäume entsprechen dann den gebildeten Clustern. Ist die Verteilung (Dichte) der Events annähernd gleich und wird das Dendrogramm auf einer Ebene aufgetrennt, so sind auch die resultierenden Cluster in etwa gleich groß.

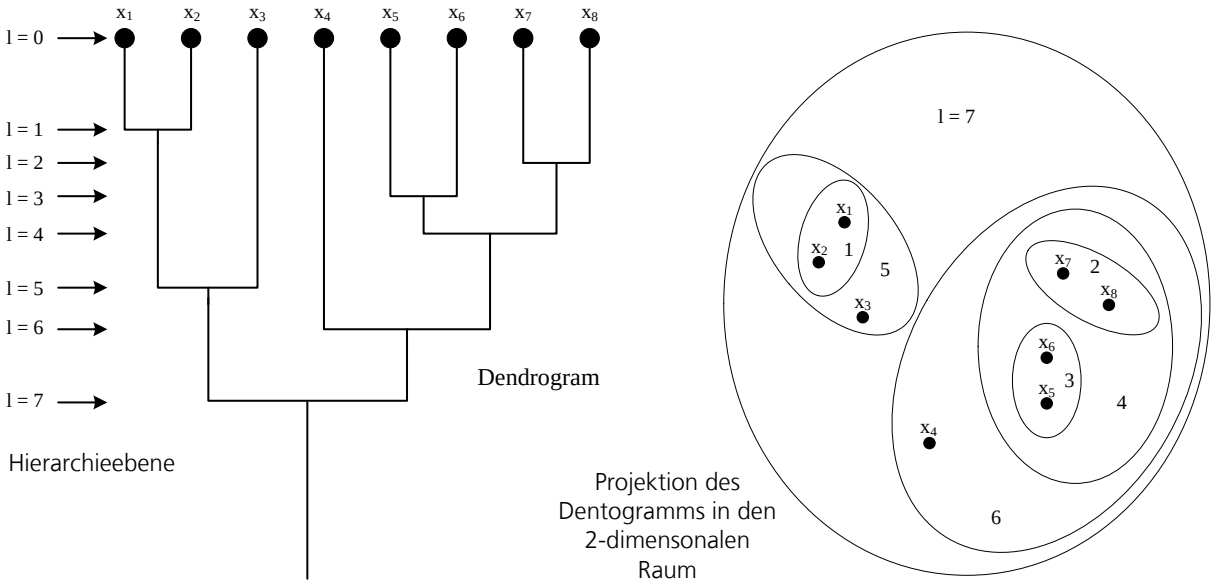

Abbildung 4-8 Darstellung eines Dendrogramms, welches die Grundlage für hierarchisches Clustering bildet (links), und die Projektion der verschiedenen Ebenen auf die zu clusternden Daten (rechts)

Bei den durchgeführten Versuchen zeigt sich aber, dass die Dichte der einzelnen Cluster sehr unterschiedlich ist. Daher muss der Cluster-Baum in unterschiedlichen Höhen aufgetrennt werden. Dies wird im vorliegenden Fall erreicht, indem Cluster, die eine Alphabetgröße von mehr als 127 Zeichen aufweisen, nochmals einer Cluster-Analyse unterzogen werden, bis die Alphabetgröße eines jeden Clusters weniger als 127 Zeichen aufweist.

4.3.2.3 Evaluierung für die Anwendung im selbstlernenden OoN-Erkennungsverfahren

Zur Evaluierung des vorgestellten Clustering-Verfahrens wurden die Referenzdaten, wie sie in Kapitel 3.4 bzw. 4.2.3 beschrieben wurden, herangezogen. Zu diesem Zweck wurde für alle im Referenztrace vorkommenden Events ein Frequenzspektrum erzeugt (Kapitel 4.3.2.2). Die maximale Anzahl von identifizierten Frequenzanteilen wurde auf 8 festgelegt ($max_{freq} = 8$).

Um eine Bewertung der in Kapitel 4.3.2.2 beschriebenen Abstandsfunktionen zu erhalten, wurden für jede Abstandsfunktion ein eigenes Dendrogramm und die dazugehörigen Cluster bestimmt. Durch sehr große Dichteunterschiede kann es vorkommen, dass einzelne Cluster nur aus einem Event bestehen. Diese Cluster bzw. diese Zeichen wurden verworfen, da aus einem Trace, welches nur aus einem Event

besteht, keine Automaten erlernt werden können. Eine Übersicht der Ergebnisse des Cluster-Verfahrens zeigt Tabelle 4-1.

Tabelle 4-1 Ergebnisse des hierarchischen Clusterings mit verschiedenen Abstandsfunktionen.

Abstandsfunktion	Anzahl der erstellten Cluster (Anzahl der Subalphabete)	Event-Coverage	Trace-Coverage	Durch-schnittliche Subalphabet-größe (Anzahl Zeichen)
Frequenzabstand	694	94 %	94 %	9,7
euklidischer Abstand	422	97 %	94 %	16,4
Manhattan-Abstand	469	97 %	93 %	14,7
Hamming-Abstand	570	96 %	96 %	12,6

Die in Tabelle 4-1 dargestellten Clustering-Ergebnisse sind im Sinne der Problemstellung nur wenig aussagekräftig, da sie noch nicht auf Abhängigkeiten hin bewertet wurden. Es zeigt sich aber, dass der eingeführte Frequenzabstand eine größere Anzahl an Clustern mit nur einem Zeichen generiert. Somit werden für den Frequenzabstand die meisten Zeichen verworfen. Als Folge ergibt sich bereits bei diesem Prozessschritt eine geringere Coverage.

Um die gefundenen Cluster auf ihre Abhängigkeit hin zu evaluieren, wird, wie in Kapitel 4.2 vorgeschlagen, auf jedes resultierende Subtrace der L^*-Algorithmus angewendet. Somit werden die resultierenden Cluster einer Evaluierung im Sinne der Problemstellung unterzogen.

Zur Evaluierung der erstellten Cluster wurde der L^*-Algorithmus nach einer maximalen Lernzeit von drei Minuten pro Cluster abgebrochen, wenn bis dahin kein Automat erlernt werden konnte. Ein erfolgreich erlernter Automat muss mindestens mit der maximal getesteten Wortlänge $n = 6$ erstellt worden sein und es muss die Äquivalenzfrage (Kapitel 3.3.3.1) bei $n = 6$ positiv beantwortet worden sein. Weiterhin darf die Komplexität $\mathbb{K}$ den Wert drei nicht überschreiten. Sind alle Bedingungen erfüllt, wird im Folgenden von einem lernbaren Automaten gesprochen. Ergibt ein Cluster einen lernbaren Automaten, wird davon ausgegangen, dass das geclusterte Subalphabet ein abhängiges Alphabet ist.

Es zeigt sich, dass bis auf sehr wenige Ausnahmen alle Automaten, welche innerhalb der gesetzten Zeit die maximal getestete Wortlänge von sechs erreichen, auch eine Komplexität von weniger als drei aufweisen. Die Ausnahmen (ca. 1 bis 2 %) sind durchweg Automaten, welche nur einen Zustand aufweisen (ähnlich des in Abbildung 3-8 dargestellten Automaten) und daher die maximale Komplexität besitzen.

Diese sind für eine OoN-Erkennung nicht verwendbar, da sie jedwedes zu prüfende Trace akzeptieren würden.

In Tabelle 4-2 sind die einzelnen Ergebnisse des Clustering-Verfahrens mit den unterschiedlichen Abstandsmaßen dargestellt. Es zeigt sich, dass mit dem Frequenzabstand zwar die meisten Cluster lernbare Automaten ergeben (506 bzw. 73 % der ursprünglichen Cluster), sowohl die Trace-Coverage als auch die Event-Coverage aber geringer als bei den alternativen Abstandsmaßen sind.

Die fünfte Zeile der Tabelle 4-2 beschreibt die Anzahl der Zeichen, welche nur durch die Anwendung dieses spezifischen Abstandsmaßes abhängigen Alphabeten zugeordnet werden konnten. Diese Eigenschaft ist vor allem im Hinblick auf die vorgestellte Optimierung in Kapitel 4.5 interessant, bei der mehrere Clustering-Verfahren kombiniert werden. Hier ist sehr gut sichtbar, dass insbesondere der eingeführte Frequenzabstand einen guten Beitrag zur Diversifizierung der Ergebnisse leistet und viele Zeichen, welche durch andere Abstandsmaße nicht zuordenbar sind, zu abhängigen Alphabeten mit lernbaren Automaten zuordnen kann.

Tabelle 4-2 Ergebnisse der Evaluierung der erstellten Cluster mittels des L^*-Algorithmus

	Frequenz-abstand	euklidischer Abstand	Manhattan-Abstand	Hamming-Abstand
Anzahl der lernbaren Cluster (Anteil der erstellten Cluster)	506 (73 %)	254 (60 %)	295 (63 %)	316 (55 %)
Durchschnittliche Komplexität $\mathbb{K}$	1,50	1,65	1,66	1,62
Event-Coverage | Anteil der geclusterten Subalphabete am Gesamtalphabet (absolute Anzahl der Zeichen)	37 % (2.689)	40 % (2.883)	46 % (3.298)	37 % (2.676)
Trace-Coverage | Anteil der Subtraces an Referenztrace (absolute Anzahl der Zeichen)	27 % (678.820)	36 % (917.442)	49 % (1.259.452)	32 % (805.294)
Anzahl der Alphabetzeichen, welche ausschließlich mit diesem Abstand geclustert wurden	511	162	351	478

4.3.2.4 Fazit

Die besten Ergebnisse des frequenzbasierten Clusterings konnten mit dem Manhattan-Abstand erzielt werden. Mit diesem konnten eine Event-Coverage von 46 % und eine Trace-Coverage von 49 % erzielt werden. Weiterhin ist festzuhalten, dass die Ergebnisse unter Verwendung des euklidischen Abstandes fast ausschließlich eine Untermenge der Ergebnisse des Manhattan-Abstandes sind. Der selbstdefinierte Frequenzabstand liefert

zwar nicht die besten Ergebnisse, unterscheidet sich aber in Bezug auf die gefundenen Subalphabete deutlich von den übrigen Abstandsmaßen, was vor allem im Hinblick auf die Optimierung in Kapitel 4.5 wichtig ist. Die Komplexität der erstellten Automaten ist für alle Abstandsmaße fast identisch, womit die Qualität der abhängigen Alphabete und somit der Cluster auch sehr ähnlich ist.

Bezüglich der Anwendbarkeit des Verfahrens muss zudem die gute Performance im Sinne der benötigten Rechenzeit hervorgehoben werden. Der aufwendigste Rechenschritt ist dabei die Erstellung der Spektren, danach folgt bezüglich des Aufwandes die Berechnung der Abstandsfunktionen für das hierarchische Clustering. Die komplette Aufarbeitung des Referenztraces mittels frequenzbasierten Clusterings ist auf einem derzeit handelsüblichen PC mit ca. 2 Stunden Rechenzeit verbunden.

4.4 Clustering auf Basis der Entropie

Entropiebasiertes Clustering kann zum Finden von Zusammenhängen in kategorischen Daten verwendet werden, also für Daten, welche nicht ohne weiteres mittels eines Abstandsmaßes vergleichbar sind. Der Ansatz besteht darin, die Daten so zu gruppieren, dass die resultierende Entropie minimal wird (Barbará, Li, Couto 2002 und Tao, Sheng, Mitsunori 2004). Teile der hier gezeigten Ergebnisse wurden vom Autor selbst in (Langer, Oswald 2014 b) vorgestellt.

4.4.1 Grundlagen des Verfahrens

Nach (Shannon 1948) bestimmt sich für eine zufällige Variable X die Entropie $H(X)$ gemäß Gleichung 4-15 und die Entropie von zwei Abhängigen Variablen X, Y gemäß Gleichung 4-16.

Entropie $H(X)$

$$H(X) = -\sum_{x \in S(X)} p(x)\log(p(x)) \qquad 4\text{-}15$$

wobei $p(x)$ die Wahrscheinlichkeit eines bestimmten Wertes von X darstellt

Verbund- bzw. gemeinsame Entropie von X und Y.

$$H(X,Y) = -\sum_{x \in S(X), y \in S(Y)} p(x,y)\log(p(x,y)) \qquad 4\text{-}16$$

Das Ziel des entropiebasierten Clusterings ist es, die Cluster X und Y so zusammenzustellen, das $H(X) + H(Y)$ möglichst klein wird und somit X und Y möglichst unabhängig voneinander sind.

Wie sich aus Gleichung 4-17 ablesen lässt, ist das entropiebasierte Clustering, wie es hier verwendet werden soll, eine Optimierungsaufgabe. Diese besteht für das hier vorliegende Problem darin, die einzelnen Zeichen des Alphabetes Σ so in Cluster (Subalphabete) X_i zu gruppieren, dass die Summe der Entropie alle Cluster $H(X^*)$ möglichst klein ist (Gleichung 4-18).

Zu optimierende Funktion der Summe der Entropie zweier Teilsysteme

$$H(X,Y) \leq H(X) + H(Y) \quad \text{4-17}$$

Minimum von $H(X) + H(Y)$, wenn X und Y unabhängig sind.

Entropie des Systems als Summe der Teilsysteme (Verallgemeinerung der Gleichung 4-17)

$$H(X^*) = H(X_1) + H(X_2) + \cdots + H(X_n) \quad \text{4-18}$$

mit

$$X_i \subset \Sigma, X_i \cap X_j = \emptyset$$

Dabei ist sowohl die Größe der Cluster $|X_i|$ als auch die Anzahl n der Cluster nicht bekannt.

Die Lösung der Optimierungsaufgabe ist nach (Barbará, Li, Couto 2002) NP-schwer. So wird beispielsweise in (Tao, Sheng, Mitsunori 2004) ein Monte-Carlo-Verfahren zum Finden der optimalen Verteilung vorgeschlagen. Dieser Ansatz ist allerdings aufgrund der fehlenden Anwendbarkeit der Bewertungsfunktion für das hier vorliegende Problem nicht anwendbar.

4.4.1.1 Bewertungsfunktion für Cluster

Obwohl die Berechnung der Entropie nach Gleichung 4-15 zunächst recht simpel erscheint, ist die Anwendung für die hier gegebene Problemstellung nicht ohne Anpassung möglich.

Bereits (Barbará, Li, Couto 2002) nutzen als Entropiemaß die Minimum Description Length (MDL) für ihren Clustering-Algorithmus. Die MDL wurde von (Rissanen 1983) definiert und kann nach (Wallace, Dowe 1999) als Beschreibungsmaß für die Komplexität einer Nachricht verwendet werden. In ihren Grundzügen baut die MDL auf der Theorie nach Shannon auf.

Im Unterschied zu (Barbará, Li, Couto 2002) ist die MDL für das hier vorgestellte Problem nicht zwingend anwendbar, da mit ihr Subtraces unterschiedlicher Länge nicht vergleichbar sind. Als sehr ähnlich zur MDL beschreibt (Wallace, Dowe 1999) die Kolmogorov-Komplexität. Diese wird im Wesentlichen begründet durch die Theorie nach (Kolmogorov 1965) zur algorithmischen Beschreibung des quantitativen Informationsgehaltes und die Theorie nach (Solomonoff 1964) über algorithmische Wahrscheinlichkeit zur Vorhersage von Zeichenketten. Aus diesen Arbeiten ist in der Literatur der im Weiteren verwendete Begriff Kolmogorov-Komplexität entstanden.

Die Kolmogorov-Komplexität geht im Unterschied zur Entropiedefinition nach (Shannon 1948) nicht von der Wahrscheinlichkeit eines Zeichens oder eines Wortes aus. Vielmehr definiert sich die Komplexität nach Kolmogorov aus der Länge des Programms (oder auch Größe der Turing-Maschine), welche das zu prüfende Trace beschreiben würde. Den umgekehrten Weg geht (Solomonoff 1964); dieser beschreibt, wie durch

induktives Schließen aus einer Zeichenkette Zusammenhänge erschlossen werden können (Rathmanner, Hutter 2011). Aufgrund der Ähnlichkeit zur Definition der Entropie nach (Shannon 1948) wird die Kolmogorov-Komplexität auch mitunter algorithmische Entropie genannt.

Die Problemstellung der Prüfung eines Traces mittels eines Automaten ist in ihrem Wesen sehr ähnlich der Theorie nach (Solomonoff 1964) und (Kolmogorov 1965). Jedoch ist die Kolmogorov-Komplexität nicht formal berechenbar, wenn die beschreibende Turing-Maschine für das gegebene Trace nicht vorhanden ist. In dem hier vorliegenden Anwendungsfall wäre es möglich, die Kolmogorov-Komplexität mittels der erlernten Automaten aus dem L^*-Algorithmus näherungsweise zu berechnen. Der wichtigste Unterschied liegt darin, dass die gelernten Automaten Traces nicht wiedergeben, sondern lediglich akzeptieren können. Trotzdem sollte die Größe des gelernten Automaten zumindest eine relative Aussage über die Kolmogorov-Komplexität des Referenztraces erlauben.

4.4.1.2 Suche nach optimalen Subalphabeten

Aufgrund der NP-Schwere des Optimierungsproblems werden für das entropiebasierte Clustering meist heuristische Suchverfahren eingesetzt (wie z. B. Monte-Carlo in (Tao, Sheng, Mitsunori 2004) oder simulierte Abkühlung (englisch simulated annealing) in (Barbará, Li, Couto 2002)).

Zur Suche nach der optimalen Gruppierung der Zeichen des Alphabetes wird in dieser Arbeit ein evolutionärer Algorithmus (EA) verwendet. EA sind nicht-deterministische bzw. heuristische Suchverfahren und können für viele Optimierungsprobleme angewendet werden. Im Gegensatz zum Monte-Carlo-Algorithmus, welcher in (Barbará, Li, Couto 2002) verwendet wird, sind EA eher als parallele Suchverfahren anzusehen (Jong 2006). Die große Anzahl an Freiheitsgraden, eine unscharfe Bestimmung des Optimums und der vergleichsweise große Suchraum, der sich aus der Kombination von ca. 7.500 Zeichen ergibt, lässt daher für die hier gegebene Problemstellung einen EA als effiziente Lösung erscheinen.

Das Optimierungsziel wurde bereits in Gleichung 4-18 beschrieben. Zur Lösung dieses Problems muss der EA das Gesamtalphabet Σ^* in Subalphabete Σ_i aufteilen. Dabei sind weder die Anzahl der Subalphabete noch deren Größe (Anzahl der Zeichen) bekannt. Jedes Subalphabet beschreibt ein Subtrace τ_i (Gleichung 2-21). Die Entropie für $H(X_i)$ wird durch das Subtrace τ_i beschrieben (Gleichung 4-19).

Definition Abhängigkeit Subalphabet zu Entropie

$$\Sigma_i \longmapsto \tau_i \longmapsto H(X_i) \qquad 4\text{-}19$$

4.4.1.3 Grundlagen evolutionärer Algorithmen

Um die Wirkungsweise des angewendeten Verfahrens besser zu verstehen, wird an dieser Stelle eine kurze Einführung in die Wirkungsweise von evolutionären Algorithmen gegeben.

Ein EA arbeitet nach dem Vorbild der Evolution in der Natur. Eine gewisse Anzahl von Individuen passt sich nach mehreren Generationen mittels eines Selektionsdruckes

(Selektion) möglichst optimal an die Lebensbedingungen an. Nach dem natürlichen Vorbild wird die Information über die Ausprägung in Genen codiert. Diese Gene werden bei der Fortpflanzung aus den Elternpaaren kombiniert. Zur Fortpflanzung werden möglichst die am besten angepassten Individuen gewählt. Durch die Mutation einzelner Eigenschaften werden sprunghafte Veränderungen ermöglicht, was vor allem das Überspringen lokaler Maxima ermöglicht.

Grundlegend arbeitet ein EA meist nach einem einfachen Schema, wie er in Abbildung 4-9 als Pseudocode dargestellt ist.

```
Initialisiere Population;
Bewerte Individuen;
Wiederhole so lange wie Abbruchkriterium nicht erfüllt
{
      Wähle zwei Individuen zur Fortpflanzung aus;
      Erzeuge aus Eltern-Individuen ein Kind;
      Bewerte Individuen;
      Reduziere Populationsgröße;
}
```

Abbildung 4-9 Ablauf eines EA (Pseudocode)

4.4.2 Realisierung des Clustering-Verfahrens auf Basis der Entropie

Die wichtigsten Kriterien zur erfolgreichen Lösung des Optimierungsproblems mittels eines EA sind nach (Weise 2009; Jong 2006; Weicker 2007) folgende:

(1) Codierung einer gültigen Lösung – Abbildung der für die Lösung wichtigen Eigenschaften, so dass eine Kombination (Vererbung) möglich ist

(2) Bewertungsfunktion – Qualität der Lösung, welche ein Individuum repräsentiert (Berechnung der Fitness)

(3) Selektion – Auswahl der Individuen, welche sich fortpflanzen können

(4) Vererbung – Art und Weise der Weitergabe der Eigenschaften der Eltern an die Nachkommen (Kind), Kombination der Eigenschaften der Eltern

Der grobe Ablauf eines EA ist, wie gezeigt, recht simpel und kann allgemeingültig angewendet werden. Die Herausforderung für den Einsatz eines EA stellt die Realisierung der vier hier aufgezählten Kriterien dar. Daher wird in den nächsten Kapiteln näher auf die jeweilige Realisierung dieser Kriterien eingegangen.

4.4.2.1 Codierung einer gültigen Lösung:

Eine Codierung muss eine gültige Lösung des Optimierungsproblems darstellen können und Mechanismen wie Vererbung, Kreuzung und Mutation ermöglichen. Eine einzelne gültige Lösung des Optimierungsproblems wird typischerweise durch ein Individuum $\mathcal{I}$ repräsentiert. Ein Individuum kann mehrere Genome $\mathcal{G}$ besitzen. Wie die genaue

Abbildung von Eigenschaften auf Genome vorgenommen werden muss, ist dabei sehr abhängig von der Problemstellung.

Die allgemeinste Form der Codierung ist eine binäre Codierung. In den allermeisten Anwendungen stellt jedes Genom $\mathcal{G}$ einen einzelnen Lösungsparameter dar. Das Genom selbst ist dabei mit einer festen Länge binär codiert. Somit können bei einer Fortpflanzung einzelne Parameter verschiedener Individuen durch Kreuzung der Genome weitergegeben werden. Die Verwendung der Binärcodierung für Lösungsparameter ist zwar allgemeingültig, hat aber den Nachteil, dass sie unter Umständen den potenziellen Lösungsraum vergrößert oder verzerrt darstellt. Dies kann dazu führen, dass der EA keine zufriedenstellenden Ergebnisse liefert. Aus diesem Grund werden für manche Problemstellungen spezifische Lösungscodierungen verwendet, welche dann den Lösungsraum optimal abbilden.

Auch für die hier gegebene Problemstellung scheint eine binäre Codierung nicht geeignet. Daher wird eine spezielle Codierung vorgeschlagen, welche im Folgenden beschrieben wird.

Ein Individuum repräsentiert eine gültige Lösung; es muss daher die Aufteilung aller Zeichen des Gesamtalphabetes Σ^* in verschiedene Cluster abbilden können. Ein Cluster entspricht einem Subalphabet Σ_i. Die Größe der einzelnen Subalphabete ist variabel. Allerdings muss die Vereinigung aller Zeichen in allen Subalphabeten der Menge des Gesamtalphabetes entsprechen, wobei eine Überschneidung der Subalphabete nicht zugelassen wird.

In der vorgeschlagenen Codierung repräsentiert ein Genom $\mathcal{G}$ ein einzelnes Subalphabet $\mathcal{G}_i(\Sigma_i)$. Ein Individuum $\mathcal{I}$ besitzt somit i Genome. Der Vorteil dieser Lösung ist eine Einschränkung des Lösungsraumes auf gültige Lösungen. Ein Nachteil liegt allerdings in einer schwierigen Vererbung von Eigenschaften. Dies resultiert daraus, dass unterschiedliche Individuen nicht zwangsläufig vergleichbare Genome besitzen müssen, zwischen denen eine Vererbung von Eigenschaften stattfinden kann (Aufteilungen von Alphabeten in Subalphabete sind nur in seltenen Fällen identisch). Daher wird im Folgenden u. a. eine spezielle Vererbungsstrategie vorgeschlagen.

4.4.2.2 Bewertungsfunktion

Die Bewertungsfunktion ist eine wichtige Grundlage für den Selektionsprozess und ein entscheidender Faktor für den Erfolg des Optimierungsproblems. Wie in Gleichungen 4-18 f. ausgeführt wird, ist das Ziel der Optimierung die Minimierung der Entropie der aus den Subalphabeten Σ_i resultierenden Subtraces τ_i. Daher muss die Bewertungsfunktion die Entropie eines Subtraces τ_i abbilden, so dass ein besseres Ergebnis der Bewertungsfunktion auch eine kleinere Entropie des Subtraces darstellt. Die zu berechnende Größe sollte demnach, wie in Kapitel 4.4.1.1 beschrieben, der Kolmogorov-Komplexität entsprechen. Es wurde daher in Kapitel 4.4.1.1 vorgeschlagen, die Komplexität des mittels L^*-Algorithmus erlernten Automaten als Bewertungsfunktion zu verwenden.

Es zeigt sich allerdings bei der Anwendung des EA, dass die Verwendung des L^*-Algorithmus als Bewertungsfunktion zwar gute Werte liefert, aber die Laufzeit des L^*-Algorithmus einer Verwendung innerhalb eines EA entgegensteht. Es muss für jedes

Genom eines Individuums und für alle Individuen in mehreren hundert Elterngenerationen eine Bewertung durchgeführt werden. Daraus resultieren für eine Alphabetgröße, wie sie aus dem Evaluierungsdatensatz bekannt ist, mehr als 10^6 Berechnungen der Bewertungsfunktion. Selbst mit der Verringerung der maximalen Ausführungszeit des L^*-Algorithmus (zeitliches Abbruchkriterium) auf eine Minute würde ein Durchlauf auf einem handelsüblichen PC mehrere Monate dauern. Aus diesem Grund wird an dieser Stelle eine etwas abweichende Berechnung der resultierenden Entropie eines Subtraces vorgestellt, welche sich auf die Komprimierungsrate des Traces stützt.

Es kann gezeigt werden, dass als obere Grenze der Kolmogorov-Komplexität einer Zeichenkette die Größe des komprimierten ZIP-Archivs inklusive des entkomprimierenden Programmes angesehen werden kann. Für das hier zu lösende Problem ist dies allerdings nur beschränkt anwendbar, da, wie in Kapitel 4.4.1.1 bereits beschrieben, die Länge des Referenztraces keinen Einfluss auf die Entropie der gefundenen Lösung haben sollte.

Aus diesem Grund wird als Bewertungsfunktion nicht die absolute Größe des Archivs verwendet, sondern die Komprimierungsrate dieses Archivs vorgeschlagen. Die Komprimierungsrate *k* wird berechnet durch folgende Gleichung 4-20.

$$Komprimierungsrate\ k = 1 - \frac{Größe\ komprimiertes\ Trace}{Größe\ unkomprimiertes\ Trace} \qquad 4\text{-}20$$

Eine Komprimierungsrate von 1 stellt somit das beste Ergebnis bzw. die geringste Entropie dar. Mit abnehmender Komprimierungsrate nimmt die Entropie des Traces zu.

Die Anwendbarkeit dieser näherungsweisen Berechnung für die Entropie eines Traces wird im Folgenden anhand eines einfachen Versuches mit künstlich generierten Traces gezeigt. Dazu wurden aus bekannten Automaten mit variierender Komplexität Subtraces generiert und als ZIP-Archiv komprimiert. Für diese komprimierten Subtraces wurde anschließend nach Gleichung 4-20 die Komprimierungsrate berechnet. Wie in Abbildung 4-10 dargestellt, besitzen Traces, welche von Automaten geringer Komplexität ($\mathbb{K} = 1$) erzeugt werden, eine Komprimierungsrate nahe dem Maximum. Die Komprimierungsrate nimmt mit der Komplexität des zugrunde liegenden Automaten ab. Die untere Grenze der Komprimierungsrate stellt für das angewendete Verfahren die Komprimierungsrate der zufällig erzeugten Traces dar. Die Alphabetgröße hat auf die Komprimierungsrate offensichtlich keinen oder nur geringen Einfluss.

Somit kann davon ausgegangen werden, dass die Komprimierungsrate zumindest sehr stark mit der Kolmogorov-Komplexität korreliert. Die Berechnung der Komprimierungsrate kann für die gegebene Problemstellung in einem Bruchteil der benötigten Zeit, die ein L^*-Algorithmus benötigen würde, ausgeführt werden. Somit kann die benötigte Zeit für die Ausführung des EA für die gegebene Problemstellung auf eine Dauer von ca. 2 bis 3 Stunden reduziert werden.

Die Fitness eines Individuums bildet sich demnach aus dem Durchschnitt der Komprimierungsraten der Genome des Individuums[7] (Gleichung 4-21).

Definition Fitness eines Individuums

$$\mathbb{F}_{Indiviuum} = \frac{\sum_{i=0}^{n} k_i(\mathcal{G}_i)}{n} \qquad 4\text{-}21$$

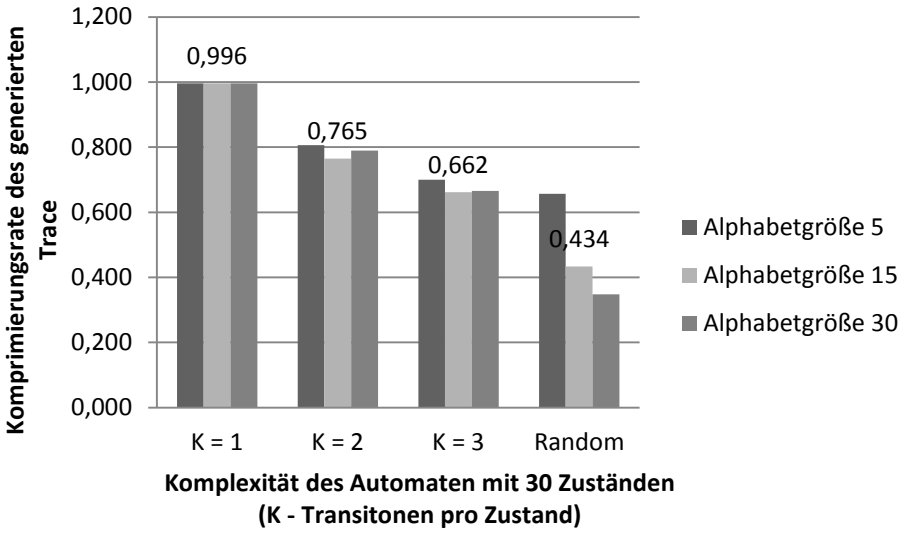

Abbildung 4-10 Komprimierungsraten künstlich erzeugter Traces von Automaten mit unterschiedlicher Komplexität

4.4.2.3 Selektion

Die Selektion ist ein Prozess, der bestimmt, welche Individuen für eine Fortpflanzung ausgewählt werden. Im Normalfall werden die Individuen mit der besten Fitness zur Fortpflanzung ausgewählt. Der Selektionsprozess sollte allerdings nicht zwangsläufig nur die besten Individuen auswählen, da dies eine zu schnelle Verkleinerung des Suchraumes des EA nach sich ziehen würde. Daher wird in diesem Zusammenhang meist mit Wahrscheinlichkeitsverteilungen gearbeitet.

In dem hier angewendeten Selektionsverfahren überleben nach der Fortpflanzung auch die Elternpaare. Innerhalb der Selektion werden aus allen verfügbaren Individuen die ausgewählt, welche in die nächste Fortpflanzungsrunde aufgenommen werden sollen. Die restlichen Individuen werden aus der Population entfernt. Dies ist sehr eng angelehnt an die Theorie des „Survival of the Fittest" im Sinne der Evolutionstheorie nach Darwin. Im Regelfall werden die Individuen mit der besten Fitness in die jeweils nächste Fortpflanzungsrunde aufgenommen. Um eine zu starke Dominanz der Eltern zu unterbinden, wird entweder ein Höchstalter für Individuen eingeführt oder ebenfalls mit einer Wahrscheinlichkeitsverteilung für das Entfernen von Individuen gearbeitet.

[7] Ein Genom repräsentiert jeweils ein Subtrace mit einem zugehörigen Subalphabet. Ein Individuum besteht aus so vielen Genomen, dass das gesamte Alphabet abgebildet ist.

Im vorliegenden Fall wird die Auswahl der Individuen zur Fortpflanzung in zwei Schritten vollzogen. In einem ersten Schritt werden die Individuen nach ihrer Fitness $\mathbb{F}$ sortiert und der Reihe nach jeweils zwei Individuen zur Fortpflanzung ausgewählt (1. und 2.; 3. und 4. …). Anschließend werden die ursprünglichen Elternpaare zufällig sortiert und nach dem gleichen Muster zur Fortpflanzung ausgewählt. So werden sowohl eine Auswahl der Fittesten als auch eine zufällige Auswahl realisiert.

Nach der Durchführung der Fortpflanzung werden alle vorhandenen Individuen mittels der Bewertungsfunktion bewertet. Die entstandene Menge an Individuen wird vor dem nächsten Fortpflanzungszyklus wieder auf die ursprüngliche Populationsgröße reduziert. Hierzu werden entsprechend nur die Individuen mit der besten Fitness herangezogen. Eine Begrenzung des Alters eines Individuums findet nicht statt.

4.4.2.4 Vererbung

Die Vererbung ist neben der Bewertungsfunktion einer der wichtigsten Mechanismen der EA. Der Prozess der Vererbung gibt Eigenschaften von den Elternindividuen auf die Kinderindividuen weiter. Dabei werden Eigenschaften zur Weitergabe ausgewählt, Eigenschaften kombiniert und eine Mutation von Eigenschaften durchgeführt. Zudem wird in vielen Fällen eine Reparatur der Genome eines Individuums vorgenommen, sollte durch die Kombination oder Mutation der Genome eine ungültige Lösung (ungültiges Individuum) entstehen.

Gewöhnlich werden, wie in der Natur auch, jeweils dieselben Genome (dieselbe Eigenschaft) der Elternindividuen verwendet, um über eine Kreuzung ein neues Genom des Kindindividuums zu erzeugen. Bei einer binären Codierung der Genome wird hierzu ein Kreuzungspunkt in Form einer Bit-Position festgelegt. Anschließend wird zufällig gewählt, welcher Teil der Bitfolge von welchem Elternteil beigesteuert wird. So entsteht ein neues Genom. In manchen Fällen wird nach der Erstellung eines neuen Genoms eine Mutation vorgenommen. Hierzu wird zufällig eine Bit-Position ausgewählt und entsprechend verändert. Ein Reparaturmechanismus überprüft die neu entstandenen Eigenschaften des Individuums auf ihre Gültigkeit. Da durch Mutation und Vererbung neue Werte für die Eigenschaften eines Individuums entstehen können, muss geprüft werden, ob die so repräsentierte Lösung (Individuum) eine gültige Lösung darstellt. Ist dies nicht der Fall, müssen die entsprechenden Genome repariert werden. Hierzu werden meist Werteschranken oder Resetwerte verwendet. Andere Reparaturmechanismen sind von der gegebenen Problemstellung abhängig.

Bei der Anwendung des oben beschriebenen Standardvorgehens auf die vorgeschlagene Abbildung der Genome mittels Subtraces und Subalphabete muss berücksichtigt werden, dass keine „gleichen Genome“ in den Elternpaaren vorhanden sind. Zur Kreuzung werden aber im Normalfall jeweils Genome, welche die gleiche Eigenschaft beschreiben, herangezogen. Daher ist bei der Repräsentation durch Subtraces und Subalphabete diese Standardprozedur für eine Vererbung nicht anwendbar und es muss eine eigene Strategie entwickelt werden. Diese wird im Folgenden dargestellt.

Finden von ähnlichen Genomen

Üblicherweise werden bei der Vererbung stets die gleichen Genome der Eltern miteinander kombiniert. Da im vorliegenden Fall jedes Individuum eine unterschiedliche Aufteilung der Subalphabete besitzt und die Anzahl der Genome inklusive deren Größe

unterschiedlich sind, muss ein anderer Weg gefunden werden, um Genome miteinander zu kreuzen.

Innerhalb der durchgeführten Versuche hat sich gezeigt, dass eine zufällige Auswahl der zu kreuzenden Genome nicht sinnvoll ist, da so keine (guten) Eigenschaften der Eltern erhalten bleiben. Prinzipbedingt muss es das Ziel sein, Genome für das Kindindividuum zu erzeugen, welche in möglichst großen Teilen mit den Elterngenomen übereinstimmen.

Daher wird zunächst nach ähnlichen Genomen gesucht. Ähnlich bedeutet in diesem Fall, dass die Subalphabete der Genome eine bestimmte Menge an gleichen Zeichen aufweisen. Werden in beiden Elternteilen Subalphabete mit mehreren gleichen Zeichen gefunden, so wird aus den übereinstimmenden Zeichen jeweils ein neues Genom erstellt. Zu diesem Zweck wird ein Gleichheitsfaktor berechnet. Dieser gibt an, wie groß der Anteil an gleichen Zeichen im Verhältnis zur Größe des Subalphabetes ist (Gleichung 4-22).

Definition Gleichheitsfaktor

$$\gamma = \frac{|\Sigma_a \cap \Sigma_b|}{|\Sigma_a|} \qquad 4\text{-}22$$

Für den Gleichheitsfaktor wird zunächst eine fixe Grenze festgelegt, ab der Subalphabete als gleich angenommen werden. Mit fortschreitender Ausführung des EA kann der Gleichheitsfaktor nach oben korrigiert werden, um gezielter größere (gute) Subalphabete zu vererben.

Direkte Vererbung von Genomen

Ein einfaches Kreuzen von Genomen ist aufgrund der Codierung nicht möglich. Daher wird aus jedem Elternindividuum jeweils zufällig ein komplettes Genom zur Vererbung gewählt. Das Genom mit der höheren Fitness (bzw. Kompressionsrate) wird an das Kindindividuum vererbt. Dies wird so oft wiederholt, bis für das Kindindividuum die gleiche Anzahl an Genomen wie in den Elternindividuen vorhanden ist. Somit ergibt sich für das Kindindividuum eine Mischung aus Genomen beider Elternindividuen.

Reparatur

Es ist nicht sichergestellt, dass nach den zwei vorangestellten Schritten (d. h. der direkten Vererbung und der Weitergabe mit ähnlichen Alphabeten) jeweils alle Zeichen des Alphabetes wieder genau einmal in den Genomen des Kindindividuums vorhanden sind. Da dies aber die zentrale Forderung einer korrekten Lösung ist, muss ein Reparaturmechanismus nachgeschaltet werden. Zu diesem Zweck werden in einem ersten Schritt die fehlenden Zeichen des Alphabetes identifiziert und zufällig auf neue Genome des Kindindividuums verteilt. In einem zweiten Schritt werden die bestehenden Genome des Kindindividuums auf doppelt vorkommende Zeichen geprüft und diese entsprechend gelöscht.

Weiterhin wird eine Begrenzung der Größe der Subalphabete vorgenommen, diese Begrenzung wird zum einen für die Initialisierung, zum anderen für die Erstellung neuer Genome im Zuge der Reparatur benötigt. Zu diesem Zweck werden eine obere und eine untere Grenze für die Alphabetgröße definiert.

4.4.3 Evaluierung für die Anwendung im selbstlernenden OoN-Erkennungsverfahren

Zur Evaluierung wurde das gleiche Referenztrace herangezogen, wie es auch für die Evaluierung des Clustering-Verfahrens mittels Frequenzspektrum verwendet wurde (Kapitel 4.2.3). Das Gesamtalphabet von 7.173 unterschiedlichen Zeichen wird dabei durch den EA in mehrere Subalphabete aufgeteilt. Zur Evaluierung, ob die gefundenen Subalphabete abhängige Alphabete darstellen, wird wie auch bei dem frequenzspektrumbasierten Clustering-Verfahren der L^*-Algorithmus verwendet (Kapitel 4.2).

4.4.3.1 Evaluierungsparameter

Zur Evaluierung werden zwei verschiedene Parametrierungen des EA verwendet.

In der ersten Variante *„EA-Clust-V1"* wird eine maximale Subalphabetgröße von 30 Zeichen und eine minimale von 5 Zeichen verwendet. Die minimale Subalphabetgröße wird in dieser Variante allerdings nur für die Initialisierung genutzt. Eine Begrenzung der Subalphabetgröße bei der Reparatur von Genomen wird nicht vorgenommen. Der Gleichheitsfaktor wird von 0,4 bis 0,6 nach jeweils 15 Epochen (Fortpflanzungsschritten) um 0,1 erhöht. Die Anzahl der Individuen wird nach jeder Epoche auf 10 begrenzt.

In der zweiten Variante *„EA-Clust-V2"* wird eine maximale Subalphabetgröße von 10 Zeichen und eine minimale von 4 Zeichen verwendet. Die minimale Subalphabetgröße wird in dieser Variante für die Initialisierung und für den Reparaturmechanismus genutzt. Somit werden keine Subalphabete mit weniger als 4 Zeichen akzeptiert. Der Gleichheitsfaktor wird von 0,2 bis 0,3 nach jeweils 15 Epochen (Fortpflanzungsschritten) um 0,1 erhöht. Die Anzahl der Individuen wird nach jeder Epoche auf 10 begrenzt.

4.4.3.2 Ergebnisse

Wie in Abbildung 4-11 gut zu sehen ist, verbessern sich die Fitness bzw. die Kompressionsrate der Individuen im Laufe der Durchführung des EA. Vorausgesetzt, die Fitnessfunktion ist richtig gewählt, kann davon ausgegangen werden, dass die Implementierung des EA tatsächlich eine Optimierung der Entropie der Subalphabete berechnet. Ein genauer Rückschluss auf die eigentliche Problemstellung des Clusterings ist daraus allerdings noch nicht ableitbar.

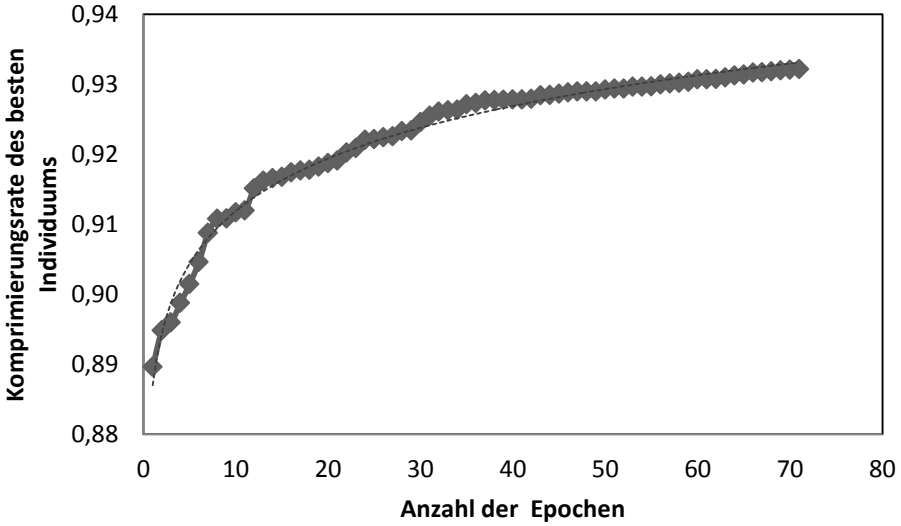

Abbildung 4-11 Exemplarische Darstellung der Entwicklung der Komprimierungsrate des besten Individuums bei einem exemplarischen Durchlauf eines EA (Trendlinie logarithmisch)

Um einen genauen Rückschluss auf die Anwendbarkeit des EA im Sinne des Clusterings von abhängigen Subalphabeten zu geben, werden die gefundenen Subalphabete mittels des L^*-Algorithmus bewertet.

Werden die noch nicht bewerteten Clustering-Ergebnisse betrachtet, wie sie in Tabelle 4-3 dargestellt sind, so sind zunächst alle Zeichen in Cluster aufgeteilt. Bei der *EA-Clust-V1*-Variante wurden jedoch die Cluster mit nur einem Zeichen herausgerechnet, da diese von vornherein nicht weiter verwendet werden können. Es zeigt sich, dass lediglich ca. 6 % der Zeichen auf diese Weise nicht weiter verwertet werden können. Auffällig ist auch, dass die durchschnittliche Subalphabet Größe von *EA-Clust-V1* nur halb so groß ist wie die von *EA-Clust-V2*.

Tabelle 4-3 Ergebnisse des entropiebasierten Clustering

Abstandsfunktion	Anzahl der erstellten Cluster (Anz. Subalphabete)	Event-Coverage (nur Cluster mit mehr als 1 Zeichen)	Trace-Coverage	Durch-schnittliche Subalphabet-größe (Anz. Zeichen)
EA-Clust-V1	2.125	95 %	94 %	3,38
EA-Clust-V2	1.189	100 %	100 %	6,03

Die Ergebnisse nach der Bewertung der gefundenen Subalphabete mittels des L^*-Algorithmus sind in Tabelle 4-4 dargestellt. Es zeigt sich, dass die Version *EA-Clust-V1* einen wesentlich größeren Anteil an erlernbaren bzw. abhängigen Subalphabeten erzeugt hat. Somit hat der Verlust durch die bereits aussortierten Subalphabete mit nur

einem Zeichen keine Auswirkung auf das Endergebnis. Es ist zu vermuten, dass die Möglichkeit, kleinere Subalphabete zu generieren und auch einzelne Zeichen ganz zu separieren, dem EA mehr Spielraum gibt, schwierig zuzuordnende Zeichen zu separieren.

Tabelle 4-4 Ergebnisse der Evaluierung der erstellten Cluster mittels des L^*-Algorithmus

	EA-Clust-V1	*EA-Clust-V2*
Anzahl der lernbaren Cluster (Anteil der erstellten Cluster)	1.030 (49 %)	447 (37 %)
Durchschnittliche Komplexität $\mathbb{K}$	1,52	1,60
Event-Coverage | Anteil der geclusterten Subalphabete am Gesamt-Alphabet (absolute Anzahl der Zeichen)	51 % (3.638)	32 % (2.305)
Trace-Coverage | Anteil der Subtraces an Gesamt-Trace (absolute Anzahl der Zeichen)	21 % (543.480)	11 % (280.890)
Anzahl der Alphabet-Zeichen, welche ausschließlich mit diesem Abstand geclustert wurden	2.355	1.022

4.4.4 Fazit

Die Ergebnisse des entropiebasierten Clustering-Verfahrens sind in etwa vergleichbar mit denen des frequenzspektrumbasierten Clustering-Verfahrens. Die Konfiguration *EA-Clust-V1* schneidet bei der Evaluierung am besten ab und gruppiert ca. 51 % des Gesamtalphabetes (Event-Coverage), was von keinem der frequenzspektrumbasierten Ansätze erreicht wurde. Allerdings ist die resultierende Abdeckung des Traces (Trace-Coverage) nur bei ca. 21 %. Dies deutet darauf hin, dass mit dieser Methode vorrangig seltener vorkommende Zeichen geclustert werden. Weiterhin ist die durchschnittliche Subalphabetgröße signifikant kleiner als bei den frequenzspektrumbasierten Clustering-Verfahren. Dies ist als nachteilig zu werten, da so einige Abhängigkeiten nicht aufgelöst werden.

Die benötigte Rechendauer dieser Clustering-Methode ist etwas größer als die der frequenzspektrumbasierten Methode. Mit einer Dauer von ca. 1 bis 2 Stunden auf einem handelsüblichen PC für das gegebene Referenztrace hält sich die Rechenzeit noch in Grenzen, um das Verfahren auch praktisch anwenden zu können.

Eine hervorzuhebende Eigenschaft der hier vorgestellten Methode ist, dass sie mit jedem Durchlauf etwas abweichende Ergebnisse produziert. Obwohl sich dies auch nachteilig auswirken kann, besteht damit die Möglichkeit, das Verfahren z. B. iterativ anzuwenden. Das bedeutet, dass für Zeichen, welche nicht erfolgreich gruppiert

wurden, das Verfahren wiederholt werden kann. Zudem kann das Verfahren auch mehrmals angewendet werden, um das jeweils beste Ergebnis zu verwenden.

4.5 Optimierung durch Kombination verschiedener Lösungsansätze

In den vorhergehenden Kapiteln 4.3 und 4.4 wurden einzelne Lösungen diskutiert, mit denen jeweils aus einem Alphabet mehrere möglichst abhängige Subalphabete extrahiert werden können. Zu diesem Zweck sind unterschiedliche Clustering-Verfahren entwickelt und evaluiert worden. Resultierend aus der Anwendung von Clustering-Verfahren wird dabei jedes Zeichen jeweils exakt einem Cluster zugeordnet.

Das eigentliche Ziel der Anwendung der Clustering-Verfahren ist die Verringerung der Komplexität des Lernproblems durch das Finden von unabhängigen Automaten. Würde das Clustering-Verfahren perfekte Ergebnisse liefern, so gäbe es zwischen den resultierenden Automaten keine Abhängigkeiten und die Automaten wären nur durch ein Produkt vereinbar. Bei genauer Betrachtung der Clustering-Ergebnisse nach der Evaluierung mittels des L^*-Algorithmus ist jedoch festzustellen, dass die unterschiedlichen Clustering-Verfahren zwar abhängige Alphabete finden, diese allerdings nicht vollständig sein müssen. Vollständig bedeutet in diesem Fall, dass alle Zeichen identifiziert werden, welche zu diesem Handlungsstrang gehören. Es wird sozusagen jeweils nur eine Teilmenge des abhängigen Alphabetes gefunden.

Dies hat für die Evaluierung mittels des L^*-Algorithmus keine Auswirkungen, da auch Teilmengen von abhängigen Alphabeten sinnvolle Automaten darstellen. Es verringert aber zum einen die Aussagekraft der gelernten Automaten, da Abhängigkeiten zwischen nicht enthaltenen Zeichen verloren gehen. Zum anderen sinkt dadurch der Anteil der Zeichen, welche überhaupt von den gelernten Automaten abgedeckt werden (Event-Coverage).

Für die Anwendung zur Bewertung des Systemverhaltens ist es nicht erforderlich, dass ein Zeichen jeweils nur von einem Automaten beschrieben wird, wie es bei der Anwendung der Ergebnisse eines Clustering-Verfahrens der Fall ist. Es kann im Gegensatz sogar förderlich wirken, wenn ein Zeichen von mehreren Automaten überwacht wird. Daher wird im Folgenden als Optimierungsmöglichkeit die Kombination der Clustering-Verfahren untersucht. Dies soll zu einer höheren Systemabdeckung und einer genaueren Beschreibung des Referenztraces führen.

4.5.1 Beschreibung der Optimierung

Der Ansatz für die hier vorgestellte Optimierung des Cluster-Verfahrens liegt hauptsächlich in der Kombination verschiedener Clustering-Ergebnisse. Wie oben bereits geschildert, kann ein einzelnes Clustering-Verfahren meist nicht alle Zeichen, welche zu einem abhängigen Subalphabet gehören, einem gemeinsamen Subalphabet zuordnen. Zudem sind nach der Evaluierung mittels L^* je nach Clustering-Verfahren zwischen 90 % und 55 % der Zeichen im Sinne der Event-Coverage nicht zuordenbar.

Der Clustering-Verfahrensschritt des selbstlernenden OoN-Erkennungsverfahren soll aber möglichst viele abhängige Zeichen gruppieren, so dass ein möglichst großer Teil der Traces mittels gelernter Automaten geprüft werden kann. Somit scheint es sinnvoll, die Ergebnisse aus mehreren verschiedenen Clustering-Verfahren zu verbinden. Eine

Kombination der Ergebnisse der einzelnen Clustering-Verfahren kann durch zwei mögliche Vorgehensweisen erreicht werden.

Zum Ersten können die entstandenen Subalphabete auf Überschneidungen überprüft und danach die entsprechenden Subalphabete zusammengeführt werden. Aus den neu erstellten Subalphabeten können dann wiederum neue Automaten mittels L^* erlernt werden. Die zweite und einfachere Möglichkeit besteht darin, alle bereits gelernten und für gut befundenen Automaten gleichzeitig bzw. parallel zu verwenden.

Der resultierende Automat A^*, welcher bei der Vereinigung zweier überschneidender abhängiger Subalphabete Σ_i entsteht, entspricht der Komposition der beiden Automaten A_i mit $\Sigma^* = \Sigma_1 \bigcup \Sigma_1$ und $A^* = A_1 \circ A_1$ gemäß der Gleichungen 2-18 und 2-20. Dabei ergeben sich die Abhängigkeiten zwischen Alphabet, Trace und Automat gemäß der Gleichungen 2-21 und 2-22 wie folgt: $\Sigma_1 \rightarrow \tau_1 \rightarrow A_1$, $\Sigma_2 \rightarrow \tau_2 \rightarrow A_2$ und $\Sigma^* \rightarrow \tau^* \rightarrow A^*$.

Es ist demzufolge mit einem geringen Verlust an Beschreibungsgenauigkeit zu rechnen, wenn die zweite Variante (d. h. die der parallelen Ausführung) statt der Zusammenführung der Subalphabete gewählt wird. Der Verlust der Beschreibungsgenauigkeit der erlernten Automaten wird bei einer schwachen Überlappung der Subalphabete entsprechend größer sein und resultiert aus der nicht vollständig aufgelösten Kompositionsberechnung des resultierenden Automaten. Die Einbuße an Beschreibungsgenauigkeit muss aber mit der schwerer zu bewältigen Lernaufgabe abgewogen werden. Stichprobenartige Tests haben gezeigt, dass nicht aus allen vereinigten Subalphabeten zwangsläufig wieder ein gültiger Automat gelernt werden konnte. Aus diesem Grund wird für die Optimierung die einfachere zweite Variante der parallelen Ausführung mehrerer Automaten mit überschneidenden Subalphabeten gewählt.

Eine schematische Darstellung der Optimierung der Lernergebnisse durch Überlappung der Clustering-Ergebnisse liegt in Abbildung 4-12 vor. Es werden aus allen Clustering-Verfahren die positiv bewerteten und mit L^* erlernten Automaten zur Bewertung herangezogen. So können sowohl die Event- als auch die Trace-Coverage signifikant verbessert werden. Weiterhin wird durch eine größere Anzahl an Überlappungen eine bessere Beschreibung von Querbeziehungen zwischen den einzelnen Automaten erreicht. Dies verbessert zudem die Qualität des abgebildeten Verhaltens und somit der OoN-Erkennung (d. h. Erhöhung der Rate der erkennbaren Abweichungen, Richtig-Positiv-Rate).

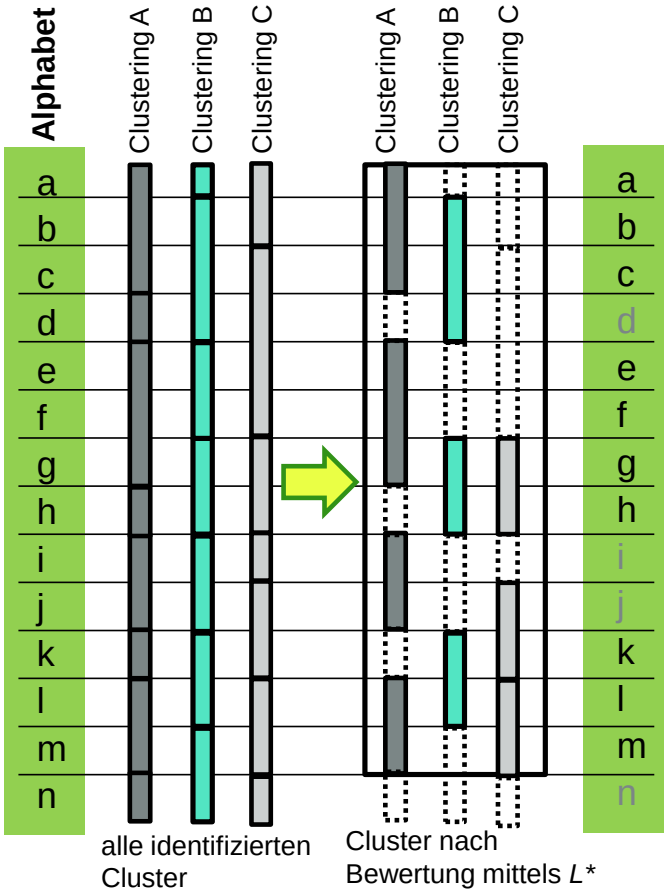

Abbildung 4-12 Schematische Darstellung der Kombination bzw. Überlagerung mehrerer Clustering-Ergebnisse

4.5.2 Ergebnisse der Optimierung

Für die vorgestellte Optimierung müssen keine neuen Versuche durchgeführt werden. Die Grundlage bildet die Gesamtheit der erlernten Automaten aus der Evaluierung des frequenzbasierten Clusterings und des entropiebasierten Clusterings, wie sie bereits in den Kapiteln 4.3.2.3 und 4.4.3 vorgestellt worden sind.

Zunächst werden die Ergebnisse des frequenzbasierten Clusterings (Tabelle 4-2) zusammengeführt. Die Resultate sind in Tabelle 4-5 und Abbildung 4-13 dargestellt. Die Resultate der Vereinigungsmenge gegenüber dem besten Einzelresultat, dem Manhattan-Abstand (46 % Event-Coverage und 49 % Trace-Coverage), weisen dabei eine deutliche Verbesserung auf (69 % Event-Coverage und 65 % Trace-Coverage). Durch die recht ähnlichen Clustering-Ergebnisse der Einzelverfahren ist zudem eine hohe Überdeckung der Zeichen zu beobachten. So ist ein Zeichen durchschnittlich in 3,5 Subalphabeten enthalten. Insgesamt steht aus dem frequenzbasierten Clustering eine Anzahl von 2.848 Subalphabeten mit resultierenden Überwachungsautomaten zur Verfügung.

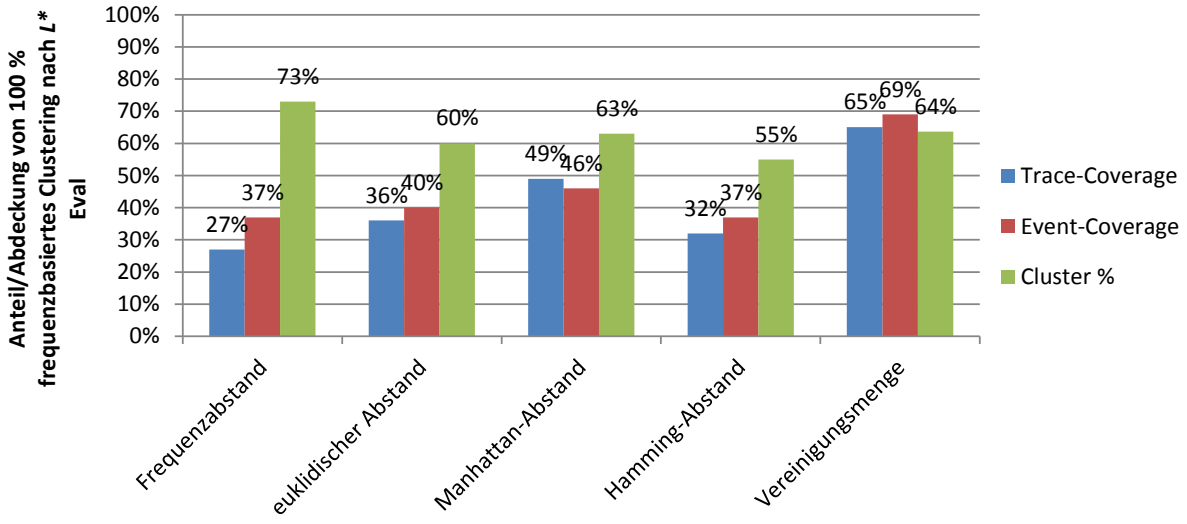

Abbildung 4-13 Darstellung der Vereinigungsmengen der frequenzbasierten Clustering-Verfahren nach L^*-Evaluierung

Tabelle 4-5 Ergebnisse der zusammengeführten frequenzbasierten Clustering-Ergebnisse

	Zeichenanzahl	Anteil
Vereinigungsmenge der Subalphabete aller erlernbaren Cluster (Event-Coverage)	4.981	69 %
Vereinigungsmenge der Subtraces aller erlernbaren Cluster (Trace-Coverage)	1.664.373	65 %

Nachfolgend werden die Ergebnisse des entropiebasierten Clusterings (Tabelle 4-4) zusammengeführt. Die Resultate sind in Tabelle 4-6 und Abbildung 4-14 dargestellt. Die Vereinigungsmenge ist im Vergleich zu den frequenzbasierten Ergebnissen deutlich größer, verglichen mit den Einzelergebnissen der Clustering-Verfahren. Dies resultiert daraus, dass die Überschneidung der Clustering-Ergebnisse aus *EA-Clust-V1* und *EA-Clust-V2* nur sehr gering ist. Somit können mit der Kombination beider Versuche eine Event-Coverage von 64 % und eine Trace-Coverage von 28 % erreicht werden. Insgesamt steht aus dem entropiebasierten Clustering eine Anzahl von 1.477 Subalphabeten mit resultierenden Überwachungsautomaten zur Verfügung.

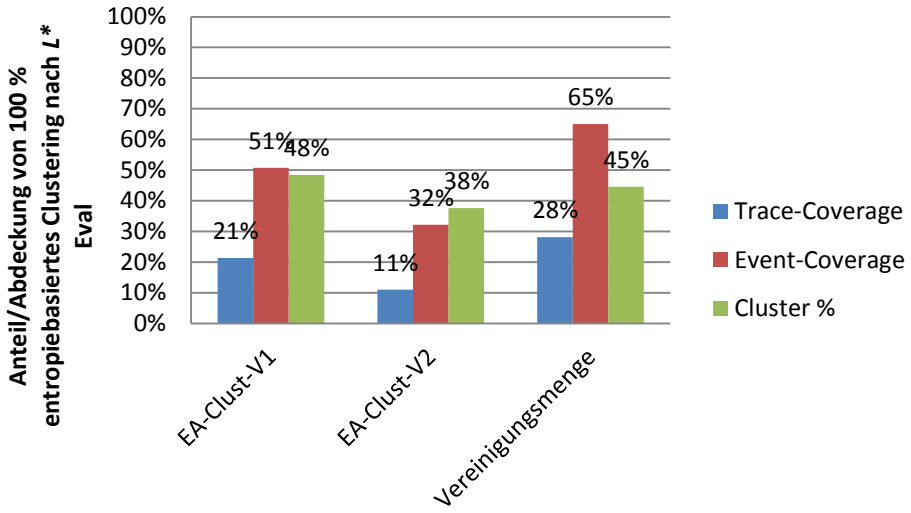

Abbildung 4-14 Darstellung der Vereinigungsmengen der entropiebasierten Clustering-Verfahren

Tabelle 4-6 Ergebnisse der zusammengeführten entropiebasierten Clustering-Ergebnisse

	Zeichenanzahl	Anteil
Vereinigungsmenge der Subalphabete aller erlernbaren Cluster (Event-Coverage)	4.660	65 %
Vereinigungsmenge der Subtraces aller erlernbaren Cluster (Trace-Coverage)	716.449	28 %

Abschließend werden die Ergebnisse aus den frequenzbasierten und entropiebasierten Clustering-Verfahren zusammengeführt. Die Ergebnisse sind in Abbildung 4-15 und Tabelle 4-7 dargestellt.

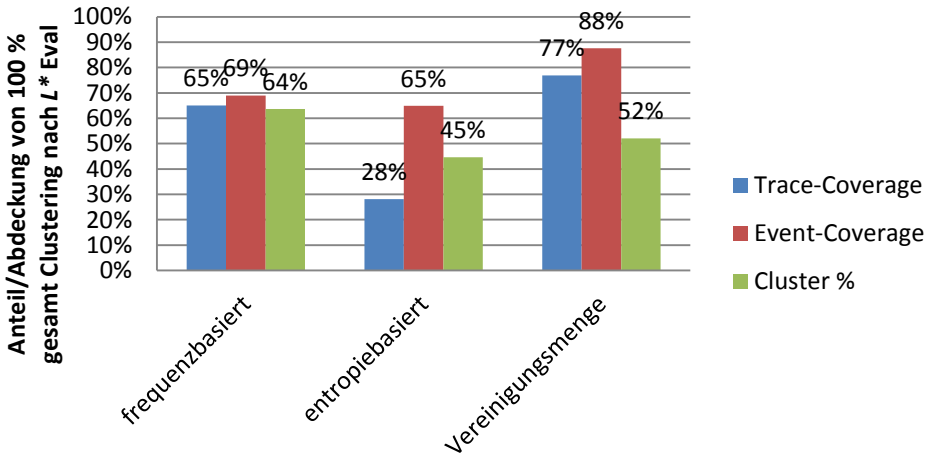

Abbildung 4-15 Darstellung der Vereinigungsmengen der frequenz- und entropiebasierten Clustering-Verfahren

Tabelle 4-7 Ergebnisse der Zusammenführung der frequenz- und entropiebasierten Clustering-Verfahren

	Zeichenanzahl	Anteil
Vereinigungsmenge der Subalphabete aller erlernbaren Cluster (Event-Coverage)	6.285	88 %
Vereinigungsmenge der Subtraces aller erlernbaren Cluster (Trace-Coverage)	1.958.579	77 %

Insgesamt ergeben sich bei der Zusammenführung aller Clustering-Ergebnisse 2.848 Subalphabete mit zugehörigen Überwachungsautomaten, welche 77 % des Alphabetes abdecken (Event-Coverage). Für das Referenztrace ergibt dies eine Abdeckung von 88 % aller aufgezeichneten Daten (Trace-Coverage).

4.5.3 Einbeziehung von Wissen aus vorhandener Spezifikation

Ähnlich wie bei den bereits untersuchten Lernverfahren kann es auch für die Clustering-Verfahren von Vorteil sein, zusätzliche Informationen aus einer vorliegenden Systembeschreibung oder Spezifikation heranzuziehen. Diese Informationen sind im Falle des Clusterings vornehmlich Informationen über bekannte funktionale Zusammenhänge.

Ist beispielsweise bekannt, welche Eingangs- und Ausgangssignale zu einer bestimmten Funktion gehören, so könnten diese als ein Cluster angenommen werden. Denkbare mögliche Abhängigkeiten können sich beispielsweise auch durch einen gemeinsamen Empfänger mehrerer Signale oder einen gemeinsamen Sender ergeben. Sicherlich gibt es noch eine Vielzahl anderer bekannter bzw. ableitbarer Abhängigkeiten aus der Spezifikation. Zur Anwendung für das hier untersuchte selbstlernende OoN-Erkennungs-

verfahren ist ausschließlich die Frage von Interesse, inwieweit diese Informationen einen Mehrwert für die Bewertung des Kommunikationsverhaltens mit sich bringen. Ist dieser Mehrwert erkennbar, sollte es eine Möglichkeit geben, diese Informationen in den Clustering-Prozess einzubeziehen.

Eine Einbeziehung von Event-Gruppierungen aus externen Wissensquellen kann für das vorgestellte Verfahren äquivalent zu einem weiteren Clustering-Verfahren behandelt werden. Somit ist es ohne weiteres möglich, die automatisch generierten Cluster durch weitere Cluster aus bekannten Beziehungen zu ergänzen. Können für diese neuen Cluster Automaten mittels des L^*-Algorithmus erlernt werden, so ist auch davon auszugehen, dass die Informationen aus der Spezifikation einen zusätzlichen Nutzen für das selbstlernende OoN-Erkennungsverfahren mit sich gebracht haben.

4.6 Fazit

Es wurden verschiedene Methoden zum Clustering des Alphabetes vorgestellt, mit denen es möglich ist, abhängige Subalphabete zu identifizieren. Mittels der Optimierung in Form der Zusammenfassung unterschiedlicher Clustering-Verfahren konnte eine signifikante Verbesserung in der Abdeckung des Alphabetes und des Traces erreicht werden. Somit kann das Verhalten für 88 % der Events des Referenztraces durch erlernte Automaten geprüft werden.

Durch die Evaluierung bzw. Bewertung mittels des L^*-Algorithmus wurde eine Methode vorgeschlagen, welche es ermöglicht, die Clustering-Ergebnisse zu verifizieren. Diese Methode wird sowohl zur Evaluierung der Clustering-Verfahren als auch zur Bewertung und Auswahl der gefundenen Subalphabete für das selbstlernende OoN-Erkennungsverfahren angewendet. Insgesamt wurden mit diesem Bewertungsverfahren 52 % der erstellten Cluster als brauchbar eingestuft. Eine sichere Aussage über die Brauchbarkeit der gelernten Automaten in Bezug auf die OoN-Erkennung ist allerdings auf Grundlage der vorliegenden Methodik noch nicht endgültig möglich. Zu diesem Zweck werden im nächsten Kapitel weitere zu prüfende Traces herangezogen, mit denen die gelernten Automaten einer detaillierteren Bewertung unterzogen werden.

5 Anwendung des selbstlernenden OoN-Erkennungsverfahrens

Im Kapitel 3 wurden Lernverfahren zum Erstellen von Abhängigkeitsmodellen aus den Subtraces und im Kapitel 4 Clustering-Verfahren zum Finden von unabhängigen Subalphabeten vorgestellt. Diese beiden Einzelschritte bilden zusammen den ersten Teilschritt des selbstlernenden OoN-Erkennungsverfahrens, welcher in dieser Arbeit als Lernphase bezeichnet wird.

In diesem Kapitel wird nun die Anwendungsphase des vorgestellten selbstlernenden OoN-Erkennungsverfahrens näher betrachtet. In der Anwendungsphase werden die erstellten Automaten für ihren eigentlichen Zweck, die Erkennung von Abweichungen zum Referenztrace, benutzt. Dieses Kapitel untersucht daher gezielt die Wirkungsweise und Anwendbarkeit des selbstlernenden OoN-Erkennungsverfahrens in seinem letztendlichen Anwendungsumfeld. Der Schwerpunkt der Untersuchungen liegt dabei auf der für den Nutzer wichtigen Größe der Fehlalarme und der Erkennung tatsächlicher Abweichungen. Um eine möglichst hohe Aussagekraft der Ergebnisse in dem späteren Anwendungsumfeld zu gewährleisten, wird die Evaluierung anhand mehrerer realer Fahrzeug-CAN-Aufzeichnungen durchgeführt.

Im Kapitel 5.1 werden zunächst der Aufbau und die Wirkungsweise des selbstlernenden OoN-Erkennungsverfahrens in der Anwendungsphase vorgestellt. Anschließend werden im Kapitel 5.2 die Problematik der Fehlalarmrate näher betrachtet sowie eine Optimierung zur Verringerung dieser vorgestellt. Die vorgeschlagene Optimierungsmethode basiert auf der Auswertung der OoN-Indikationen mehrerer geprüfter Traces und wird daher in der Anwendungsphase ausgeführt. Aufbauend auf den Ergebnissen der Optimierung wird zudem eine Abschätzung über die Größenordnung der zu erwartenden Fehlalarmrate erstellt. In Kapitel 5.4 wird die experimentelle Bestimmung der Erkennungsrate von tatsächlichen Abweichungen beschrieben. Abschließend werden in Kapitel 5.5 alle wichtigen Ergebnisse und Kennzahlen der Evaluierung diskutiert. Mit Hilfe der zusammengefassten Erkenntnisse soll es möglich sein, eine Einschätzung über die Anwendbarkeit und den zu erwartenden Mehrwert bei der Auswertung von Netzwerktraces bei Fahrzeugtests zu geben.

5.1 Wirkungsweise der OoN-Erkennung in der Anwendungsphase

In der Lernphase werden nach dem Clustering des Alphabetes zur Prüfung des Verhaltens des Referenztraces mehrere Abhängigkeitsmodelle gelernt. Wie in Abbildung 5-1 zu sehen ist, sind in der Anwendungsphase Abhängigkeitsmodelle, welche das Verhalten des Referenztraces beschreiben, bereits erlernt worden. Diese Abhängigkeitsmodelle sollen nun dazu genutzt werden, herauszufinden, ob weitere zu prüfende Traces das gleiche Verhalten beschreiben. Beschreiben die zu prüfenden Traces abweichende Systemzustände, so liegt eine Abweichung vom Verhalten des Referenztraces vor. In diesem Fall wird dies als Abweichung von der mittels des Referenztraces festgelegten Norm gewertet.

In der Anwendungsphase wird geprüft, ob das Abhängigkeitsmodell das zu prüfende Trace akzeptiert. Dazu wird jedes Event in der entsprechenden Reihenfolge des zu prüfenden Traces dem Abhängigkeitsmodell präsentiert. Aufgrund des internen Status

des Abhängigkeitsmodells kann entschieden werden, ob das präsentierte Event ein Event ist, welches in diesem Zustand erwartet wird oder nicht. Wird ein derartiges Event vom Abhängigkeitsmodell akzeptiert, so geht dieses, in den durch das akzeptierte Event resultierenden Zustand, über und wartet auf das nächste Event.

Je nach eingesetztem Lernverfahren und resultierendem Abhängigkeitsmodell unterscheiden sich die Wirkungsweisen der Akzeptanzvergleiche ein wenig. So wird bei den neuronalen Netzen die Gewichtung der Ausgangsneuronen bewertet und sowohl bei den Markov-Ketten als auch bei den Akzeptorautomaten des L^*-Lernverfahrens werden jeweils die abgehenden Transitionen des gerade aktiven Zustandes zur Prüfung des jeweils nächsten Zeichens des Traces genutzt. Die genaue Anwendung der erlernten Abhängigkeitsmodelle der untersuchten Lernverfahren ist in Kapitel 3 beschrieben.

Nach der Evaluierung in Kapitel 3.3.4 wurde festgestellt, dass der L^*-Algorithmus das am besten geeignete Lernverfahren für eine OoN-Erkennung ist. Zudem stehen durch die Anwendung des L^*-Algorithmus zur Bewertung der Cluster-Ergebnisse nach der Durchführung des Clustering-Verfahrensschrittes die notwendigen Akzeptorautomaten zur Verfügung. Daher wird für das hier entwickelte und evaluierte selbstlernende OoN-Erkennungsverfahren der L^*-Algorithmus verwendet, um Abhängigkeitsmodelle zu erlernen.

Die Anwendungsphase unterteilt sich in drei Arbeitsschritte: (i) die Datenaufarbeitung, (ii) die Akzeptanzfrage und (iii) die OoN-Indikation bei einer fehlgeschlagenen Akzeptanzprüfung (Abbildung 5-1). Diese Arbeitsschritte werden im Folgenden näher erläutert.

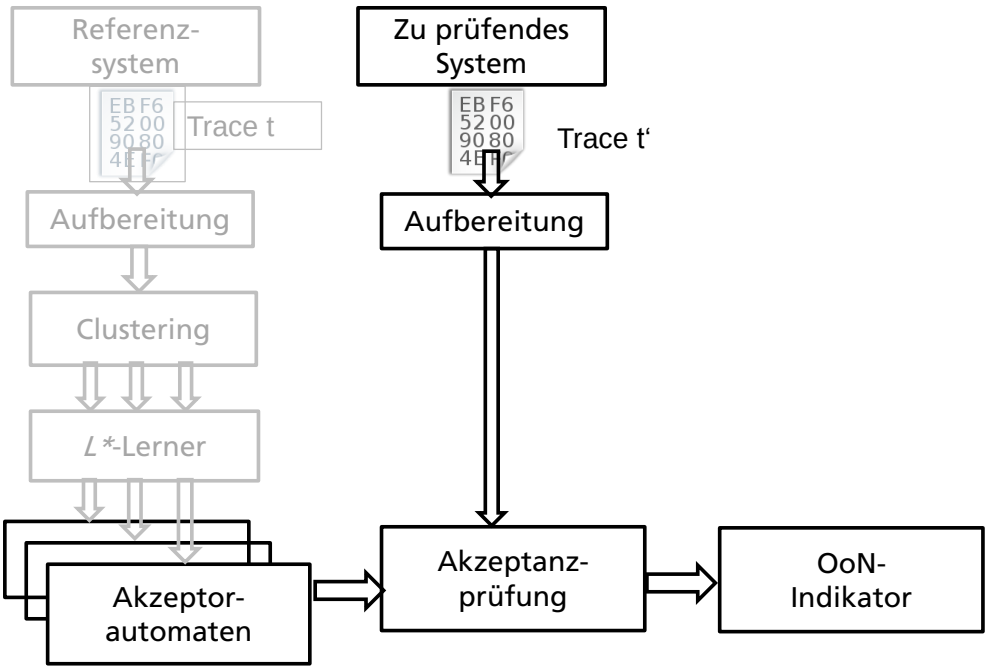

Abbildung 5-1 Anwendungsphase des selbstlernenden OoN-Erkennungsverfahrens (die erlernten Akzeptorautomaten unterziehen das zu prüfende Trace einer Akzeptanzprüfung und generieren eine OoN-Indikation)

5.1.1 Datenaufbereitung

Die Aufarbeitung der zu prüfenden Traces erfolgt im Wesentlichen analog zur Aufarbeitung des Referenztraces. Nach der Datenaufbereitung des Netzwerktraces liegt ein Trace mit zu prüfenden Events vor, wie es auch als Input für das Lernverfahren in Kapitel 3.4.2 verwendet wird. Der wichtigste Unterschied bei der Aufbereitung im Vergleich zum Referenztrace in der Lernphase liegt darin, dass die Vergabe der IDs für Events nicht mehr willkürlich erfolgen darf. Die Event-IDs, welche im Referenztrace für bestimmte Signal/Wert-Kombinationen vergeben worden sind, müssen für neue zu prüfende Traces in der Anwendungsphase in der gleichen Konstellation zugewiesen werden.

Daher wird die bei der Generierung des Referenztraces erstellte Übersetzungstabelle herangezogen, um ein eindeutiges Mapping zwischen Signal/Wert-Kombinationen und Event-IDs äquivalent zum Referenztrace herzustellen. Wie auch bei der Datenaufbereitung des Referenztraces werden anschließend periodisch mit gleichem Wert gesendete Events herausgefiltert, um dem zyklisch wiederholten Datenversand im CAN-Netzwerk entgegenzuwirken.

Abschließend werden die in der Lernphase identifizierten Subalphabete eingelesen und das zu prüfende Trace wird entsprechend gefiltert, so dass für jeden Automaten ein entsprechendes Subtrace zur Prüfung zur Verfügung steht.

5.1.2 Akzeptanzprüfung

Nach dem Clustering und der Anwendung des L^*-Algorithmus stehen für die Anwendungsphase mehrere Akzeptorautomaten zur Verfügung, um weitere Traces dahin gehend zu prüfen, ob sie dem Verhalten des Referenztraces entsprechen.

Grundlegend handelt es sich bei der Akzeptanzprüfung um einen Klassifikationsvorgang, wie er in Kapitel 2.3.5 beschrieben ist. Klassifiziert wird in: „Verhalten entspricht dem Referenztrace“ und „Verhalten entspricht *nicht* dem Referenztrace“. Wie die Klassifizierung realisiert ist, hängt stark vom eingesetzten Lernverfahren ab. Im Falle der hier vorgeschlagenen Akzeptorautomaten, wie sie vom L^*-Algorithmus erstellt werden, übersetzt sich die Klassifizierung in: „zu prüfendes Trace wird vom Automaten akzeptiert“ und „zu prüfendes Trace wird vom Automaten *nicht* akzeptiert“. Da nicht nur ein Automat zur Klassifizierung verwendet wird, sondern mehrere, müssen jeweils alle Automaten ihr jeweiliges Subtrace akzeptieren, damit von einem positiven Testergebnis gesprochen werden kann. Sobald ein Automat ein Subtrace nicht akzeptiert, ist das Klassifizierungsergebnis negativ zu werten, also im Sinne von „Verhalten entspricht *nicht* dem Referenztrace“.

Es ist an dieser Stelle festzuhalten, dass die Akzeptanzprüfung im Prinzip dem Verfahren entspricht, mit welchen die entsprechenden Lernverfahren evaluiert wurden. Der wesentliche Unterschied besteht darin, dass bei der Evaluierung das ausgeführte System bekannt war und somit ein genauer Rückschluss auf das Verhalten der Lernverfahren gezogen werden konnte. Bei der Anwendung des Lernverfahrens mit realen Daten ist dieser direkte Rückschluss nicht möglich. Es ist aber davon auszugehen, dass die gefundenen Eigenschaften des Lernverfahrens, welche in der Evaluierung mit künstlichen Daten ermittelt wurden, auch auf reale Daten übertragbar sind.

5.1.3 OoN-Indikation

Die OoN-Indikation reflektiert dem Nutzer das Ergebnis der Akzeptanzprüfung. Somit ist die OoN-Indikation der letzte Schritt in der OoN-Erkennung. An dieser Stelle muss der Nutzer darüber entscheiden, wie mit der festgestellten Abweichung vom Referenztrace weiter umgegangen werden soll.

Wird eine OoN-Indikation ausgelöst, heißt das für den Nutzer zunächst, dass ein bisher noch nicht beobachtetes, also nicht im Referenztrace enthaltenes Verhalten entdeckt worden ist. Aus Sicht des Nutzers gibt es nun drei verschiedene Ursachen, weshalb die OoN-Indikation ausgelöst worden ist. Diese sind:

(1) Es wurde ein korrektes, aber noch nicht beobachtetes Verhalten entdeckt. Dies kann z. B. der Fall sein, wenn neue Features getestet werden oder wenn die Software des Fahrzeuges angepasst wurde.

(2) Das beobachtete Verhalten ist nicht korrekt im Sinne der Spezifikation und wird als Bug bzw. Fehler bewertet.

(3) Es wurde fälschlicherweise eine OoN-Indikation ausgelöst (Fehlalarm).

Die Entscheidung, welcher dieser drei Fälle vorliegt, muss abschließend von einem Experten getroffen werden.

5.2 Verfahren zur Verringerung der Fehlalarmrate

Es ist leicht vorstellbar, dass eine hohe Anzahl an Fehlalarmen das selbstlernende OoN-Erkennungsverfahren schnell unbrauchbar machen kann. Fehlalarme sind für derartige Verfahren nicht auszuschließen. So produzieren viele statische Code-Analyse-Tools bis zu 90 % Fehlalarme (Kremenek u. a. 2004), dies wird aber vom Nutzer akzeptiert, wenn die Anzahl der insgesamt zu prüfenden Indikationen überschaubar bleibt. Da auch das hier vorgestellte Verfahren einer solchen Problematik ausgesetzt ist, wird an dieser Stelle ein Verfahren entworfen, mit dem die Anzahl der ausgelösten Fehlalarme signifikant reduziert werden kann.

Um die Ursachen von Fehlalarmen besser zu verstehen, werden diese im Folgenden zunächst genauer untersucht. Daraus ableitend wird ein Verfahren vorgeschlagen, welches die Fehlalarmrate nach der Überprüfung mehrerer Traces signifikant reduzieren kann.

5.2.1 Ursachen für Fehlalarme

Zunächst ist festzustellen, dass der L^*-Algorithmus, so wie er in dieser Arbeit angewendet wird, kein heuristisches Verfahren darstellt. Er ist vielmehr ein definierter bzw. terminierter und nicht zufallsbedingter Algorithmus (Kapitel 3.3.3.1). In (Angluin 1987) wurde bereits gezeigt, dass der L^*-Algorithmus einen Automaten konstruiert, der die ihm vorgestellten Sequenzen bzw. Gegenbeispiele optimal abbildet. Durch die hier vorgestellte Realisierung der Äquivalenzfrage ist weiterhin sichergestellt, dass alle vorhandenen Gegenbeispiele, welche im Referenztrace enthalten sind, gefunden werden. Daher kann davon ausgegangen werden, dass der L^*-Algorithmus in seiner hier vorgestellten Realisierung das Referenztrace bis zur maximal getesteten Wortlänge optimal abbildet. Optimal bedeutet in diesem Fall, dass er nicht weniger und

nicht mehr Zustände findet, als das Referenztrace tatsächlich beschreibt. Daher kann ein Fehlalarm des erlernten Automaten in Bezug auf das Referenztrace ausgeschlossen werden.

Die einzige Fehlerquelle beim Einsatz des L^*-Algorithmus stellt demzufolge eine mögliche Überspezialisierung der erlernten Automaten dar. Eine Überspezialisierung kann hier nur durch zu geringe Beschreibung des möglichen zugrunde liegenden Systems im Referenztrace herrühren. In diesem Fall wäre ein geprüftes Trace, welches nicht akzeptiert wird, allerdings kein Fehlalarm, sondern ein bisher nicht beobachtetes Systemverhalten.

Es kann somit davon ausgegangen werden, dass der L^*-Algorithmus bei der Erstellung eines Automaten eines abhängigen Alphabetes keine Fehlalarme produziert. Dies entspricht auch den Beobachtungen bei der Evaluierung des L^*-Algorithmus mit künstlichen Daten.

Bei der Evaluierung mit künstlichen Daten wurde immer von einem perfekten abhängigen Alphabet ausgegangen, da stets nur Zeichen eines einzelnen Handlungsstranges betrachtet wurden. Wird dies auf die Problemstellung mehrerer paralleler Handlungsstränge im Trace bezogen, kann die Aussage, dass L^* keine Fehlalarme produziert, nur unter der Annahme eines perfekten Clustering-Ergebnisses getroffen werden. Bei der Suche nach parallelen Handlungssträngen und damit verbundenen abhängigen Alphabeten wird es aber zu teilweise unvollständigen oder fehlerhaften Ergebnissen kommen. Dies resultiert aus der heuristischen Natur der eingesetzten Clustering-Verfahren und der mangelnden Überprüfbarkeit der Clustering-Ergebnisse.

Somit liegt die Schlussfolgerung nahe, dass die Ursache für Fehlalarme aus fehlerhaften Clustering-Ergebnissen resultiert. Diese fehlerhaften Clustering-Ergebnisse enthalten dann Events aus mehreren Handlungssträngen. Zur Bestimmung von potenziellen Fehlalarmen ist es demzufolge notwendig, herauszufinden, ob der Automat, welcher ein zu prüfendes Trace nicht akzeptiert, von einem fehlerhaft geclusterten Subalphabet erlernt wurde. Die Beantwortung der Frage, welche gefundenen Subalphabete nicht abhängige Events enthalten, ist allerdings auf Basis des Referenztraces nicht möglich. Daher müssen andere Informationsquellen herangezogen werden, um diese Frage beantworten zu können. Im nächsten Kapitel wird daher ein Verfahren vorgestellt, welches durch den Vergleich der Ergebnisse aus mehreren geprüften Traces versucht, schlechte Clustering-Ergebnisse und somit überspezialisierte Automaten zu identifizieren.

5.2.2 Verfahren zur Verringerung von Fehlalarmen

Wie bereits in Kapitel 3.4 gezeigt können mit dem L^*-Algorithmus nur Automaten geringer Komplexität effizient gelernt werden. Die Wahrscheinlichkeit, dass der L^*-Algorithmus für zufällige Event-Reihenfolgen einen Automaten erlernen kann, ist im Vergleich zu abhängigen Events sehr gering. Dies resultiert aus der hohen Komplexität, die ein Automat aufweisen muss, wenn er eine zufällige Event-Reihenfolge beschreiben muss (zufällige Event-Reihenfolgen entsprechen nicht abhängigen Alphabeten). Es kann aber trotzdem vorkommen, dass im Referenztrace auch nicht abhängige Events so zueinander positioniert sind, dass trotzdem ein sinnvoller Automat erlernt werden kann.

Im Falle einer solchen Konstellation versagt der eingeführte Bewertungsmechanismus auf Basis des L^*-Algorithmus. Wenn aber die in einem solchen Automaten abgebildeten Events tatsächlich unabhängig zueinander sind, so wird dieser Automat mit sehr großer Wahrscheinlichkeit alle weiter zu prüfenden Traces nicht akzeptieren, da er keine zufälligen Event-Reihenfolgen akzeptieren kann. Bei nicht abhängigen Subalphabeten ist die Wahrscheinlichkeit, dass mehrmals hintereinander ähnliche Event-Konstellationen wie im Referenztrace auftreten, äußerst gering, sodass dies auf falsch geclusterte Subalphabete hindeutet. Der erlernte Automat aus den unabhängigen Events ist in diesem Fall in gewisser Weise auf das Referenztrace überspezialisiert.

Es sollte daher unter Zuhilfenahme weiterer Traces möglich sein, herauszufinden, ob ein Automat ein nicht abhängiges Subalphabet enthält. Akzeptiert ein Automat keines der zu prüfenden Traces, so ist dieser mit sehr hoher Wahrscheinlichkeit auf das Referenztrace überspezialisiert und enthält ein nicht abhängiges Alphabet. Bei Anwendung dieser Erkenntnis auf das gegebene Problem sind diejenigen Automaten bzw. Subalphabete, welche mindestens ein weiteres zu prüfendes Trace akzeptieren, mit großer Wahrscheinlichkeit korrekte Automaten. Für alle weiteren Automaten ist die Wahrscheinlichkeit einer Überspezialisierung und somit eines Fehlalarmes demzufolge entsprechend groß.

Um die Anzahl der möglichen Fehlalarme möglichst klein zu halten, werden daher nur Automaten in die Erzeugung der OoN-Indikation einbezogen, welche mindestens ein weiteres zu prüfendes Trace akzeptiert haben.

Dieser Prozess stellt nach der Bewertung mittels des L^*-Algorithmus zur Auswahl der Subalphabete einen weiteren Konsolidierungsschritt der gefundenen Subalphabete und erlernten Automaten dar. Durch die vorhandene Überdeckung des Alphabets mittels der Kombination mehrerer Clustering-Verfahren ist es ohne einen großen Qualitätsverlust für die OoN-Erkennung möglich, einzelne gefundene Subalphabete auf diese Weise zu entfernen. Somit ergeben sich zwei Konsolidierungsschritte für die mittels Clustering identifizierten Subalphabete (Abbildung 5-2).

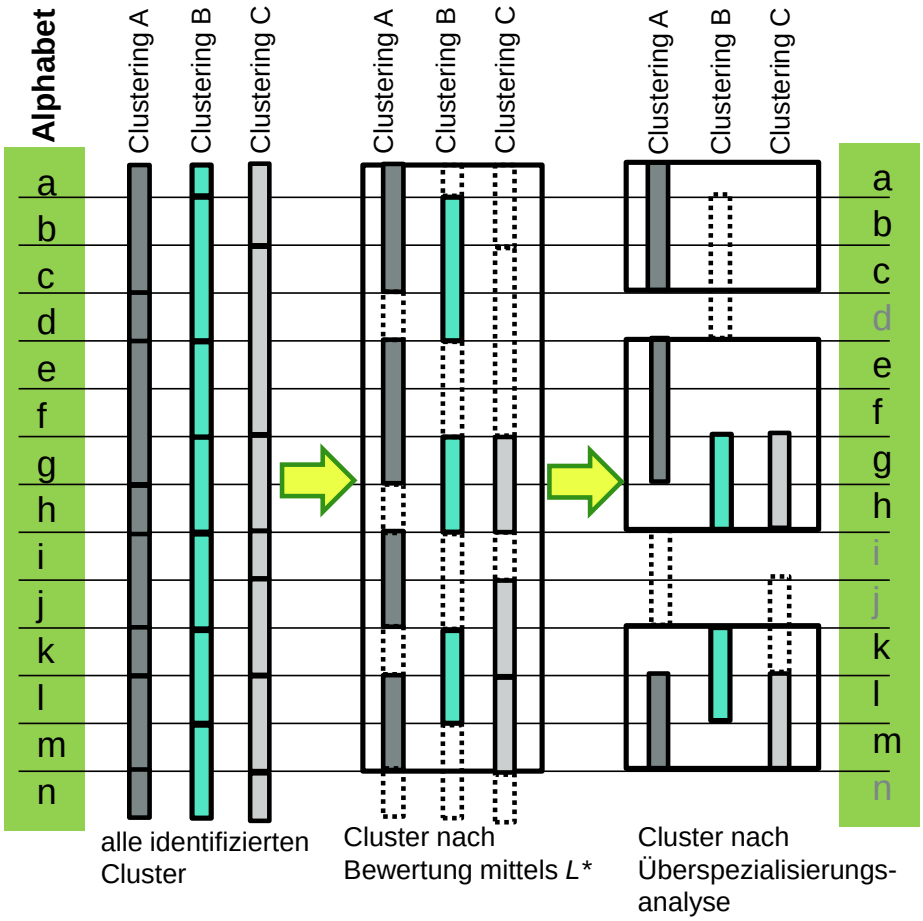

Abbildung 5-2 Schema der Konsolidierungsschritte für gefundene Subalphabete und der resultierenden Überdeckung des Alphabets

Der hier beschriebene Verfahrensschritt stellt eine Erweiterung der oben beschriebenen Anwendungsphase dar, da die zu prüfenden Traces allen Automaten zur Akzeptanzprüfung vorgelegt werden. Erst nach der Prüfung mehrerer Traces können zuverlässige Aussagen für eine entsprechende Bewertung einzelner Automaten und somit zur Verringerung der Fehlalarmrate getroffen werden.

5.2.3 Evaluierung

An dieser Stelle wird das verbesserte Verfahren zur Verringerung von Fehlalarmen evaluiert. Zu diesem Zweck werden die Traces in der Anwendungsphase den gelernten Automaten zur Akzeptanzprüfung vorgelegt. Zur Evaluierung werden vier weitere Traces des gleichen Fahrzeuges zur Prüfung herangezogen. Die Aufzeichnungsdauer im Vergleich zum Referenztrace ist in Abbildung 5-3 dargestellt.

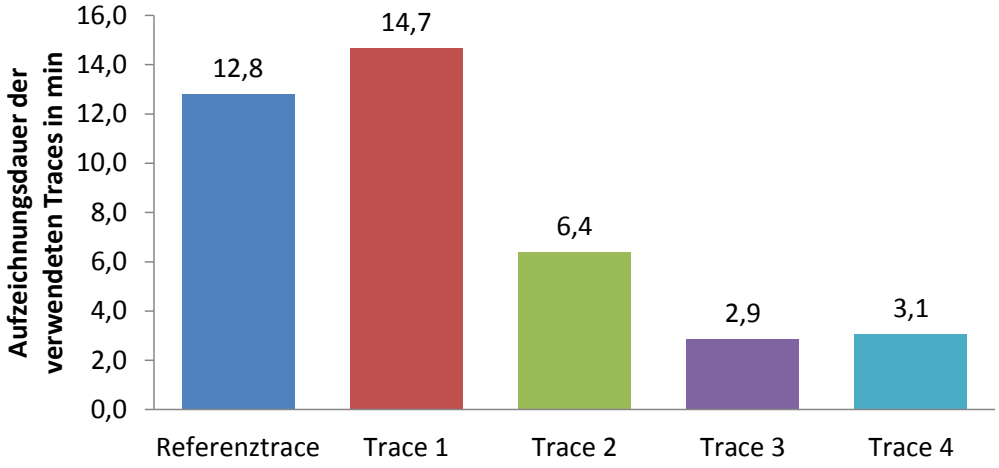

Abbildung 5-3 Aufzeichnungsdauer der verwendeten Traces (Referenztrace und zu prüfende Traces 1-4)

Bei der Überspezialisierungsanalyse der Automaten aus den frequenzbasierten Clustering-Verfahren sind im Schnitt 65 % der erstellten Automaten überspezialisiert. Wie in Abbildung 5-4 zu sehen ist, geht dabei aber die Abdeckung des Alphabetes und des Referenztraces in der Vereinigungsmenge nur um ca. 50 % zurück.

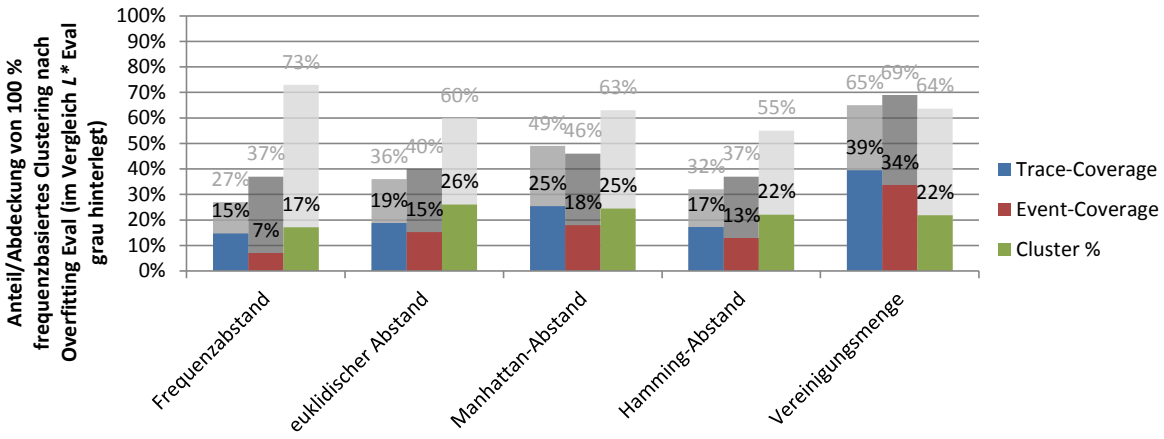

Abbildung 5-4 Darstellung der Vereinigungsmengen der frequenzbasierten Clustering-Verfahren nach L^*-Evaluierung (grau hinterlegt) und weiterführender Überspezialisierungsanalyse mit vier weiteren Traces

Die Ergebnisse der Überspezialisierungsanalyse für das evolutionsbasierte Clustering sind in Abbildung 5-5 dargestellt. Auffällig ist vor allem, dass für die *EA-Clust-V2*-Variante alle erstellten Automaten verworfen worden. Somit ist die Parametrisierung *EA-Clust-V2* des evolutionären Algorithmus als nicht brauchbar einzustufen.

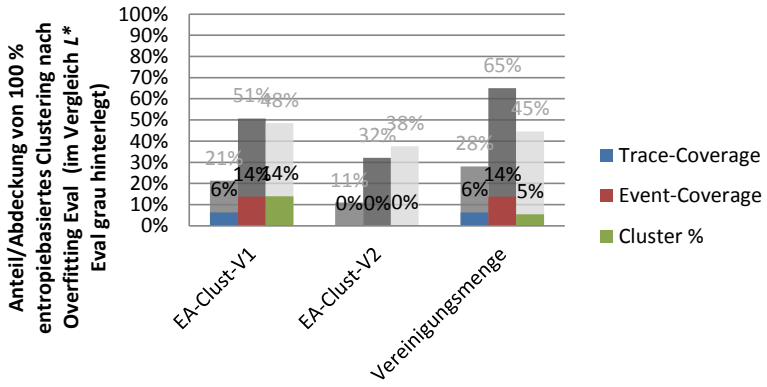

Abbildung 5-5 Darstellung der Vereinigungsmengen der entropiebasierten Clustering-Verfahren nach L^*-Evaluierung (grau hinterlegt) und weiterführender Überspezialisierungsanalyse mit vier weiteren Traces

Die zusammengeführten Ergebnisse der Überspezialisierungsanalyse für alle Clustering-Verfahren sind in Abbildung 5-6 dargestellt. Nach der Überprüfung von 2.848 erlernten Automaten mittels vier weiterer Traces sind in diesem Fall insgesamt noch 770 Automaten als anwendbar für die OoN-Erkennung anzusehen. Es wurde also im Zuge der Überspezialisierungsanalyse eine große Menge an erlernten Automaten verworfen. Es stehen nun noch ca. 25 % der nach der L^*-Bewertung für gut befundenen Automaten zur Verfügung. Trotzdem hat sich die Abdeckung des Referenztraces nur um ca. 50 % reduziert. Dies resultiert daraus, dass die ursprünglich sehr hohe Überdeckungsrate, bei der jedes Zeichen im Mittel von 2,8 Automaten überwacht wird, auf eine mittlere Überdeckung von 1,5 Automaten je Zeichen schrumpft. Die verbleibenden 770 Automaten weisen noch eine Event-Coverage von 45 % und eine Trace-Coverage von 44 % für das Referenztraces auf.

Um die Wirksamkeit der Überspezialisierungsanalyse zu zeigen, ist in Abbildung 5-6 der Verlauf der ausgelösten Trigger für zwei ausgewählte Traces dargestellt. Es ist gut zu sehen, dass bereits nach der Plausibilisierung mit einem weiteren Trace die Anzahl der OoN-Indikationen signifikant abfällt. So werden beispielsweise für Trace 1 1.943 Indikationen ausgelöst. Nach der Plausibilisierung mit Trace 2 kann aufgrund der Überspezialisierungsanalyse für 90 % dieser Trigger mit hoher Wahrscheinlichkeit ein Fehlalarm angenommen werden, so dass noch 181 auszuwertende OoN-Indikationen (Trigger) vorliegen. Werden zusätzlich noch die Traces 3 und 4 zur Plausibilisierung benutzt, so verringert sich die Zahl der auszuwertenden OoN-Indikationen auf 114 Trigger. Somit kann für dieses Beispiel bei der Plausibilisierung mittels drei weiterer Traces mit einer Fehlalarmrate von ca. 40 % gerechnet werden.

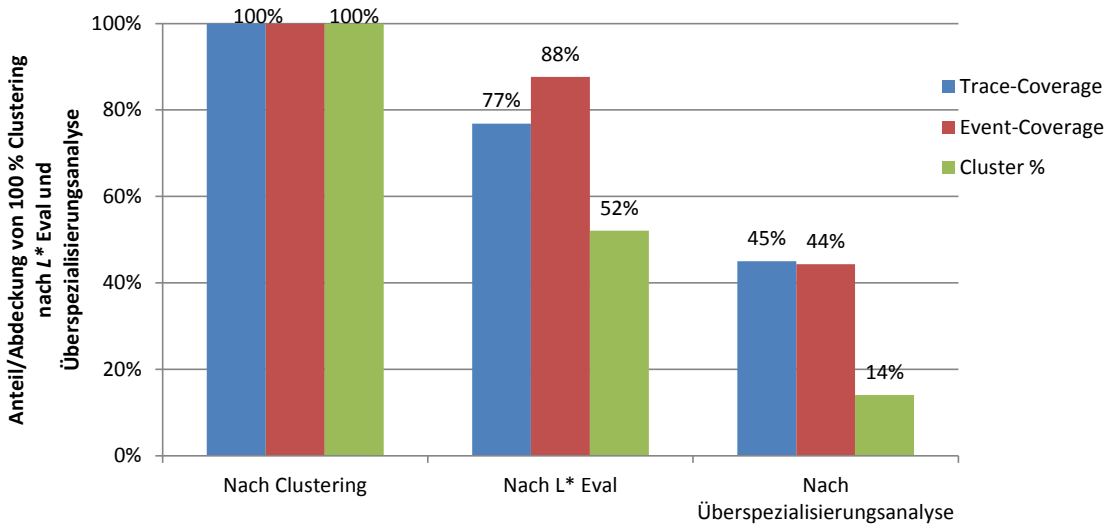

Abbildung 5-6 Entwicklung der Systemabdeckung der Vereinigungsmengen aller Clustering-Verfahren jeweils nach dem Clustering, nach der Evaluierung mittels L^* und nach der Überspezialisierungsanalyse

5.3 Bestimmung der verbleibenden Fehlalarmrate

Die Bestimmung der verbleibenden Fehlalarmrate anhand der vorliegenden Testdaten ist nicht ohne weiteres möglich. Dies rührt daher, dass für das untersuchte System keine Spezifikation zur Verfügung steht und somit keine gesicherten Aussagen über funktionale Zusammenhänge zwischen Netzwerkdaten und Systemverhalten getroffen werden können. Somit kann auf dieser Basis nicht entschieden werden, ob es sich um einen Fehlalarm handelt oder um ein neues, bisher nicht beobachtetes Verhalten.

Es ist daher nur möglich, eine Abschätzung der verbleibenden Fehlalarmrate zu geben. Bei Betrachtung der Entwicklung des Plausibilisierungsvorganges der einzelnen geprüften Traces im Zuge der Überspezialisierungsanalyse (Kapitel 5.2.3) ist zumindest eine gewisse Konsolidierung der Anzahl der verbleibenden OoN-Indikationen festzustellen. Die Hinzunahme weiterer Traces zur Plausibilisierung reduziert die Anzahl der mutmaßlich überspezialisierten Automaten zunehmend weniger. Da jeder überspezialisierte Automat einen Fehlalarm auslösen wird, ist auch von einer geringer werdenden Anzahl an Fehlalarmen durch die verbleibenden Automaten zu rechnen.

Wie exemplarisch in Abbildung 5-7 und Abbildung 5-8 dargestellt, ist nach der Plausibilisierung mit drei weiteren Traces für alle Traces eine Reduzierung der Fehlalarme im Vergleich zu den OoN-Indikationen ohne die hier vorgestellte Optimierung mit einem weiteren Trace um 90 % und mit drei weiteren Traces um weitere 30 bis 40 % zu beobachten. Aufgrund der geringen Verbesserung durch die Plausibilisierung mit mehr als einem Trace kann auf eine etwaige verbleibende Fehlalarmrate von ca. 35 % geschlossen werden.

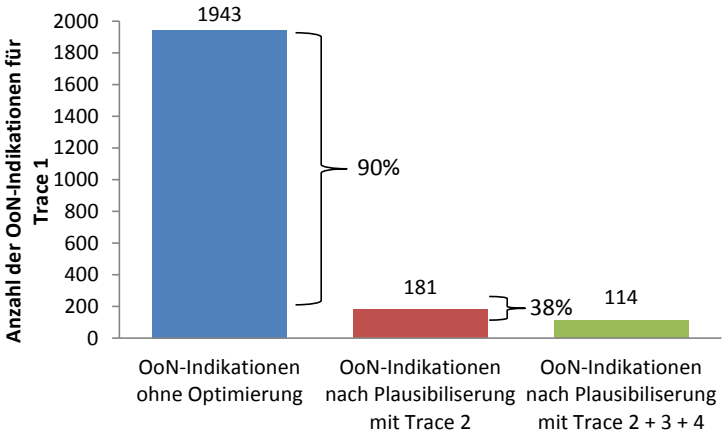

Abbildung 5-7 Verlauf der Anzahl der OoN-Indikationen für Trace 1 nach der Plausibilisierung mit Trace 2 und der Plausibilisierung mit den Traces 2, 3 und 4 (die Prozentangaben beziehen sich auf die Verringerung der Fehlalarme)

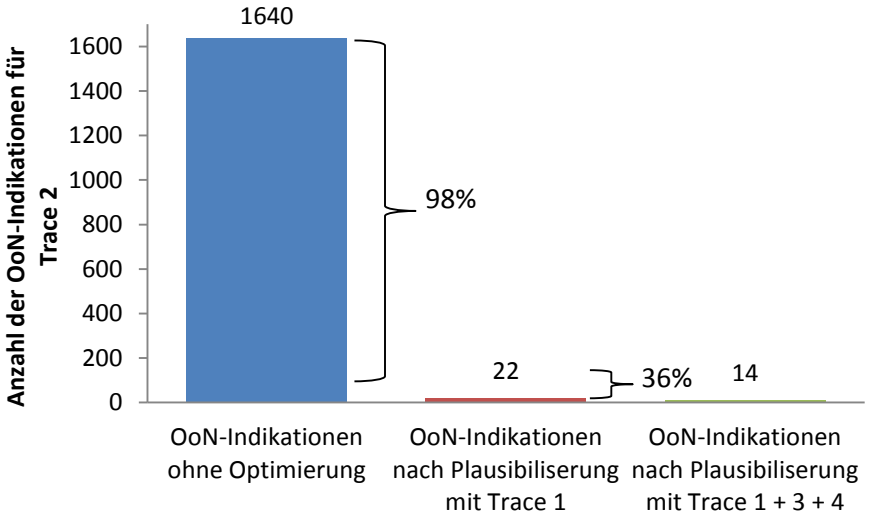

Abbildung 5-8 Verlauf der Anzahl der OoN-Indikationen für Trace 2 nach der Plausibilisierung mit Trace 1 und der Plausibilisierung mit den Traces 1, 3 und 4 (die Prozentangaben beziehen sich auf die Verringerung der Fehlalarme)

5.4 Bestimmung der Erkennungsrate tatsächlicher Abweichungen

Besitzt das zu prüfende Trace eine tatsächliche funktionale Abweichung in Bezug auf das Referenztrace, so muss das zu prüfende Trace von der OoN-Erkennung als „nicht der Norm entsprechend" gekennzeichnet werden. Dies geschieht dadurch, dass das zu prüfende Trace von mindestens einem gelernten Automaten nicht akzeptiert wird.

Die Eigenschaften der tatsächlich erkennbaren Abweichungen hängen im Wesentlichen von den abbildbaren Eigenschaften eines diskreten endlichen Automaten (DEA) ab. Für Eigenschaften, welche die gelernten Automaten nicht abbilden können, wird es auch nicht möglich sein, Abweichungen zwischen Referenz und zu prüfendem Trace zu

erkennen. Im Folgenden werden die Eigenschaften, welche von einem DEA nicht abbildbar sind, kurz aufgezählt und erläutert:

- Zähler von Schleifen: Ein DEA, wie er hier verwendet wird, kann zwar Schleifen abbilden, aber keine Zähler. Somit ist es nicht möglich, das Über- bzw. Unterschreiten bestimmter Wiederholungen zu erkennen.

- Quantitatives Verhalten: Bei der Nutzung eines DEA ist es nicht möglich, die Wahrscheinlichkeit einer bestimmten Transition anzugeben. Somit ist eine statistische Bewertung eines Traces nicht möglich. Einfache Bewertungen wären z. B., wie häufig ein Event auftritt oder ob ein bestimmter Pfad durch den Automaten häufiger oder seltener durchlaufen wird.

- Langzeitabhängigkeiten: Eine durch das Lernverfahren bedingte Limitierung ergibt sich aus der bereits diskutierten Abbildung von Langzeitabhängigkeiten. Durch das Lernverfahren kann eine Abbildung bis zur maximal getesteten Wortlänge garantiert werden. Abhängigkeiten, welche darüber hinausgehen, werden nur eingeschränkt berücksichtigt.

- Nicht sichtbare Ereignisse: Eine systembedingte Limitierung ergibt sich aus der Beschränkung der Informationen, welche ein Trace enthält. So sind im Referenztrace viele von außen einwirkende Ereignisse oder Einflüsse nicht sichtbar bzw. nicht enthalten. Demnach können diese Informationen auch nicht in die erstellten Überwachungsmodelle einfließen.

Die Einschränkungen, welche sich aus der limitierten Erkennung von Langzeitabhängigkeiten ergeben, wurden bereits in Kapitel 3.2.2.2 diskutiert. Alle weiteren genannten Einschränkungen werden an dieser Stelle nicht evaluiert, da diese verfahrensbedingt nicht abbildbar sind.

Vor dem Hintergrund der Anwendung des Verfahrens für eventbasierte Kontrollsysteme sollte die Frage nach der Erkennung von Abweichungen im Ausführungsgraphen des Systems näher betrachtet werden. Der Ausführungsgraph ist der Weg, welcher innerhalb des Automaten abgeschritten wird. Häufig wird dies auch als Kontrollfluss bezeichnet.

Die Erkennung von Abweichungen im Ausführungsgraphen ist abhängig vom Aufbau des ausgeführten Systems bzw. dessen resultierenden Überwachungsautomaten. Ein Überwachungsautomat bildet die Summe aller möglichen Ausführungsgraphen ab. Im Folgenden wird gezeigt, dass jede Änderung eines gültigen Ausführungsgraphen, welche nicht wiederum zu einem gültigen Ausführungsgraphen führt, vom Überwachungsautomaten erkannt wird. In Bezug auf die in Kapitel 3.3.3.1 mit der Definition 3-5 eingeführten Nomenklatur stellt ein Ausführungsgraph ein gültiges Wort einer Sprache dar. Diese Sprache wird vom Überwachungsautomaten beschrieben. Eine Abweichung ist nicht erkennbar, wenn die Änderung eines gültigen Wortes wiederum zu einem gültigen Wort des Überwachungsautomaten führt. Führt die Änderung zu einem Wort, welches nicht zur Sprache gehört, so wird diese Abweichung vom Überwachungsautomaten erkannt werden.

Die Wahrscheinlichkeit, dass eine Änderung im Ausführungsgraphen vom Überwachungsautomaten erkannt wird, kann im vorliegenden Fall allerdings nicht direkt berechnet werden. Dies hängt vor allem damit zusammen, dass ein Trace potenziell mehrere Wörter enthalten kann und dass durch die Abbildung von Schleifen die Länge der Wörter beliebig variieren kann. Somit ist die Wahrscheinlichkeit der Erkennung einer Abweichung im Ausführungsgraphen abhängig von dem Zustand, in dem sich der Überwachungsautomat befindet, und von der Struktur des Überwachungsautomaten selbst.

Ein einfaches Beispiel soll diese Aussage verdeutlichen: Wird in einer Schleife mit mehreren aneinandergereihten gleichen Zeichen ein Zeichen entfernt oder hinzugefügt, so wird dies der Überwachungsautomat nicht erkennen können. Wurde die Schleife aber nur einmal durchlaufen und wird im Trace genau dieses eine Zeichen entfernt, so ist die Wahrscheinlichkeit sehr groß, dass das komplette Fehlen des Zeichens erkannt wird.

5.4.1 Evaluierungsmethodik zur Bestimmung der Erkennungsrate tatsächlicher Abweichungen

Um einen Eindruck über die Wahrscheinlichkeit der Erkennung der Abweichung vom Ausführungsgraphen zu bekommen, wird eine empirische Untersuchung durchgeführt, in der das Referenztrace in unterschiedlicher Weise manipuliert wird.

Nach der Lernphase wird das Referenztrace von allen erlernten Automaten vollständig akzeptiert. Werden nun gezielt Veränderungen am Referenztrace vorgenommen, so müssten dies die erlernten Automaten als Abweichung erkennen. Wird dieses manipulierte Referenztrace in der Anwendungsphase der selbstlernenden OoN-Erkennungsverfahrens geprüft, so sollte entsprechend für jede Manipulation eine OoN-Indikation ausgelöst werden.

Zur Bestimmung der Erkennungsrate tatsächlicher Abweichungen werden im Referenztrace drei verschiedene Manipulationen an einzelnen Zeichen durchgeführt. Diese sind:

(1) Ein zufällig gewähltes Zeichen wird vom Trace entfernt (Entfernung).

(2) Ein zufällig gewähltes Zeichen wird verdoppelt (Verdopplung).

(3) Ein zufällig gewähltes Zeichen wird durch ein anderes zufällig gewähltes Zeichen des Alphabetes ersetzt (Ersetzung).

Es wird dazu jeweils beim Referenztrace 1 % der Zeichen zufällig ausgewählt und in einer der drei oben genannten Varianten modifiziert. Bei einer Länge von ca. $2{,}5*10^6$ Zeichen werden demnach pro Durchgang im Schnitt 2.500 Manipulationen durchgeführt. Anschließend wird das modifizierte Referenztrace den gelernten Automaten zur Akzeptanzprüfung vorgelegt.

Wird von einem Automaten eine Abweichung erkannt, so wird dies protokolliert. Anschließend wird der Automat in den Startzustand versetzt und die Prüfung des Traces fortgesetzt. Nach dem Durchlauf werden die protokollierten Manipulationen mit den

protokollierten Ergebnissen der festgestellten OoN-Indikationen verglichen. Dieser Vorgang wird für jedes der drei Manipulationsverfahren viermal durchgeführt und die Erkennungsraten werden jeweils gemittelt, um statistische Abweichungen zu vermeiden.

Die Erkenntnisse aus der Anwendung des so präparierten zu prüfenden Traces sollen eine Abschätzung der Größenordnung der Richtig-Negativ-Rate der Klassifizierung im Sinne der OoN-Erkennung ermöglichen. Die Richtig-Negativ-Rate ist nach Gleichung 2-25 das Äquivalent zur Bestimmung der tatsächlich gefundenen Abweichungen.

5.4.2 Evaluierungsergebnisse

Die Ergebnisse für die oben beschriebene Evaluierungsmethodik zur Bestimmung der Erkennungsrate für tatsächliche Abweichungen sind in Tabelle 5-1 dargestellt. Es werden in allen Versuchen über 90 % der beschriebenen Manipulationen am Referenztrace entdeckt. Insgesamt wurden pro Manipulationsmethode jeweils ca. 10.000 Änderungen am Referenztrace durchgeführt.

Tabelle 5-1 Ergebnisse der Bestimmung der Erkennungsrate für tatsächliche Abweichungen

Manipulationsverfahren	(1) Entfernung	(2) Verdopplung	(3) Ersetzung
Erkennungsrate	94 %	95 %	91 %

5.5 Fazit

Das Kapitel beschreibt und evaluiert den Einsatz der erlernten Akzeptorautomaten in der Anwendungsphase zur Prüfung weiterer Traces. Die Evaluierung findet anhand realer Netzwerkdaten und mittels Injizierung von Fehlern in das Referenztrace statt. Evaluiert werden die in Kapitel 2.3.5 festgelegten Kriterien für die Anwendung des selbstlernenden OoN-Erkennungsverfahren. Der besondere Schwerpunkt liegt dabei auf der Bestimmung der Fehlalarmrate und der Erkennungsrate tatsächlicher Abweichungen.

Es zeigt sich, dass ohne eine weitere Plausibilisierung der erlernten Automaten die Fehlarmrate durch auf das Referenztrace überspezialisierte Automaten bei über 90 % aller ausgelösten OoN-Indikationen liegen würde. Da die Fehlalarmrate aber eine besonders kritische Größe für die Anwendung des selbstlernenden OoN-Erkennungsverfahrens ist, wird ein weiterer Verfahrensschritt, die Überspezialisierungsanalyse, zur Verringerung der Fehlalarmrate eingeführt.

Mit der Überspezialisierungsanalyse kann die Fehlalarmrate durch die Plausibilisierung der Ergebnisse aus mehreren geprüften Traces auf ca. 30 bis 40 % gesenkt werden. Die Ergebnisse der Evaluierung der Erkennungsrate tatsächlicher Abweichungen zeigen, dass die erlernten Automaten mit einer Quote von über 90 % Abweichungen im Verhalten erkennen können.

6 Abschließende Diskussion des selbstlernenden OoN-Erkennungsverfahrens

Nachdem alle nötigen Einzelschritte des selbstlernenden OoN-Erkennungsverfahrens vorgestellt und evaluiert worden sind, wird an dieser Stelle eine abschließende Bewertung und Diskussion des Verfahrens vorgenommen. Dazu werden die bisher gewonnen Ergebnisse zusammengefasst und im Hinblick auf die Anwendbarkeit der OoN-Erkennung ausgewertet.

6.1 Zusammenfassung der Verfahrensschritte

Das hier vorgestellte OoN-Erkennungsverfahren gliedert sich in drei Verarbeitungsschritte. Diese sind (i) die Identifikation von parallelen Handlungssträngen mittels Clustering-Verfahren (Kapitel 4), (ii) das Lernen von Abhängigkeitsmodellen (Kapitel 3) in der Lernphase und (iii) die Akzeptanzprüfung (Kapitel 5.1) in der Anwendungsphase.

Es wurden mehrere Clustering-Verfahren vorgestellt, welche es ermöglichen, Subtraces aus dem Referenztrace zu extrahieren. Durch die Kombination der Ergebnisse mehrerer Clustering-Verfahren wird die Abdeckung des bewertbaren Systemverhaltens signifikant verbessert. Von den angepassten und evaluierten Lernverfahren wird der L^*-Algorithmus gewählt, um aus den erstellten Subtraces Akzeptorautomaten als Abhängigkeitsmodelle zu erstellen. In der Akzeptanzprüfung werden die erstellten Akzeptorautomaten verwendet, um neue Traces bezüglich ihres Verhaltens im Vergleich zum Referenztrace zu prüfen. Mit der vorgestellten Plausibilisierung der OoN-Indikationen wird die Fehlalarmrate für die OoN-Indikation signifikant reduziert.

6.2 Grundlage der durchgeführten Bewertung

Zur Durchführung der Evaluierung konnte in dieser Arbeit nicht auf reale Netzwerkdaten mit bekannten Abweichungen und entsprechende Referenzdaten zurückgegriffen werden. Derartige Daten fallen zwar stets bei der Durchführung von Systemtests und Regressionstests an, werden aber aus Platzgründen nicht vollständig gespeichert. Somit wäre eine Integration des vorgestellten selbstlernenden OoN-Erkennungsverfahrens in einen produktiven Entwicklungs- bzw. Testbetrieb notwendig, um eine erschöpfende Evaluierung des Verfahrens durchzuführen. Da dies im Rahmen der Arbeit nicht möglich ist, werden die entsprechenden Kenndaten mittels anderweitig verfügbarer Daten ermittelt.

Mit Hilfe künstlich erzeugter Daten aus vorgegebenen und zufällig erstellten Automaten wurde die Evaluierung des L^*-Algorithmus durchgeführt. Eine sehr realitätsnahe, verfügbare Datenbasis stellt die Sammlung mehrerer realer Netzwerkaufzeichnungen eines Fahrzeug-CAN dar (Kapitel 3.4 und Kapitel 5.2.3). Von diesen Datensätzen ist bekannt, dass sie vom gleichen System in unterschiedlichen Umgebungsbedingungen (Fahrsituationen) erzeugt wurden. Das System wies zum Zeitpunkt der Aufzeichnung keine erkennbaren Fehlfunktionen auf, so dass jede der Aufzeichnungen als Referenztrace verwendet werden kann.

6.3 Qualitative Eigenschaften der OoN-Erkennung

Das wichtigste qualitative Bewertungskriterium des selbstlernenden OoN-Erkennungsverfahrens wird über die Klassifizierungsgüte ermittelt (Kapitel 2.3.5). Die zwei anzu-

wendenden Kriterien sind dabei (i) die Fehlalarmrate (Falsch-Negativ-Rate, fn) und (ii) die Erkennungsrate von tatsächlichen Abweichungen (Richtig-Negativ-Rate, rn).

Die Fehlalarmrate ist ein sehr wichtiges Merkmal für eine Anwendung in einem produktiven Umfeld. Jede vermeintlich erkannte Abweichung bedarf einer manuellen Bewertung, um deren Ursache oder auch deren Richtigkeit herauszufinden. Ist die Anzahl der Fehlalarme im Verhältnis zu den tatsächlich gefundenen Abweichungen zu groß, so gehen die tatsächlichen Fehler in der Auswertung unter. Überschreitet die Anzahl der Fehlalarme eine gewisse Größenordnung, so ist die Anwendbarkeit des Verfahrens gefährdet. Es gibt eine starke Wechselwirkung zwischen der Fehlalarmrate und der Erkennungsrate tatsächlicher Abweichungen. Wird das Verfahren zu „tolerant" ausgelegt, so dass die Fehlalarmrate sinkt, wird in den meisten Fällen die Erkennungsrate tatsächlicher Abweichungen ebenfalls verkleinert.

Es kann gezeigt werden, dass das eingesetzte Lernverfahren, der L^*-Algorithmus, keine Fehlalarme bei einer optimalen Trennung des Alphabetes in abhängige Zeichen produziert (Kapitel 3.3.3). Die Quelle für Fehlalarme ist somit eine unscharfe Trennung der Subalphabete durch die eingesetzten Clustering-Verfahren. Die daraus resultierende Fehlalarmrate des OoN-Erkennungsverfahren wurde in Kapitel 5.3 anhand der Plausibilisierung mit weiteren zur Verfügung stehenden Referenzdaten, mit 35 bis 40 % abgeschätzt. Im Vergleich zu anderen automatischen bzw. heuristisch basierten Analyseverfahren wie z. B. einer Code-Analyse ist dies ein durchaus vertretbarer Wert (Kremenek u. a. 2004).

Die Erkennungsrate der tatsächlich gefundenen Abweichungen wurde in Kapitel 5.4 mittels der Injizierung von Fehlern auf 90 bis 95 % abgeschätzt. Im Vergleich zu Fehlererkennungsmechanismen, welche ebenfalls den Kontrollfluss von Programmen überwachen, ist dies ein recht guter Wert. So erreicht beispielsweise (Wolf 2013) eine Erkennungsrate von 64 % bei Veränderungen am Ausführungsgraphen mittels einer Instrumentierung des Binärcodes. Es sei an dieser Stelle aber angemerkt, dass eine Anwendung zur Fehlererkennung einen entsprechend größeren Fokus auf eine niedrige Fehlalarmrate legen muss, welche bei (Wolf 2013) unter 3 % liegt.

Als weiteres Kriterium ist auch die Erkennung von Langzeitabhängigkeiten zur Klassifizierungsgüte zu zählen (Kapitel 2.3.5). Wie in Kapitel 3.3.3 gezeigt, wird für die hier vorgestellte Realisierung des L^*-Algorithmus die Langzeitabhängigkeit mit Hilfe der maximalen Länge der Gegenbeispiele (maximal getestet Wortlänge) bestimmt. Für die Anwendung an den realen Fahrzeugdaten wurde diese auf sechs Zeichen festgelegt. Diese Eigenschaft hat einen Einfluss auf die Erkennung tatsächlicher Abweichungen. Es werden Abhängigkeiten zwischen einzelnen Traces nur bis zur getesteten Wortlänge aufgelöst, was die Erkennung von Abweichungen bei Langzeitabhängigkeiten limitiert. Wie groß dieser Einfluss ist, hängt entscheidend von der Komplexität des zugrunde liegenden Systems ab. Der eingesetzte L^*-Algorithmus schneidet zwar im Punkt Langzeitabhängigkeiten gegenüber den ebenfalls evaluierten künstlichen neuronalen Netzen weniger gut ab, hat aber dafür eine wesentlich bessere Performance und bessere Werte in puncto Fehlalarmrate (Kapitel 3.3.2).

6.4 Ergebnisse aus Evaluierung mit realen Anwendungsdaten

Neben den im vorhergehenden Kapitel erläuterten qualitativen Eigenschaften gibt es noch einige Eigenschaften des vorgestellten selbstlernenden OoN-Erkennungsverfahrens, welche eher eine Aussage über die Anwendbarkeit des Systems geben. Diese Eigenschaften sind z. B. die Systemabdeckung oder die absolute Anzahl der ausgelösten OoN-Indikationen. Diese Eigenschaften sind aber sehr stark von den vorliegenden Daten abhängig und basieren daher auch größtenteils auf Messungen, welche aus den gegebenen Referenzdaten entstanden sind. Da die verwendeten Referenzdaten auf realen Fahrzeugdaten beruhen, sollten die hier vorgestellten Ergebnisse einen Anhaltspunkt über zu erwartende Größenordnungen bei der Anwendung des OoN-Erkennungsverfahrens verschaffen.

Die Grundlage für die hier vorgestellten Ergebnisse bilden die fünf Traces eines Antriebs-CAN, wie sie bereits in Kapitel 5.2.3 zur Optimierung der Fehlalarmrate verwendet wurden. Ausgehend von einem Referenztrace, welches eine Fahrtdauer von ca. 13 min aufweist, werden entsprechende Akzeptorautomaten erlernt. Mit diesen Automaten werden vier weitere Traces mit einer Aufzeichnungsdauer von 3 bis 15 min geprüft.

Das Referenztrace hat nach der Vorverarbeitung 7.172 einzelne Events (Alphabetgröße) und eine Länge von ca. 2,5 $*10^6$ Events. Nach der Anwendung der Clustering-Verfahren sind 5.472 Subalphabete identifiziert worden. Diese Werte sind in Abbildung 6-1, links als 100 % aufgetragen. Nach der ersten Bewertung der Subalphabete mittels des L^*-Algorithmus sind noch ca. 50 % der ursprünglichen Subalphabete als brauchbar gekennzeichnet. Nach der Plausibilisierung der verbleibenden Subalphabete anhand weiterer geprüfter Traces (Überspezialisierungsanalyse) bleiben noch 14 % (770) der ursprünglich gefundenen Subalphabete übrig, um in der OoN-Erkennung verwendet zu werden. Diese 14 % der gefundenen Alphabete decken dabei aber ca. 45 % (Event-Coverage C_e = 45 %, Trace-Coverage C_τ = 44 %) des Referenztraces ab (Abbildung 6-1).

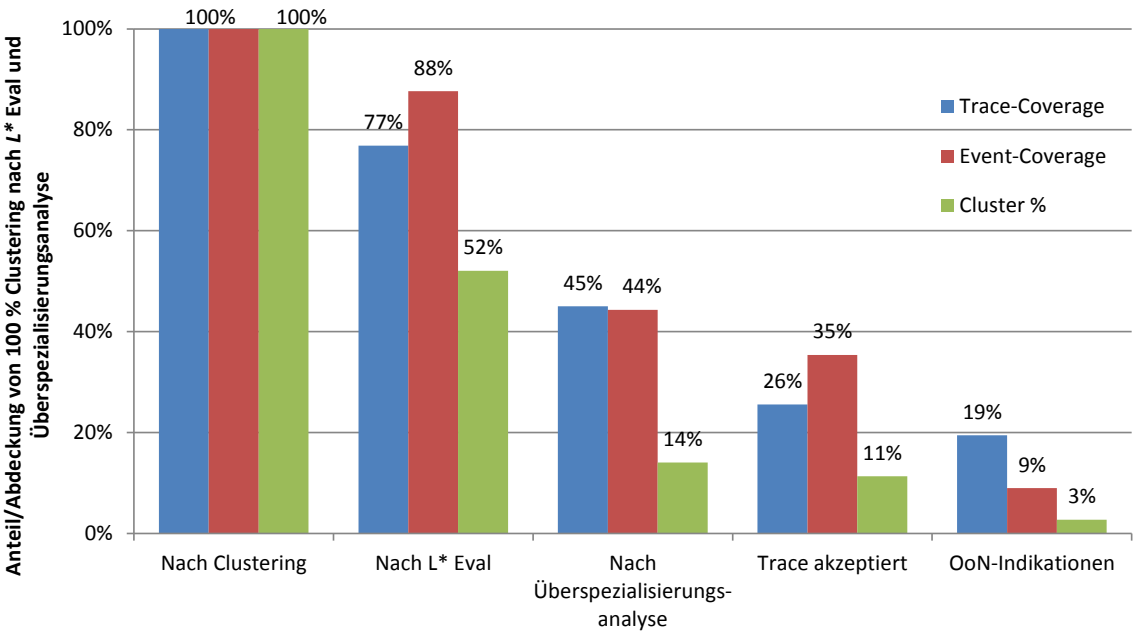

Abbildung 6-1 Anteil des Referenztraces, welches vom vorgestellten Verfahren bewertet werden kann (1.-3. Abschnitt), und der Anteil der Automaten, welche das Trace vollständig akzeptieren im Vergleich zu den ausgelösten OoN-Indikationen (Automaten, welche Trace nicht akzeptieren) (4. und 5. Abschnitt)

Ein Vergleich dieser Werte mit herkömmlichen Testverfahren zeigt, dass von einer 45 %igen Testabdeckung des Verhaltens des Referenztraces gesprochen werden kann. Leider liegen keine genauen Vergleichszahlen für die Testabdeckung des Netzwerkverkehrs bei Gesamtfahrzeugtests vor. Es ist aber davon auszugehen, dass manuell erstellte Testmodelle für Positivtests nur in Ausnahmefällen für einige wenige wichtige Funktionalitäten erstellt werden. Daher ist die Überwachung von annähernd der Hälfte des Netzwerkverkehrs als signifikanter Fortschritt zu werten.

Für alle der 770 validen Subalphabete stehen nach den beiden Plausibilisierungsschritten Akzeptorautomaten zur Verfügung, welche für die OoN-Erkennung verwendet werden. Diese 770 Automaten wurden zur OoN-Erkennung auf die vier verbleibenden zu prüfenden Traces mit einer Gesamtaufzeichnungszeit von 27 min angewendet. Bei der Überprüfung wurden 149 OoN-Indikationen ausgelöst. Demgegenüber stehen 621 Automaten, welche das Verhalten aller vier geprüften Traces akzeptiert haben (Abbildung 6-2).

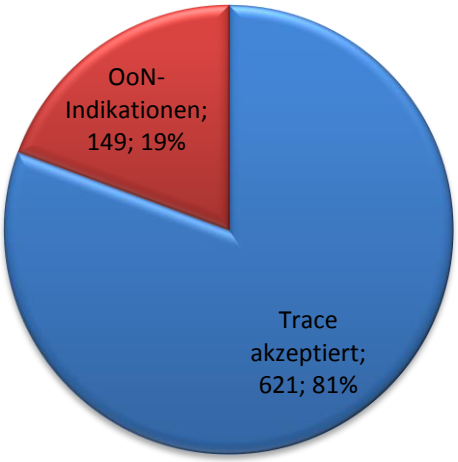

Abbildung 6-2 Anteil der Automaten, welche eine OoN-Indikation ausgelöst haben, im Vergleich zu Automaten, welche ein Normverhalten festgestellt haben

Somit wird in 149 Fällen eine Abweichung vom Referenztrace festgestellt. Dies entspricht in etwa 19 % des insgesamt bewerteten Verhaltens. Diese Zahl erscheint in ihrer Größenordnung recht realistisch. Ein Grund dafür ist die relativ kurze Aufzeichnungszeit des Referenztraces mit nur etwa 45 % der Aufzeichnungszeit im Vergleich zu den zu prüfenden Traces. Weiterhin wurden keine speziellen Testszenarien bei der Aufnahme des Referenztraces absolviert, die vier zu prüfenden Traces beinhalten aber durchaus unterschiedliche Fahrtszenarien im Vergleich zum Referenztrace.

Würde es sich um einen realen Anwendungsfall handeln, müsste in 149 Fällen geprüft werden, ob die Abweichungen vom Referenztrace ein erlaubtes Verhalten darstellen oder als fehlerhaft eingestuft werden müssen. Dies ist für die insgesamt 27 Minuten Fahrzeit aller fünf Traces immer noch ein recht hoher Wert, der aber im Wesentlichen auf ein zu kurzes Referenztrace (bzw. eine mangelnde Abdeckung des zulässigen Systemverhaltens im Referenztrace) hindeutet.

Eine sehr interessante Perspektive auf die eingesetzten Clustering-Verfahren ergibt sich, wenn die Anzahl der gefundenen Abweichungen auf das Clustering-Verfahren zurückgeführt wird, mit dessen Hilfe der Automat erstellt wurde, welche das Verhalten nicht akzeptiert. Wie in Abbildung 6-3 dargestellt, zeigt sich, dass das Clustering anhand des in dieser Arbeit definierten Frequenzabstandes fast die Hälfte aller gefundenen Abweichungen identifiziert und das entropiebasierte Clustering, obwohl es eine wesentlich geringere Systemabdeckung erzielt hat, für einen großen Anteil der OoN-Indikationen mit fast 25 % verantwortlich ist.

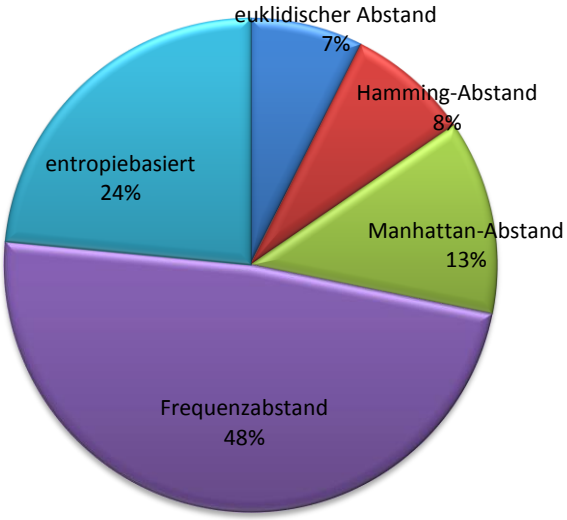

Abbildung 6-3 Aufteilung der OoN-Indikationen auf zugrunde liegende Clustering-Verfahren bzw. das benutzte Abstandsmaß

6.5 Rechenaufwand zur Durchführung des Verfahrens

Der Rechenaufwand zur Auswertung des Referenztraces wird im Wesentlichen durch die Clustering-Verfahren und die Erstellung der Akzeptorautomaten verursacht. Für das vorliegende Referenztrace benötigt ein aktueller 4-Kernrechner mit 2 GHz ca. 15 bis 20 Stunden, um alle vorgestellten Verfahrensschritte durchzuführen. Dies ist im Bereich des Data-Mining oder auch Machine-Learning ein durchaus vertretbarer Aufwand, zumal bei der Entwicklung der benutzten Software keine besonderen Optimierungen diesbezüglich vorgenommen wurden. Hervorzuheben ist an dieser Stelle vor allem die gute Parallelisierbarkeit der Verfahrensschritte. Bei einer entsprechenden Optimierung der Implementierung und der Ausführung auf entsprechender Hardware kann eine Durchlaufzeit von unter einer Stunde durchaus als realistisch angesehen werden.

7 Schlussbetrachtung

In diesem Kapitel werden die Eigenschaften des entwickelten selbstlernenden OoN-Erkennungsverfahrens zusammengefasst. Anschließend wird in Form eines kurzen Ausblickes auf weitere Anwendungsmöglichkeiten sowie die Erweiterung des entwickelten Verfahrens und einzelner Arbeitsschritte eingegangen.

7.1 Zusammenfassung der Arbeit

Der Anwendungsfokus des in dieser Arbeit vorgestellten selbstlernenden OoN-Erkennungsverfahrens liegt in der Überprüfung des Kommunikationsverhaltens verteilter eingebetteter Systeme. Im Mittelpunkt steht das Kommunikationsverhalten von Steuergeräten im Automobil, welches mit dem vorgestellten Verfahren besser bewertbar gemacht werden soll.

Aufgrund der stark wachsenden Anzahl softwarebasierter Funktionen und der immer stärker werdenden Vernetzung dieser Funktionen wird es zunehmend schwieriger, das Verhalten des Systems vollständig zu spezifizieren und zu testen. Die Arbeit zeigt daher einen neuen Ansatz, um der Problematik unvollständiger oder fehlender Spezifikation und daraus resultierender mangelnde Bewertbarkeit bzw. Testbarkeit verteilter eingebetteter Systeme entgegenzuwirken. Zu diesem Zweck wird in der Arbeit ein Verfahren entwickelt, welches es erlaubt, Referenztraces als Ersatz für nicht vorhandene Spezifikation zu nutzen.

Das Ziel dieses Verfahrens ist es, Abweichungen von dem im Referenztrace beschriebenen Verhalten bei Tests auf Systemebene (z. B. bei Gesamtfahrzeugtests) festzustellen. Somit können Abweichungen vom Normalverhalten des Systems erkannt werden und das beobachtete Verhalten einer weiteren Prüfung unterzogen werden.

Den Schwerpunkt der Arbeit stellt die Extraktion von Verhaltensmustern aus einem Referenztrace ohne das Vorhandensein einer genauen Systembeschreibung dar. Diese so extrahierten Verhaltensbeschreibungen erlauben eine qualitative Bewertung des Kommunikationsverhaltens eingebetteter Systeme. Dies ist ein wesentlicher Fortschritt im Bereich selbstlernender Systembewertung. Bisherige Verfahren (z. B. Anomalie-Erkennung zur Einbruchserkennung bei klassischen IT-Systemen) sind ohne Vorwissen über das System ausschließlich in der Lage, quantitative Eigenschaften (insbesondere Häufigkeit, Frequenz) von Events als Normalverhalten zu erlernen. Andere selbstlernende Verfahren werden zur Verbesserung einer bestehenden Spezifikation angewandt und benötigen im Vorfeld ein tiefgreifendes detailliertes Systemverständnis. Im Unterschied dazu erlaubt das hier entwickelte Verfahren die Anwendung ohne Vorwissen über das ausgeführte System. Das entwickelte Verfahren bewertet Abhängigkeiten zwischen Events (qualitative Bewertung), wie sie beispielsweise durch Protokollautomaten vorgegeben werden könnten.

Ein wichtiges Merkmal des hier entwickelten Verfahrens ist eine direkte Anwendbarkeit innerhalb bestehender Fahrzeugtests, womit ein unmittelbarer Praxisbezug gegeben ist. Derzeit wird das Kommunikationsverhalten eines Fahrzeugs bei Testfahrten aufgezeichnet und einer Prüfung auf bekannte Fehlerfälle unterzogen. Eine Überprüfung des Sollverhaltens findet auf Netzwerkebene bei Testfahrten nicht statt. Der Ansatz des hier entwickelten Verfahrens besteht darin, die an dieser Stelle zur

Verfügung stehenden Traces einer weiterführenden selbstlernenden, automatischen Auswertung zuzuführen. Diese soll das Norm- oder Normalverhalten des zu analysierenden Systems erlernen und in einer späteren Phase Abweichungen vom Normverhalten erkennen. Daher wird das Verfahren als selbstlernendes Out-of-Norm-Erkennungsverfahren bezeichnet.

Um das Ziel eines selbstlernenden Verfahrens zu erreichen, werden in der Arbeit zunächst drei infrage kommende Lernverfahren auf ihre Eignung für die zu lösende Problemstellung untersucht: (i) Markov-Modelle, (ii) künstliche Neuronale Netze (in diesem Fall LSTM-Netze) und (iii) diskrete endliche Automaten (in diesem Fall Akzeptorautomaten). Während für die Anwendung der Markov-Modelle bereits Ergebnisse zur Anwendbarkeit auf die Problemstellung in der Literatur vorliegen, wurden sowohl für die LSTM-Netze als auch für den Angluin L^*-Algorithmus (als Lernverfahren für Automaten) die Anwendbarkeit auf das Problem in der Arbeit nachgewiesen. Hierzu wurden für diese Ansätze eigene Erweiterungen erstellt, um sie als Lernverfahren einsetzen zu können.

Aufgrund der sehr unterschiedlichen Wirkungsweisen der Lernverfahren wurde eine eigene Evaluierungsmethodik erstellt, welche speziell die Belange der vorgestellten Problemstellung berücksichtigt. Durch den Einsatz spezieller Referenzgrammatiken werden die wichtigsten Kriterien wie die Fehlalarmrate und die Erkennung von Langzeitabhängigkeiten vergleichbar gemacht. Dadurch ist es möglich, die Anwendbarkeit und die Performance der Lernverfahren im Bezug auf die Problemstellung genau zu bewerten.

Bei der Anwendung der Lernverfahren auf tatsächliche Fahrzeugdaten zeigt sich zunächst, dass die Lernaufgabe wesentlich komplexer ist, als in der vorher durchgeführten Evaluierung angenommen. Durch die Aufstellung einer Systemhypothese, welche auf der Annahme mehrerer paralleler Handlungsstränge in einem Referenztrace beruht, werden für das beobachtete Verhalten eine Erklärung gefunden und ein geeigneter Lösungsweg abgeleitet. Es kann gezeigt werden, dass die Komplexität der Lernaufgabe exponentiell wächst, wenn die beobachteten Events zu unterschiedlichen Handlungssträngen gehören, welche ihrerseits von unabhängigen Subautomaten beschrieben werden. Um die Komplexität auf ein für die Lernverfahren anwendbares Maß zu reduzieren, müssen die im Referenztrace beobachteten Events in kleinere Gruppen mit hoher Abhängigkeit unterteilt werden.

Die für dieses Problem zu lösende Herausforderung besteht in der Ermanglung adäquater Eigenschaften, welche die Zugehörigkeit eines Events zu einem Handlungsstrang kennzeichnen. Zu diesem Zweck werden zwei verschiedene Clustering-Verfahren entwickelt, bei denen durch die Anwendung der Systemhypothese entsprechende Merkmale generiert werden können. Das erste der vorgestellten Clustering-Verfahren baut auf der Annahme einer zeitlichen Abhängigkeit zwischen Events des gleichen Handlungsstranges auf. Das zweite Clustering-Verfahren versucht die informationstheoretische Entropie der Subalphabete zu optimieren.

Das Finden abhängiger Events mittels Clustering stellt einen signifikanten Fortschritt bei der maschinellen Analyse des Kommunikationsverhaltens eingebetteter Systeme dar. Durch die Anwendung der neu entwickelten Clustering-Verfahren wird ein effizientes Lernen von Modellen mittels des vorgeschlagenen Lernverfahrens auf Basis des Angluin

L^*-Algorithmus ermöglicht. Im Ergebnis der durchgeführten Evaluierung zeigt sich, dass so für über 80 % der Events eines realen Fahrzeug-CAN-Traces aussagekräftige Verhaltensmodelle in Form von diskreten endlichen Automaten (DEA) erlernt werden können.

Diese Verhaltensmodelle werden in der Anwendungsphase genutzt, um die eigentliche OoN-Erkennung bei weiteren zu prüfenden Netzwerktraces durchzuführen. Das Ergebnis dieser Prüfung sind OoN-Indikationen, welche auf potenzielle Abweichungen hinweisen. Zur Bewertung der Anwendbarkeit des selbstlernenden OoN-Erkennungsverfahrens werden eigens definierte Kriterien herangezogen. Die drei Wichtigsten sind: (i) die Fehlalarmrate, (ii) die Wahrscheinlichkeit der Erkennung tatsächlicher Abweichungen, (iii) der Anteil des prüfbaren Verhaltens im Trace (Systemabdeckung).

Eines der sensibelsten Kriterien ist dabei die Fehlalarmrate, da zu viele falsche OoN-Indikationen einen praktischen Einsatz des Verfahrens unmöglich machen würden. Wie bei vielen Methoden des maschinellen Lernens treten auch bei dem hier entwickelten Verfahren Fehlerkennungen durch überspezialisierte Modelle auf. Diesem Effekt wird durch die Anwendung eines speziellen Plausibilisierungsverfahrens entgegengewirkt. Die Plausibilisierung beruht auf dem Vergleich der Ergebnisse mehrerer durch die OoN-Erkennung geprüfter Traces und reduziert die Anzahl der ausgelösten Fehlalarme signifikant. Bei der Überprüfung von vier CAN-Traces konnte so die Fehlalarmrate der OoN-Erkennung von über 90 % auf ca. 35 % reduziert werden. Die Größenordnung der Fehlalarmrate von 30 bis 40 % ist für automatisierte Analyseverfahren durchaus üblich und kann daher als ein gutes Ergebnis gewertet werden.

Mit diesem abschließenden Verfahrensschritt weist das entwickelte OoN-Erkennungsverfahren für die verwendeten CAN-Traces eine Abdeckung von ca. 45 % auf. Das heißt, es werden 45 % aller im CAN-Trace beobachteten Events einer entsprechenden Bewertung unterzogen. Dies ist ein beachtenswertes Ergebnis, wenn davon auszugehen ist, dass bisher nur ein Bruchteil des Kommunikationsverhaltens automatisch überwacht werden konnte.

Durch die umfangreiche Evaluierung des selbstlernenden OoN-Erkennungsverfahren mittels realer CAN-Netzwerktraces eines Fahrzeugs kann gezeigt werden, dass die prinzipielle Machbarkeit des hier entwickelten neuen Ansatzes einer selbstlernenden OoN-Erkennung gegeben ist. Neben den ermittelten Erkennungseigenschaften kann die prinzipielle Anwendbarkeit auch für die benötigte Rechenleistung des Verfahrens in der Lernphase und in der Anwendungsphase konstatiert werden. Für die Lernphase ist die gute Parallelisierbarkeit hervorzuheben und für die Anwendungsphase der sehr kleine Speicherbedarf für die erstellten Modelle sowie der geringe Rechenaufwand zur Prüfung eines Traces.

7.2 Ausblick

Die Arbeit untersucht die Machbarkeit eines Verfahrens, welches ohne die Verwendung einer Spezifikation das Kommunikationsverhalten bewerten kann. Es ist aber davon auszugehen, dass zumindest für die wichtigsten Abläufe eines Systems Spezifikationen bzw. Beschreibungen des gewünschten Verhaltens vorhanden sind. Sind solche Informationen vorhanden, wird es hilfreich sein, diese in den Lernprozess einzubeziehen.

Um dies zu erreichen, wäre eine Anpassung des Lernverfahrens notwendig. Ein gut geeignetes Lernverfahren für diese Problemstellung ist der der bereits verwendete Angluin L^*-Algorithmus. Dieser kann explizit erlaubtes und auch nicht erlaubtes Verhalten durch entsprechende Gegenbeispiele in seine erstellten Automaten integrieren. Weiterhin kann bereits vorhandenes Systemwissen sicherlich sehr gut dazu genutzt werden, parallele Handlungsstränge bzw. voneinander abhängige Events entsprechend zu gruppieren. Ist im Vorfeld bekannt, welche Events sich gegenseitig beeinflussen bzw. zu einer Funktion gehören, so können diese bekannten Eventgruppen eine sehr gute Ergänzung zu den automatisch gefundenen Gruppen darstellen.

Neben der Integration von Systemwissen in den Lernprozess besteht umgekehrt die Möglichkeit, Erkenntnisse über Zusammenhänge im System aus dem vorgestellten Verfahren zu gewinnen. Ein ähnliches Ziel wird unter anderem in (Werner 2010) verfolgt. Dort wird mittels des Angluin L^*-Algorithmus eine automatische Anpassung der Spezifikation beschrieben. Neben der Verbesserung der Spezifikation ist auch ein Erkenntnisgewinn durch die neu eingeführten Clustering-Verfahren vorstellbar. So ist es möglich, dass die identifizierten Eventgruppen neue Aufschlüsse über gewollte oder auch ungewollte Querbeziehungen aufdecken und so helfen das interne Systemverhalten besser zu verstehen. Neben der Bewertung von Netzwerkkommunikation erscheint auch eine Anwendung zur Analyse der Interaktion parallel ausgeführter Programme (z. B. auf Multi-Core-Prozessoren) für das entwickelte Verfahren denkbar. Dabei könnten z. B. Trigger zur Aufzeichnung von Debuggerdaten bei Langzeitbeobachtung mittels des hier entwickelten Verfahrens ausgelöst werden.

Eine prinzipbedingte Schwachstelle in der Wirkungsweise des entwickelten Verfahrens ist die Abhängigkeit von der Qualität der Daten im Referenztrace. Ist im Referenztrace nur wenig aussagekräftiges Verhalten enthalten, so werden auch die erlernten Überwachungsmodelle nur wenig Aussagekraft besitzen. Um dies etwas abzumildern, müsste das Verhalten mehrere Referenztraces in gewisser Weise akkumuliert werden. So könnten über mehrere Beobachtungen bzw. Entwicklungszyklen ein besseres Bild des Normzustandes des Systems aufgestellt werden und die Güte der OoN-Erkennung verbessert werden. Derzeit ist das Verfahren ausschließlich in der Lage, einen bestehenden Datensatz an Referenztraces zu verarbeiten und daraus Modelle zu generieren. Eine rekursive Vorgehensweise würde es ermöglichen, nachträglich beobachtetes Systemverhalten in die erlernten Modelle integrieren zu können.

Erlauben es die vorgenannten Verbesserungen des Verfahrens, das Normverhalten eines Systems über einen längeren Zeitraum kontinuierlich zu erlernen, so wird die Anzahl unbekannten Systemverhaltens sehr klein werden. Somit wird sich auch die Anzahl der erkannten Abweichungen in absoluten Zahlen verringern. Ist die Anzahl der Abweichungen hinreichend klein, bestünde die Möglichkeit, das vorgestellte Verfahren zur Laufzeit des Systems als erweiterte Fehlererkennung anzuwenden. Denkbar ist hier etwa ein automatischer Trigger, der die Aufzeichnung des Netzwerkverkehrs für eine spätere Analyse auslöst. Dies kann vor allem bei Langzeiterprobungen und Netzwerken mit hohen Datenraten sehr hilfreich sein, da eine komplette Aufzeichnung des Netzwerkverkehrs über mehrere Stunden auch heute schon sehr große Datenmengen produziert. Für eine solche Anwendung spricht vor allem der geringe Ressourcenverbrauch des Verfahrens in der Anwendungsphase, so dass das System auf einem mobilen Messcomputer ausgeführt werden könnte.

Literatur- und Referenzverzeichnis

Adzic 2011 ADZIC, Gojko: *Specification by example : How successful teams deliver the right software.* Shelter Island, N.Y : Manning, 2011. – ISBN 1617290084

Almeida, Moreira, Reis 2009 ALMEIDA, Marco ; MOREIRA, Nelma ; REIS, Rogério: Testing the Equivalence of Regular Languages. In: JÜRGEN DASSOW; GIOVANNI PIGHIZZINI; BIANCA TRUTHE (HRSG.): *Proceedings Eleventh International Workshop on Descriptional Complexity of Formal Systems,* 2009 (EPTCS, 3), S. 47–57

Alur, Dill 1994 ALUR, Rajeev ; DILL, David L.: *A theory of timed automata.* In: *Theoretical Computer Science* 126 (1994), Nr. 2, S. 183–235

Angluin 1987 ANGLUIN, Dana: *Learning regular sets from queries and counterexamples.* In: *Information and computation* 75 (1987), Nr. 2, S. 87–106

Ankerst u. a. 1999 ANKERST, Mihael ; BREUNIG, Markus M. ; KRIEGEL, Hans-peter ; SANDER, Jörg: *OPTICS: ordering points to identify the clustering structure.* In: *ACM SIGMOD Record* 28 (1999), Nr. 2, S. 49–60

Atkinson, Long, Hanzevack 1998 ATKINSON, C. M. ; LONG, T. W. ; HANZEVACK, E. L.: Virtual sensing: a neural network-based intelligent performance and emissions prediction system for on-board diagnostics and engine control. In: *The 1998 SAE International Congress & Exposition,* 1998, S. 39-51

Avizienis u. a. 2000 AVIZIENIS, Algirdas ; LAPRIE, Jean-Claude ; RANDELL, Brian ; VYTAUTAS: *Fundamental Concepts of Dependability, Toulouse France: LAAS-CNRS Research Report Nr. 1145,* 2000

Ball 1999 BALL, Thomas: The concept of dynamic analysis. In: *Software Engineering—ESEC/FSE'99,* 1999, S. 216–234

Barbará, Li, Couto 2002 BARBARÁ, Daniel ; LI, Yi ; COUTO, Julia: COOLCAT: An entropy-based algorithm for categorical clustering. In: *Proceedings of the eleventh international conference on Information and knowledge management,* 2002, S. 582–589

Bauer, Leucker, Schallhart 2010 BAUER, Andreas ; LEUCKER, Martin ; SCHALLHART, Christian: *Comparing LTL Semantics for Runtime Verification.* In: *Journal of Logic and Computation* 20 (2010), Nr. 3, S. 651–674. URL http://dx.doi.org/10.1093/logcom/exn075

Becker, Zeller, Weiss 2012 BECKER, Klaus ; ZELLER, Marc ; WEISS, Gereon: Towards Efficient On-line Schedulability Tests for Adaptive Networked Embedded Real-time Systems. In: CÉSAR BENAVENTE-PECES; FALAH H. ALI; JOAQUIM FILIPE (HRSG.): *PECCS 2012 - Proceedings of the 2nd International Conference on Pervasive Embedded Computing and Communication Systems, Rome, Italy, 24-26 February, 2012 :* SciTePress, 2012. – ISBN 978-989-8565-00-6, S. 440–449

Berg u. a. 2003 BERG, Therese ; JONSSON, Bengt ; LEUCKER, Martin ; SAKSENA, Mayank: *Insights to Angluin's Learning.* In: Electronic Notes in Theoretical Computer Science 118 (2003), S. 3–18

Berg u. a. 2005 BERG, Therese ; GRINCHTEIN, Olga ; JONSSON, Bengt ; LEUCKER, Martin ; RAFFELT, Harald ; STEFFEN, Bernhard: On the correspondence between conformance

testing and regular inference. In: *Lecture Notes in Computer Science :* Springer, 2005, S. 175-189

Berg 2006 BERG, Therese: *Regular Inference for Reactive Systems.* UPPSALA Schwweden, UPPSALA UNIVERSITY, Department of Information Technology. Dissertation. 2006

Biermann, Feldman 1972 BIERMANN, A. W. ; FELDMAN, J. A.: *On the Synthesis of Finite-State Machines from Samples of Their Behavior.* In: *IEEE Transactions on Computers* 21 (1972), Nr. 6, S. 592–597

Blumer u. a. 1987 BLUMER, Anselm ; EHRENFEUCHT, Andrzej ; HAUSSLER, David ; WARMUTH, Manfred K.: *Occam's razor.* In: *Information processing letters* 24 (1987), Nr. 6, S. 377–380

Bochmann, Petrenko 1994 BOCHMANN, Gregor V. ; PETRENKO, Alexandre: Protocol testing: review of methods and relevance for software testing. In: *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis.* Seattle, Washington, USA : ACM, 1994. – ISBN 0-89791-683-2, S. 109–124

Bollig u. a. 2007 BOLLIG, Benedikt ; KATOEN, Joost-Pieter ; KERN, Carsten ; LEUCKER, Martin: Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning, Bd. 4424. In: ORNA GRUMBERG; HUTH, Michael (Hrsg.): *Proceedings of the 13th International Conference on Tools and Algorithms for Construction and Analysis of Systems.* Braga, Portugal : Springer, 2007 (Lecture Notes in Computer Science), S. 435–450

Bollig u. a. 2010 BOLLIG, Benedikt ; KATOEN, Joost-Pieter ; KERN, Carsten ; LEUCKER, Martin: *Learning Communicating Automata from MSCs.* In: *IEEE Transactions on Software Engineering (TSE)* 36 (2010), Nr. 3, S. 390–408. URL http://doi.ieeecomputersociety.org/10.1109/TSE.2009.89

Brezak u. a. 2012 BREZAK, Danko ; BACEK, Tomislav ; MAJETIC, Dubravko ; KASAC, Josip ; NOVAKOVIC, Branko: A comparison of feed-forward and recurrent neural networks in time series forecasting. In: *Computational Intelligence for Financial Engineering & Economics (CIFEr), 2012 IEEE Conference on,* 2012, S. 1-6

Broy u. a. 2005 BROY, Manfred ; JONSSON, Bengt ; KATOEN, Joost-Pieter ; LEUCKER, Martin ; PRETSCHNER, Alexander: *Model-based testing of reactive systems: advanced lectures :* Springer, 2005 (3472)

Broy, Olderog 2001 BROY, M. ; OLDEROG, E. -R: Trace-Oriented Models of Concurrency. In: BERGSTRA, J. A.; PONSE, A.; SCOTT, S. A. (Hrsg.): *Handbook of Process Algebra :* Elsevier Science B.V, 2001, S. 101–195

Buller, Hanselka 2012 BULLER, Ulrich ; HANSELKA, Holger: *Fraunhofer-Systemforschung Elektromobilität : Aspekte der Fraunhofer-Systemforschung.* Stuttgart : Fraunhofer Verlag, 2012. – ISBN 978-3-8396-0410-6

Chandola, Banerjee, Kumar 2009 CHANDOLA, Varun ; BANERJEE, Arindam ; KUMAR, Vipin: *Anomaly detection: A survey.* In: *ACM Comput. Surv.* 41 (2009), Nr. 3, S. 1–58. URL http://dx.doi.org/10.1145/1541880.1541882

Chen, Lee 2002 CHEN, Y. M. ; LEE, M. L.: *Neural networks-based scheme for system failure detection and diagnosis.* In: *Mathematics and Computers in Simulation* 58

(2002), Nr. 2, S. 101-109. URL http://www.sciencedirect.com/science/article/B6V0T-44NM80H-1/2/1b64c8d3c99045fa6775c05af581990c

Cleeremans, Servan-Schreiber, McClelland 1989 CLEEREMANS, Axel ; SERVAN-SCHREIBER, David ; MCCLELLAND, James L.: *Finite state automata and simple recurrent networks.* In: *Neural computation* 1 (1989), Nr. 3, S. 372-381

Coombes u. a. 2005 COOMBES, Kevin R. ; TSAVACHIDIS, Spiridon ; MORRIS, Jeffrey S. ; BAGGERLY, Keith A. ; HUNG, Mien-Chie ; KUERER, Henry M.: *Improved peak detection and quantification of mass spectrometry data acquired from surface-enhanced laser desorption and ionization by denoising spectra with the undecimated discrete wavelet transform.* In: *Proteomics* 5 (2005), Nr. 16, S. 4107–4117

Drabek u. a. 2013 DRABEK, Christian ; PRAMSOHLER, Thomas ; ZELLER, Marc ; WEISS, Gereon: Interface Verification Using Executable Reference Models: An Application in the Automotive Infotainment⋆. In: *Proceedings of the 6th International Workshop on Model Based Architecting and Construction of Embedded Systems.* Miami, Florida, USA, 2013, S. 7:1–10

Du, Kibbe, Lin 2006 DU, Pan ; KIBBE, Warren A. ; LIN, Simon M.: *Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching.* In: *Bioinformatics* 22 (2006), Nr. 17, S. 2059–2065

Ebert, Jones 2009 EBERT, Christof ; JONES, Capers: *Embedded Software: Facts, Figures, and Future.* In: *IEEE Computer* 42 (2009), Nr. 4, S. 42–52. URL http://dblp.uni-trier.de/db/journals/computer/computer42.html#EbertJ09

Eilers, Weiss 2010 EILERS, Dirk ; WEISS, Gereon: *Device for creating a marked reference data stream.* Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. Anmeldenr. EP20090015132, Germany. 9 Jun 2010. Veröffentlichungsnr. EP2194457 A2

Ester u. a. 1996 ESTER, Martin ; KRIEGEL, Hans-peter ; S, Jörg ; XU, Xiaowei: A density-based algorithm for discovering clusters in large spatial databases with noise. In: SIMOUDIS, Evangelos; HAN, Jiawei; FAYYAD, Usama M. (Hrsg.): *Second international conference on knowledge discovery and data mining :* AAAI Press, 1996. – ISBN 1-57735-004-9, S. 226–231

Friedrich 2008 FRIEDRICH, Jan: *Das V-Modell XT : Für Projektleiter und QS-Verantwortliche ; kompakt und übersichtlich.* Berlin, Heidelberg : Springer, 2008 (Informatik im Fokus). – ISBN 9783540764045

Geisberger, Broy 2012 GEISBERGER, Eva ; BROY, Manfred: *agendaCPS - Integrierte Forschungsagenda Cyber-Physical Systems (acatech STUDIE).* 2012

Gers 2001 GERS, Felix: *Long Short-Term Memory in Recurrent Neural Networks.* In: *Lausanne, EPFL* (2001)

Gers, Schmidhuber, Cummins 1999 GERS, Felix A. ; SCHMIDHUBER, Jürgen ; CUMMINS, Fred: Learning to forget: Continual prediction with LSTM, Bd. 2. In: *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470),* 1999, S. 850–855

Gers, Schmidhuber, Cummins 2000 GERS, Felix A. ; SCHMIDHUBER, Jürgen ; CUMMINS, Fred: *Learning to forget: Continual prediction with LSTM.* In: *Neural computation* 12 (2000), Nr. 10, S. 2451-2471

Ghahramani 2004 GHAHRAMANI, Zoubin: Unsupervised learning. In: *Advanced Lectures on Machine Learning :* Springer, 2004. – ISBN 3540231226, S. 72–112

Grinchtein, Jonsson, Leucker 2004 GRINCHTEIN, Olga ; JONSSON, Bengt ; LEUCKER, Martin: Learning of event-recording automata. In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems :* Springer, 2004, S. 379–395

Guyon 2006 GUYON, Isabelle: *Feature extraction : Foundations and applications.* Berlin : Springer-Verlag, 2006 (Studies in fuzziness and soft computing 207). – ISBN 978-3-540-35487-1

Hangal, Lam 2002 HANGAL, Sudheendra ; LAM, Monica S.: Tracking down software bugs using automatic anomaly detection. In: *Proceedings of the 24th International Conference on Software Engineering.* New York, NY, USA : ACM, 2002 (ICSE '02). – ISBN 1-58113-472-X, S. 291–301

Hochreiter, Schmidhuber 1995 HOCHREITER, Sepp ; SCHMIDHUBER, Jürgen: *Long Short-Term Memory.* 1995(FKI-207-95)

Hochreiter, Schmidhuber 1997 HOCHREITER, Sepp ; SCHMIDHUBER, Jürgen: *Long short-term memory.* In: *Neural computation* 9 (1997), Nr. 8, S. 1735-1780

Hopcroft, Ullman 1992 HOPCROFT, John E. ; ULLMAN, Jeffrey D.: *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie.* 1. Aufl. Bonn [u.a.] : Addison-Wesley, 1992 (Internationale Computer-Bibliothek). – ISBN 3-89319-181-X

Hu u. a. 1999 HU, Michael Y. ; ZHANG, Guoqiang Peter ; JIANG, Christine X. ; PATUWO, B. Eddy: *A Cross-Validation Analysis of Neural Network Out-of-Sample Performance in Exchange Rate Forecasting.* In: *Decision Sciences* 30 (1999), Nr. 1, S. 197-216

ISO 26262 2009, ISO/DIS 26262-1: - *Road vehicles — Functional safety — Part 1 Glossary*

Jha, Tan, Maxion 2001 JHA, S. ; TAN, K. ; MAXION, R. A.: Markov Chains, Classifiers, and Intrusion Detection. In: *Proceedings of the 14th IEEE Workshop on Computer Security Foundations.* Washington, DC, USA : IEEE Computer Society, 2001 (CSFW '01), S. 206–219

Jong 2006 JONG, Kenneth A. de: *Evolutionary computation : A unified approach.* Cambridge, Mass : MIT Press, 2006. – ISBN 978-0-262-04194-2

Kaelbling, Littman, Moore 1996 KAELBLING, Leslie Pack ; LITTMAN, Michael L. ; MOORE, Andrew W.: *Reinforcement Learning: A Survey.* In: *CoRR* 9605103 (1996)

Kohzadi u. a. 1996 KOHZADI, Nowrouz ; BOYD, Milton S. ; KERMANSHAHI, Bahman ; KAASTRA, Iebeling: *A comparison of artificial neural network and time series models for forecasting commodity prices.* In: *Neurocomputing* 10 (1996), Nr. 2, S. 169-181

Kolmogorov 1965 KOLMOGOROV, A. N.: *Three approaches to the quantitative definition ofinformation'.* In: *Problems of information transmission* 1 (1965), Nr. 1, S. 1–7

Kranzmüller 2000 KRANZMÜLLER, Dieter: *Event graph analysis for debugging massively parallel programs.* Johannes Kepler Universitat Linz, Austria, Institute of Graphics and Parallel Processing. Dissertation. 2000. URL http://www.mcs.anl.gov/~norris/MPIGraphAnalysis-KranzlMueller.pdf

Kremenek u. a. 2004 KREMENEK, Ted ; ASHCRAFT, Ken ; YANG, Junfeng ; ENGLER, Dawson: *Correlation exploitation in error ranking.* In: *SIGSOFT Softw. Eng. Notes* 29 (2004), Nr. 6, S. 83–93. URL http://doi.acm.org/10.1145/1041685.1029909

Kwok, Yeung 1997 KWOK, Tin-Yau ; YEUNG, Dit-Yan: *Constructive algorithms for structure learning in feedforward neural networks for regression problems.* In: *IEEE Transactions on Neural Networks* 8 (1997), Nr. 3, S. 630–645

Lamport 1987 Lamport, Leslie: *Subject: distribution.* E-Mail. 28.05.1987. DEC SRC bulletin board (Adressat)

Langer, Bertulies, Hoffmann 2011 LANGER, Falk ; BERTULIES, Karsten ; HOFFMANN, F.: Self Learning Anomaly Detection for Embedded Safety Critical Systems. In: *Schriftenreihe des Instituts für Angewandte Informatik, Automatisierungstechnik am Karlsruher Institut für Technologie :* KIT Scientific Publishing, 2011, S. 31–45

Langer, Eilers, Knorr 2009 LANGER, Falk ; EILERS, Dirk ; KNORR, Rudi: Fault Detection in Discrete Event Based Distributed Systems by Forecasting Message Sequences with Neural Networks, Bd. 5803. In: MERTSCHING, Bärbel; HUND, Marcus; AZIZ, Zaheer (Hrsg.): *KI 2009: Advances in Artificial Intelligence :* Springer Berlin / Heidelberg, 2009 (Lecture Notes in Computer Science), S. 411–418

Langer, Oswald 2014 a LANGER, Falk ; OSWALD, Erik: Using Reference Traces for Validation of Communication in Embedded Systems. In: ICONS 2014, The Ninth International Conference on Systems, 2014, S. 203–208

Langer, Oswald 2014 b Langer, Falk ; Oswald, Erik: A self-learning approach for validation of communication in embedded systems. In: *3rd Workshop on Realizing Artificial Intelligence Synergies in Software Engineering,* 2014 (Proceedings of 36th International Conference on Software Engineering).

Langer, Prehofer 2011 LANGER, Falk ; PREHOFER, Christian: Anomaly detection in embedded safety critical software. In: *International Workshop on Principles of Diagnosis (DX),* 2011, S. 163–166

Lazarevic, Kumar, Srivastava 2005 LAZAREVIC, Aleksandar ; KUMAR, Vipin ; SRIVASTAVA, Jaideep: Intrusion Detection: A Survey, Bd. 5. In: KUMAR, Vipin; SRIVASTAVA, Jaideep; LAZAREVIC, Aleksandar (Hrsg.): *Managing Cyber Threats :* Springer US, 2005 (Massive Computing). – ISBN 978-0-387-24226-2, S. 19–78

Lemos u. a. 2007 LEMOS, R. de ; TIMMIS, J. ; FORREST, S. ; AYARA, M.: *Immune-Inspired Adaptable Error Detection for Automated Teller Machines.* In: *IEEE SMC Part C: Applications and Reviews* 37 (2007), Nr. 5, S. 873–886

Leucker, Schallhart 2009 LEUCKER, Martin ; SCHALLHART, Christian: *A Brief Account of Runtime Verification.* In: *Journal of Logic and Algebraic Programming* 78 (2009), Nr. 5, S. 293–303. URL http://dx.doi.org/10.1016/j.jlap.2008.08.004

Lim, Volker, Herrscher 2011 LIM, Hyung-Taek ; VOLKER, Lars ; HERRSCHER, Daniel: Challenges in a future IP/ethernet-based in-car network for real-time applications. In: *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE,* 2011, S. 7–12

Lin, Wonham 1988 LIN, Feng ; WONHAM, W. Murray: *On Observability of of Discrete-Event Systems.* In: *Information Sciences* 44 (1988), Nr. 2, S. 173–198

Liu, Motoda 1998 LIU, Huan ; MOTODA, Hiroshi: *Feature extraction, construction and selection : A data mining perspective.* Boston : Kluwer Academic, 1998 (The Kluwer international series in engineering and computer science 453). – ISBN 0-7923-8196-3

Lutz 1993 LUTZ, Robyn R.: Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems. In: *Proceedings of the IEEE International Symposium on Requirements Engineering,* 1993, S. 126-133

Lutz 2003 LUTZ, Robyn R.: Requirements Discovery during the Testing of Safety-Critical Software. In: *Proc. 25th Int'l Conf. Software Eng. (ICSE 03), IEEE CS :* Press, 2003, S. 578-583

Manderscheid, Langer 2011 MANDERSCHEID, Martin ; LANGER, Falk: Network calculus for the validation of automotive ethernet in-vehicle network configurations. In: *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on,* 2011, S. 206–211

Maxion, Tan 2000 MAXION, Roy A. ; TAN, Kymie M. C.: Benchmarking anomaly-based detection systems. In: IEEE (HRSG.): *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on Dependable Systems & Networks,* 2000, S. 623–630

Maxion, Tan 2002 MAXION, Roy A. ; TAN, Kymie M. C.: *Anomaly detection in embedded systems.* In: *Computers, IEEE Transactions on* 51 (2002), Nr. 2, S. 108–120

Mealy 1955 MEALY, George H.: *A method for synthesizing sequential circuits.* In: *Bell System Technical Journal* 34 (1955), Nr. 5, S. 1045–1079

Mühl, Fiege, Pietzuch 2006 MÜHL, Gero ; FIEGE, Ludger ; PIETZUCH, Peter: *Distributed Event-Based Systems.* Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2006. – ISBN 9783540326519

Narendra, Thathachar 1974 NARENDRA, Kumpati S. ; THATHACHAR, M. A. L.: *Learning Automata - A Survey.* In: *IEEE Transactions on Systems, Man, and Cybernetics* 4 (1974), Nr. 4, S. 323–334

Nickolayev, Roth, Reed 1997 NICKOLAYEV, Oleg Y. ; ROTH, Philip C. ; REED, Daniel A.: *Real-time statistical clustering for event trace reduction.* In: *International Journal of High Performance Computing Applications* 11 (1997), Nr. 2, S. 144–159

Obermaisser, Peti 2007 OBERMAISSER, Roman ; PETI, Philipp: Detection of Out-of-Norm Behaviors in Event-Triggered Virtual Networks, Bd. 2. In: *Industrial Informatics, 2007 5th IEEE International Conference on,* 2007, S. 971–976

Ozveren, Willsky 1990 OZVEREN, Cuneyt M. ; WILLSKY, Alan S.: *Observability of discrete event dynamic systems.* In: *Automatic Control, IEEE Transactions on* 35 (1990), Nr. 7, S. 797–806

Peti, Obermaisser, Kopetz 2005 PETI, Philipp ; OBERMAISSER, Roman ; KOPETZ, Hermann: Out-of-norm assertions [diagnostic mechanism]. In: *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE :* IEEE, 2005. – ISBN 0769523021, S. 280–291

Raman, Patrick, North RAMAN, Anand V. ; PATRICK, Jon D. ; NORTH, Palmerston: The sk-strings method for inferring PFSA. In: *Citeseer 1997 – Proceedings of the workshop automata induction, grammatical inference and language acquisition at the 14th international conference on machine learning (ICML97).*

Ramberger, Gruber, Herzner 2004 RAMBERGER, Stephan ; GRUBER, Thomas ; HERZNER, Wolfgang: Experience Report: Error Distribution in Safety-Critical Software and Software Risk Analysis Based on Unit Tests. In: *GI Jahrestagung (1),* 2004, S. 72–76

Rathmanner, Hutter 2011 RATHMANNER, Samuel ; HUTTER, Marcus: *A Philosophical Treatise of Universal Induction.* In: *Entropy* 13 (2011), Nr. 6, S. 1076–1136. URL http://dx.doi.org/10.3390/e13061076

Reber 1967 REBER, Arthur S.: *Implicit learning of artificial grammars.* In: *Journal of verbal learning and verbal behavior* 6 (1967), Nr. 6, S. 855–863

Ribeiro, Alquézar 2002 RIBEIRO, Sabrine ; ALQUÉZAR, René: Incremental construction of LSTM recurrent neural network, Bd. 2. In: *Proceedings of the VII Congreso Iberoamericano de Reconocimiento de Patrones, CIARP,* 2002, S. 171–184

Ring 1993 RING, Mark: *Sequence Learning with Incremental Higher-Order Neural Networks.* 1993(AI93-193)

Rissanen 1983 RISSANEN, Jorma: *A universal prior for integers and estimation by minimum description length.* In: *The Annals of statistics* (1983), S. 416–431

Salem Alelyani, Jiliang Tang, Huan Liu 2013 SALEM ALELYANI ; JILIANG TANG ; HUAN LIU: Feature Selection for Clustering: A Review. In: *Data Clustering: Algorithms and Applications,* 2013, S. 29–60

Schlummer 2012 SCHLUMMER, M.H: *Beitrag zur Entwicklung einer alternativen Vorgehensweise für eine Proven-in-use-Argumentation in der Automobilindustrie.* Wuppertal, Bergische Universität Wuppertal. Dissertation

Schreiber, Cleeremans, McClelland 1988 SCHREIBER, David S. ; CLEEREMANS, Axel ; MCCLELLAND, James L.: *Encoding Sequential Structure in Simple Recurrent Networks.* Pittsburgh, PA, 1988(CMU-CS-88-183)

Settles 2009 SETTLES, B.: *Active Learning Literature Survey.* 2009 Computer Sciences Technical Report 1648, University of Wisconsin–Madison

Settles 2011 SETTLES, B.: From Theories to Queries: Active Learning in Practice, Bd. 15. In: *Active Learning and Experimental Design :* Microtome Publishing, 2011 (JMLR Workshop and Conference Proceedings), S. 1–18

Shannon 1948 SHANNON, Claude E.: *A mathematical theory of communication.* In: *Bell System Technical Journal* 27 (1948), S. 379–423. URL http://dx.doi.org/10.1145/584091.584093

Solomonoff 1964 SOLOMONOFF, R.: *A formal theory of inductive inference, part I.* In: *Information and Control* 7 (1964), Nr. 1, S. 1–22

Tanenbaum, van Steen 2006 TANENBAUM, Andrew S. ; VAN STEEN, Maarten: *Distributed Systems: Principles and Paradigms (2nd Edition).* 2. Aufl. : Prentice Hall, 2006. – ISBN 0132392275

Tao, Sheng, Mitsunori 2004 TAO, LI ; SHENG, MA ; MITSUNORI, OGIHARA: *Entropy-based criterion in categorical clustering*, BD. 69. IN: CARLA E. BRODLEY (HRSG.): MACHINE LEARNING, PROCEEDINGS OF THE TWENTY-FIRST INTERNATIONAL CONFERENCE (ICML 2004), BANFF, ALBERTA, CANADA, JULY 4-8, 2004 : ACM, 2004 (ACM INTERNATIONAL CONFERENCE PROCEEDING SERIES).

Thompson 2012 THOMPSON, Haydn: *Mixed Criticality Systems : Report from the Workshop on Mixed Criticality Systems.* URL http://cordis.europa.eu/fp7/ict/embedded-systems-engineering/documents/report_mixed_criticality_systems.pdf. – Aktualisierungsdatum: 2012-04-05

Uckelmann, Harrison, Michahelles 2011 UCKELMANN, Dieter ; HARRISON, Mark ; MICHAHELLES, Florian: *Architecting the internet of things.* Berlin, Heidelberg, New York : Springer, 2011. – ISBN 3642191576

Vemuri, Polycarpou 1997 VEMURI, A.T. ; POLYCARPOU, M.M.: *Neural-network-based robust fault diagnosis in robotic systems.* In: *Neural Networks, IEEE Transactions on* 8 (1997), Nr. 6, S. 1410-1420

Waldmann, Stocker 2013 WALDMANN, Karl-Heinz ; STOCKER, Ulrike M.: *Stochastische Modelle : Eine anwendungsorientierte Einführung.* 2. Aufl. Berlin : Springer, 2013 (EMILA-stat). – ISBN 978-3-642-32911-1

Wallace, Dowe 1999 WALLACE, C. S. ; DOWE, D. L.: *Minimum Message Length and Kolmogorov Complexity.* In: *The Computer Journal* 42 (1999), Nr. 4, S. 270–283. URL http://dx.doi.org/10.1093/comjnl/42.4.270

Weicker 2007 WEICKER, Karsten: *Evolutionäre Algorithmen.* 2. Aufl. Wiesbaden : Springer; B.G. Teubner Verlag / GWV Fachverlage GmbH Wiesbaden, 2007. – ISBN 3835102192

Weise 2009 WEISE, Thomas: *Global Optimization Algorithms - Theory and Application :* it-weise.de (self-published): Germany, 2009

Weiss u. a. 2010 WEISS, Gereon ; ZELLER, Marc ; EILERS, Dirk ; KNORR, Rudi: Approach for iterative validation of automotive embedded systems. In: *Proceedings of the 3rd International Workshop on Model Based Architecting and Construction of Embedded Systems,* 2010, S. 69–83

Werner 2010 WERNER, Edith Benedicta Maria: *Learning Finite State Machine Specifications from Test Cases.* Niedersächsische Staats-und Universitätsbibliothek Göttingen. Dissertation. 2010

Wolf 2013 WOLF, Julian: *Feingranulare Korrektheitsprüfung des Kontrollflusses von Echtzeitsystemen.* Universität Augsburg, Fakultät für Angewandte Informatik. Dissertation. 2013

Zandrahimi, Zarei, Zarandi 2010 ZANDRAHIMI, Mahroo ; ZAREI, Alireza ; ZARANDI, Hamid R.: A Probabilistic Method to Detect Anomalies in Embedded Systems. In: *Defect and Fault Tolerance in VLSI Systems (DFT), 2010 IEEE 25th International Symposium on,* 2010, S. 152-159

Zimmermann, Schmidgall 2011 ZIMMERMANN, Werner ; SCHMIDGALL, Ralf: *Bussysteme in der Fahrzeugtechnik.* 4. Aufl. : Vieweg + Teubner, 2011. – ISBN 978-3-8348-0907-0

Anhang A

Beispiel Lernprozess des Angluin L^*-Algorithmus

Im Folgenden wird eine beispielhafte Ausführung des genutzten Angluin L^*-Algorithmus nach (Angluin 1987) gezeigt und erläutert.

In diesem Beispiel ist ein Automat zu erlernen, wie er in Abbildung A-1 gezeigt wird. Mit Hilfe dieses bekannten Automaten werden jeweils die Zugehörigkeitsfrage und die Äquivalenzfrage beantwortet. Die Realisierung der Zugehörigkeitsfrage und Äquivalenzfrage auf Basis von Referenztraces (Kapitel 3.3.3.1) wird hier nicht näher erläutert, da diese unabhängig von der grundlegenden Funktion des Angluin L^*-Algorithmus sind.

Abbildung A-1 Tatsächlich ausgeführter Automat

Die Beobachtungstabelle setzt sich zusammen aus den Zustandsreihen mit den Strings S als Zustandssuffix, den Transitionen mit den Strings T und den Zustandspräfixen in den Spalten mit den Strings E. Alle Strings bestehen aus Zeichen des Alphabets A. Dabei sind alle T zusammengesetzt aus $S \times A$ (T enthält alle Kombinationen $s_i + a_j$). Ein neues Zustandspräfix wird jeweils durch das Voranstellen eines neuen Zeichens aus A zu einem bestehenden Präfix gebildet ($e_{i+1} = a_n + e_i$).

In der ersten Phase erzeugt der Lernalgorithmus eine initiale Hypothese mit der resultierenden Beobachtungstabelle, wie sie in Tabelle A-1 dargestellt ist. Es wird zunächst ein leerer Zustand und für alle Zeichen des Alphabetes eine Transition eingeführt. Die resultierenden Automaten sind in Abbildung A-2 dargestellt.

Tabelle A-1 Beobachtungstabelle nach Initialisierung

		Suffixe
	Präfix	*lehr*
Zustände	*leer*	*wahr*
Transitionen	A	*wahr*
	B	*wahr*

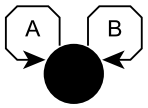

Abbildung A-2 Erlernter Automat resultierend aus initialer Beobachtungstabelle

Es wird nun die Äquivalenzfrage gestellt. In diesem Beispiel liefert diese den Wert „AA“ als negatives Gegenbeispiel zurück („AA“ kann vom Automaten in Abbildung A-2 erzeugt werden, aber nicht vom tatsächlich ausgeführten Automaten, Abbildung A-1). Dies wird als neuer Zustand in die Beobachtungstabelle eingefügt (Tabelle A-2).

Tabelle A-2 Beobachtungstabelle der ersten Iteration nach Einfügung des Gegenbeispiels „AA“

		Suffixe
	Präfix	$leer$
Zustände	$leer$	$wahr$
	AA	$falsch$
Transitionen	A	$wahr$
	B	$wahr$

Die Beobachtungstabelle ist nicht geschlossen, da Zustand $leer$ mit Transition „A“ zu dem Zustand „AA“ führen wird, aber Zustand „AA“ nicht erlaubt ist.

Ist eine Beobachtungstabelle nicht geschlossen, so wird die Transition, welche zu dem ungültigen Zustand führt, in einen Zustand gewandelt. Weiterhin werden alle möglichen Transitionen von den bisherigen Zuständen als neue Transitionen eingeführt. Es wird daher Transition „A“ in die Zustandsreihe verschoben und alle möglichen Transitionen von den bisherigen Zuständen in die Tabelle eingefügt (alle $s_i + t_j \rightarrow$ „AA“, „AAA“, „AB“, „AAB“). Nachdem für alle Einträge der Beobachtungstabelle die Zugehörigkeitsfrage gestellt wurde, ergibt sich ein Beobachtungstabelle wie sie in Tabelle A-3 dargestellt ist.

Tabelle A-3 Geschlossene Beobachtungstabelle der ersten Iteration

		Suffixe
	Präfix	$leer$
Zustände	$leer$	$wahr$
	A	$wahr$
	AA	$falsch$
Transitionen	AAA	$falsch$
	AB	$wahr$
	AAB	$falsch$
	B	$wahr$

Die Beobachtungstabelle (Tabelle A-3) ist nicht konsistent, weil der Nachfolgezustand von „A" mit „A, $leer$" wahr ist, der Zustand „AA" falsch ist. Die neue Spalte setzt sich aus einem möglichen Zeichen des Alphabetes und dem Wert einer alten Spalte zusammen. In diesem Fall wird „A, $leer$" gewählt. Nach dem Stellen aller Zugehörigkeitsfragen ergibt sich eine Beobachtungstabelle, wie sie in Tabelle A-4 dargestellt ist.

Tabelle A-4 Konsistente und geschlossene Beobachtungstabelle nach der ersten Iteration

		Suffixe	
	Präfixe	$leer$	A, $leer$
Zustände	$leer$	$wahr$	$wahr$
	A	$wahr$	$falsch$
	AA	$falsch$	$falsch$
Transitionen	AA	$falsch$	$falsch$
	AAA	$falsch$	$falsch$
	AB	$wahr$	$wahr$
	AAB	$falsch$	$falsch$
	B	$wahr$	$wahr$

Die resultierende Beobachtungstabelle (Tabelle A-4) ist nun geschlossen und konsistent. Der resultierende Automat aus Beobachtungstabelle Tabelle A-4 ist in Abbildung A-3 dargestellt. Um aus der Beobachtungstabelle einen Automaten zu erstellen, wird zunächst die Anzahl der unterschiedlichen Zustände bestimmt (in diesem Fall zwei; $leer$ und A, $leer$ in Tabelle A-4 , wobei das führende $leer$ ignoriert wird). Danach werden die Transitionen, beginnend vom $leer$ Zustand, zugeordnet.

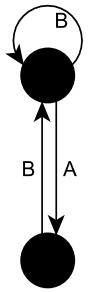

Abbildung A-3 Erlernter Automat, resultierend aus Beobachtungstabelle Tabelle A-4

Es wird nun wieder die Äquivalenzfrage gestellt. Diese liefert in diesem Beispiel den Wert „BB“ mit $falsch$ zurück („BB“ kann vom erlernten Automaten erzeugt werden, aber nicht vom tatsächlich ausgeführten Automaten ‚Abbildung A-1). Dies wird als Zustand in die Beobachtungstabelle eingefügt, womit sich Tabelle A-5 ergibt.

Tabelle A-5 Beobachtungstabelle der zweiten Iteration nach Einfügen des Gegenbeispiels „BB“

		Suffixe	
	Präfixe	$leer$	A, $leer$
Zustände	$leer$	$wahr$	$wahr$
	A	$wahr$	$falsch$
	AA	$falsch$	$falsch$
	BB	$falsch$	$falsch$
Transitionen	AAA	$falsch$	$falsch$
	AB	$wahr$	$wahr$
	AAB	$falsch$	$falsch$
	B	$wahr$	$wahr$

Die Beobachtungstabelle ist nicht geschlossen, da Zustand $leer$ mit Transition „B“ zu dem Zustand „BB“ führen wird, aber Zustand „BB“ nicht erlaubt ist. Es wird daher die Transition „B“ in die Zustände verschoben und die Transitionen werden entsprechend erweitert, so dass alle $S \times A$ in den Transitionen vorhanden sind (hinzufügen von „BA“, „BB“, „BBA“ und „BBB“). Somit entsteht die geschlossene Beobachtungstabelle Tabelle A-6.

Tabelle A-6 Geschlossene Beobachtungstabelle nach der zweiten Iteration

		Suffixe	
	Präfixe	$leer$	A, $leer$
Zustände	$leer$	$wahr$	$wahr$
	A	$wahr$	$falsch$
	AA	$falsch$	$falsch$
	B	$wahr$	$wahr$
	BB	$falsch$	$falsch$
Transitionen	AAA	$falsch$	$falsch$
	AB	$wahr$	$wahr$
	AAB	$falsch$	$falsch$
	BA	$wahr$	$wahr$
	BB	$falsch$	$falsch$
	BBA	$falsch$	$falsch$

Die Beobachtungstabelle (Tabelle A-3) ist nicht konsistent, weil der Nachfolgezustand von „B“ mit „B, $leer$“ $wahr$ ist, der Zustand „BB“ $falsch$ ist. Die neue Spalte setzt sich aus dem Wert einer alten Spalte und eines möglichen Zeichens zusammen. In diesem Fall wird „B, $leer$“ gewählt. Nach dem Stellen aller Zugehörigkeitsfragen ergibt sich eine Beobachtungstabelle, wie sie in Tabelle A-7 dargestellt ist.

Tabelle A-7 Konsistente und geschlossene Beobachtungstabelle nach der zweiten Iteration

		Suffixe		
	Präfixe	*leer*	A, *leer*	B, *leer*
Zustände	*leer*	*wahr*	*wahr*	*wahr*
	A	*wahr*	*falsch*	*wahr*
	AA	*falsch*	*falsch*	*falsch*
	B	*wahr*	*wahr*	*falsch*
	BB	*falsch*	*falsch*	*falsch*
Transitionen	AAA	*falsch*	*falsch*	*falsch*
	AB	*wahr*	*wahr*	*falsch*
	AAB	*falsch*	*falsch*	*falsch*
	BA	*wahr*	*wahr*	*wahr*
	BB	*falsch*	*falsch*	*falsch*
	BBA	*falsch*	*falsch*	*falsch*

Die resultierende Beobachtungstabelle (Tabelle A-7) ist nun geschlossen und konsistent. Der resultierende Automat aus Beobachtungstabelle Tabelle A-7 ist in Abbildung A-3 dargestellt. Um aus der Beobachtungstabelle einen Automaten zu erstellen, wird zunächst die Anzahl der unterschiedlichen Zustände bestimmt (in diesem Fall drei: *leer*, „AB" und „BA" laut Tabelle A-7). Danach werden die Transitionen beginnend vom *leer* Zustand zugeordnet.

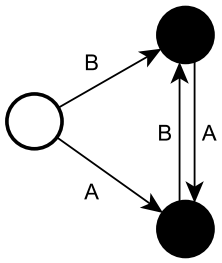

Abbildung A-4 Erlernter Automat resultierend aus Beobachtungstabelle Tabelle A-7

Wie zu sehen ist, ist der tatsächlich ausgeführte Automat (Abbildung A-1) bis auf den Startzustand identisch mit den erlernten Automaten (Abbildung A-4). Die Zugehörigkeitsfrage liefert somit kein Gegenbeispiel und der Algorithmus wird abgebrochen.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ARG	Advanced Reber-Grammatik
CAN	Controller Area Network
DEA	Diskreter Endlicher Automat
EA	Evolutionärer Algorithmus
ECU	Electronic Control Unit
ERG	Eingebettete Reber-Grammatik
FFT	Fast Fourier Transformation
FNN	Neuronales Feedforward Netz
ID	Numerische Identifikation
IT	Informationstechnologie
KNN	Künstliches Neuronales Netz
KRG	Kontinuierliche Reber-Grammatik
KERG	Kontinuierliche Eingebettete Reber-Grammatik
LSTM	Long Short Term Memory
MDL	Minimum-Deskription-Length
OoN	Out of Norm
PC	Personal Computer
RG	Reber-Grammatik
RNN	Recurrentes Neuronales Netz
TCP/IP	Transmission Control Protocol / Internet Protocol
UML	Unified-Modeling-Language
ZDEA	Zeitlich bestimmter Diskreter Endlicher Automat