Bundesamt
für Sicherheit in der
Informationstechnik

Deutschland
Digital•Sicher•BSI•

Formale Methoden und erklärbare künstliche Intelligenz

Teilergebnis der Projektforschung TK23

Änderungshistorie

Version	*Datum*	*Name*	*Beschreibung*
1.0	14.7.2022	Dr. Werner Lennartz, inducto GmbH	Erstellung und barrirefreie Übertragung von LaTeX

Tabelle 1: Änderungshistorie

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn
Tel.: +49 228 99 9582-5658
E-Mail: referat-tk23@bsi.bund.de
Internet: https://www.bsi.bund.de

Inhalt

1 Einleitung

Dieses Dokument gibt einen Überblick, wie die Technologie der Formalen Methoden die Bemühungen um Erklärbarkeit in der künstlichen Intelligenz (KI) unterstützen kann.

Frei nach Richard Feynman gilt, dass man, um etwas verstanden zu haben, in der Lage sein muss, es in einfachen Worten zu erklären. Um insbesondere biologische Systeme zu verstehen, hat die Neurowissenschaft im letzten Jahrzehnt das Prinzip der Minimierung der freien Energie vorgeschlagen, das verschiedene Aspekte von Handlung, Wahrnehmung und Kognition erklärt (Friston 2005). Dieses Prinzip nutzt die Variationsrechnung in der bayessche Inferenzstatistik, um zu verstehen, wie das Gehirn die Ursachen seiner sensorischen Eingaben ableitet. Die freie Energie kann dabei als die Überraschung, also die Differenz zwischen Erwartung und Erleben, interpretiert werden (Friston, Kilner, and Harrison 2006). Im folgenden soll das Bestreben, die Überraschung zu minimieren mit dem Minimieren des Risikos gleichgesetzt werden. Ziel ist es also diese Systeme, im folgenden auch Cyber-Physikalische Systeme, mit ihrer großen Anzahl von Freiheitsgraden trotzdem auf Stabilität und Zuverlässigkeit hin mathematisch zu verifizieren.

1.1 (Erklärbare) Künstliche Intelligenz

Während sich die klassische KI, genauso wie die Erklärbare KI, vor allem um die Entwicklung von Modellen kümmert, ist es für die Ermittlung des *Risikos* wichtig, die Systemkomponenten im Kontext ihrer Arbeitsumgebung zu berücksichtigen und zu analysieren. Ein Softwaresystem kann als KI betrachtet werden, wenn es über die grundlegenden Fähigkeiten verfügt, Daten in kontext-relevante Informationen zu übertragen und diese Informationen anschließend in Schlussfolgerungen (Wissen) abbildet (Vassev 2016). Die Modellentwicklung ist ein iterativer Entwicklungsprozess, in dem die relevanten Modelle abgeleitet, getestet und weiterentwickelt werden, bis ein Modell entsteht, das den gewünschten Kriterien entspricht. Der Prozess selber besteht aus einer Reihe von Subprozessen, die weit über die eigentliche Modellentwicklung hinausgehen:

- Problemdefinition: Prozessziele müssen spezifiziert werden.
- Datenbeschaffung: Daten in der richtigen Quantität und Qualität müssen beschafft werden.
- Datenvorbereitung: Fehlerkorrektur, Erweiterung, Zusammenfassung von Daten.
- Feature Engineering: Identifikation der relevanten Modellparameter/Features.
- Modellentwicklung: Low-Code-GUI-Modellentwicklung, Code-Based-Modellentwicklung, Auswahl aus hunderten von verfügbaren Modellen.
- Modellvalidierung: Überprüfung der Modellvorhersagen.

KI im allgemeinen hängt also von unserer Fähigkeit ab, unser Wissen effizient auf Softwaresysteme zu übertragen. Im folgenden soll das Konzept der Wahrnehmung, des Lernens und der Umweltabbildung als künstliches Bewusstsein beschrieben werden. Diese Systeme sind in der Lage, Veränderungen zu identifizieren, deren Auswirkungen zu verstehen und sowohl Musteranalyse als auch Mustererkennung anzuwenden, um normale von abnormalen Zuständen zu unterscheiden.

1.2 Künstliches Bewusstsein

Mit den wachsenden Anforderungen an computergestützte Systeme werden natürlich auch die Systemarchitekturen und -designs ihrer Autonomie immer komplexer. Die zuvor beschriebene Fähigkeit bzw. Anforderung, ohne externen Eingriff auf sich ändernde Zustände zu reagieren, wird als die **Autonomie einen bestimmten Grad an Qualität zu erbringen** bezeichnet.

Es sind die sogenannten Self-*-Fähigkeiten (De Lemos 2005), aufgrund derer ein System seine Autonomie erlangt. Diese Fähigkeiten haben einen Einfluss auf die grundlegenden Eigenschaften des Systems (Funktionalität, Benutzerfreundlichkeit, Leistung, Kosten und Zuverlässigkeit). Die Aspekte der Self-*-Fähigkeiten sollen also die Qualität der vom System bereitgestellten Dienste verbessern, das

Vorhersagerisiko minimieren und die Bereitstellungskosten senken. Aber was wird passieren, wenn die KI so programmiert ist, dass sie ihre Fähigkeiten zur Gefahrenerkennung selbst anpasst bzw. erlernt, um sie zu verbessern oder neue, zuvor unbekannte Gefahren zu erkennen. Diese Situation wird als technologische Singularität bezeichnet (Vinge 1993). Als Singularität wird hier die asymptotische Annäherung des Systems an eine Situation, in der die *normalen* Regeln nicht mehr gelten, verstanden. Eine Situation, in der wir nicht mehr in der Lage sind, Reaktionen und Verhalten dieses Systems vorherzusagen. Der Zeitpunkt, ab dem künstliche Intelligenz (KI) die menschliche Intelligenz übertrifft und sich dadurch rasant selbst verbessern und neue Erfindungen machen würde.

1.3 Formale Verifikation

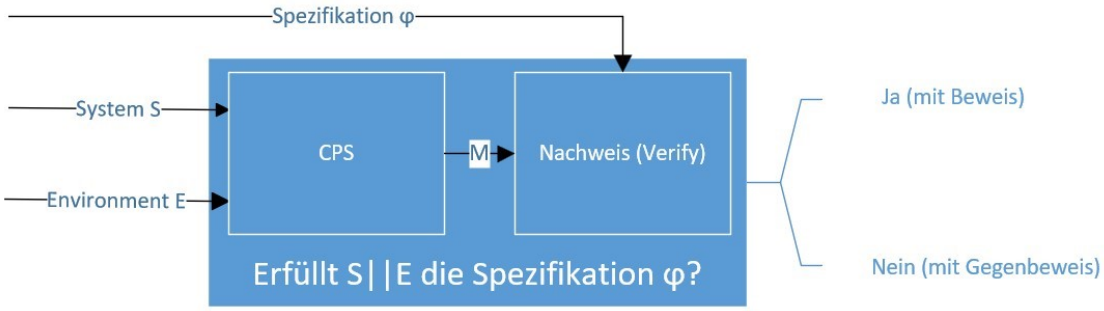
Der Begriff Formale Methode fasst die Techniken der Modellierung und mathematischer, rigoroser Überprüfung von Computersystemen zusammen, um deren Stabilität und Zuverlässigkeit mathematisch nachweisen zu können.
Überwiegend werden Formale Methoden verwendet, um die vom Benutzer möglichst vollständig spezifizierten Modelle auf Fehlerfreiheit zu analysieren. Jedoch werden immer häufiger kritische Anteile der algorithmischen Pipelines durch Modelle aus der künstlichen Intelligenz (KI) beschrieben und durch entsprechende Daten angelernt. Diese Aufgabe der Analyse von KI-Modellen, stellt die Gesamtheit der formalen Methoden vor die Herausforderung der Skalierbarkeit mit der anwachsenden Anzahl von Parametern moderner Modelle des maschinellen Lernens (ML). Es gilt also die Eigenschaften des angelernten Modells zu verifizieren, um die Erklärbarkeit, Robustheit und Sicherheit des KI-Systems (und seiner Reaktionen) nachzuweisen. Insbesondere führt das Fehlverhalten sicherheitskritischer Systeme häufig zu Verlust von Menschenleben und finanzieller Ressourcen. Techniken zum Schutz gegen solche Szenarien sind also für diese Systeme unerlässlich. Ziel dieses Dokuments ist es, die dazu notwendigen Ansätze für den Nachweis der Korrektheit durch automatisierte Deduktionen und ein exemplarisches Vorgehen zu beschreiben.

2 Motivation und Bezug

Mit steigender Komplexität elektronischer, insbesondere programmierbarer Systeme steigt die Anzahl der Möglichkeiten eines Systemversagens. In den frühen 1970er Jahren wurde man sich in der Prozessindustrie bewusst, dass bei größeren Industrieanlagen die Praxis des Lernens aus zuvor eingetretenen Fehlern nicht akzeptabel ist. Vor allem bei großen Beständen von Gefahrstoffen mussten Methoden entwickelt werden, die Gefahren zu identifizieren und die Folgen zu quantifizieren. Unter der Bezeichnung Funktionale Sicherheit ("Funktionale Sicherheit – Wikipedia" 2020) wurden in der Normenserie IEC 61508 Verfahrensweisen definiert, um die Risiken der Produktherstellung für Mensch und Umwelt in vertretbarem Rahmen zu halten. Ziel ist es, bereits mit der Entwicklung von Organisations- und Produktionsarchitektur eine überprüfbare Spezifikation zu schaffen (David J. Smith and Kenneth G.L. Simpson 2020). Gleichzeitig wird der Einsatz von ML in sicherheitskritischen Systemen zunehmend größer. Deep Learning spielt dabei mit seinen großen Wahrnehmungsfähigkeiten (*Perception*) eine besondere Rolle. Die Vielzahl der am Markt verfügbaren und erfolgreichen Modelle sind inzwischen praktisch allgegenwärtig.

Der Bereich der Formalen Methoden umfasst rechnergestütztes Beweisen (z. B. von mathematischen Theoremen) und Verifizieren von Spezifikationen. Die Aufgaben fallen in drei Kategorien:

- Spezifikation: WAS wird vom System erwartet
- Verifikation: WARUM erfüllt das System die Spezifikation
- Synthese: WIE wird erreicht, dass das System die Spezifikation erfüllt.



Standardsicht der Formalen Verifikation.

In der Abbildung Standardansicht der formalen Verifikation ist ein typischer Verifizierungsprozess abgebildet. S ist das Modell des Cyber-Physikalischen Systems, das verifiziert werden soll. E ist das Kontextmodell der Betriebsumgebung und φ spezifiziert die Eigenschaften des Cyber-Physikalischen Systems, die es erfüllen muss. Der Verifizierer generiert eine JA/NEIN Antwort, ob S die Anforderungen der Spezifikation φ im Kontext E erfüllt. Üblicherweise wird ein NEIN als Antwort von einem Gegenbeispiel (*Counterexample*) begleitet. Das Gegenbeispiel wird auch *Error Trace* genannt und beschreibt die Ausführung des Systems, sodass φ verletzt wird. Im Folgenden werden Beispiele autonomer Cyber-Physikalischer Systeme (CPS) aus Industrie und dem Bereich selbststeuernder Fahrzeuge behandelt. Um die formale Verifikation auf Cyber-Physikalische Systeme anzuwenden, muss man zumindest in der Lage sein, die drei Eingaben S, E und φ in Formalismen abzubilden, für die (idealerweise) effiziente Entscheidungsverfahren zur Beantwortung der JA/NEIN-Frage wie oben beschrieben existieren.

3 Informatische Herausforderungen autonomer Systeme

Cyber-Physikalische Systeme (CPS) gehören in die Klasse der Autonomen Systeme. Autonome Systeme sind Systeme, die selbstständig agieren, lernen, komplexe Aufgaben lösen und auf unvorhersehbare Ereignisse reagieren können. Dies bezieht sich nicht nur auf klassische Roboter, sondern auch auf intelligente Maschinen, Geräte oder Softwaresysteme, die in speziellen Bereichen im Interesse des Menschen eingesetzt werden.

Autonome Systeme weisen die folgenden sieben Merkmale auf, die in verschiedenen Kombinationen intelligentes Verhalten realisieren (Wahlster 2017):

Entscheidungsfähigkeit:
Ein autonomes System muss selbst frei entscheiden können, wie es eine vorgegebene Zielsetzung am besten erreicht, wenn es Wahlmöglichkeiten zwischen Handlungsalternativen gibt.

Selbstlernfähigkeit:
Das System kann ohne Hilfe von außen, rein aufgrund von Erfahrungsdaten und Beobachtungen seine Wissensbasis ergänzen und sein Problemlösungsverhalten optimieren.

Selbsterklärungsfähigkeit:
Das System kann seine Handlungsentscheidungen gegenüber einem Menschen in verständlicher, rationaler Weise erklären.

Resilienz:
Das System kann auch bei Funktionsausfällen in seinen Komponenten oder trotz massiver externer Störungen seine wesentlichen Leistungen aufrechterhalten und seine Aufgaben zumindest partiell weiter erfüllen.

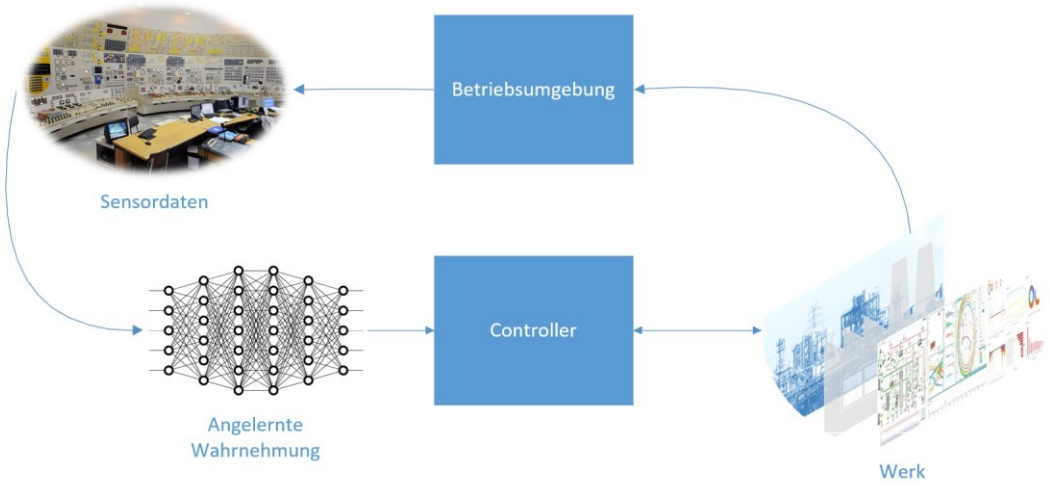
Kooperativität:
Das System kann mit anderen autonomen Systemen oder Menschen in seiner Umgebung im Team zusammenwirken, um seine Ziele zu erreichen. Es setzt auch vage artikulierte Aufträge und Änderungswünsche seines Betreibers um.

Ressourcenadaption:
Das System macht seine jeweilige Vorgehensweise abhängig von den aktuell verfügbaren Ressourcen (z. B. Zeit, Energie, Werkzeuge, Teammitglieder) und erkennt frühzeitig eigene Leistungsgrenzen.

Proaktivität:
Das System kann vorausschauend agieren und bei seiner Handlungsplanung zukünftig zu erwartende Ereignisse in seiner Umgebung antizipieren.

4 Formale Methoden

Formalen Methoden werden eingesetzt, um den Entwurfsprozess des Systems zu prüfen und weniger das Produkt selbst. Es handelt sich um technische Ansätze, die sich stark auf Tests stützen, um (bestimmte, aber nicht alle) Design- und Programmierfehler aufzudecken. Die Schwierigkeit beim Einsatz Formaler Methoden besteht in der Bereitstellung bzw. der Beschreibung der komplexen Umgebungsmodelle, ihrer Agenten und in den Anforderungen an das Herz des CPS, das künstliche neuronale Netzwerk. Neuere Arbeiten haben gezeigt, dass neuronale Netze trotz ihres enormen Erfolgs oft gefährliche Fehler machen. So wurde z. B. demonstriert, dass die meisten aktuellen neuronalen Netze oft nach kleinen vom Menschen nicht wahrnehmbaren Störungen oder nach Änderungen der Beleuchtung und Orientierung ansonsten regulärer Eingaben falsche Ausgaben erzeugen (Goodfellow, Shlens, and Szegedy 2015).



Ein auf maschinellem Lernen basierender Klassifizierer wird verwendet, um Prozesse im Produktionsumfeld zu erkennen und zu steuern. Nachgestellt aus (Dreossi et al. 2018).

Eine Aufgabe der Verifikation ist es, die Zustandsräume (*State Spaces*) des hochdimensionalen Dateneintritts (z. B. Bildinformationen oder Big Data vieler tausender Sensoren) zu durchlaufen und entweder die Gültigkeit sämtlicher Zustände zu beweisen oder aber Gegenbeispiele zu finden.

Zu Formalen Methoden unterscheidet man drei allgemeine Hauptmerkmale (Garavel and Graf 2013):

Sprachen:
Formale Methoden werden durch eine mathematische Notation bzw. durch eine Computersprache mit einer formalen Semantik ausgedrückt. Die Formalen Methoden beschreiben so die vom System erwarteten Eigenschaften bzw. die Art und Weise, wie das System entworfen wird (z. B. Architektur, Algorithmen, etc.). Eine zentrale Idee des Einsatzes formaler Methoden ist die objektorientierte Systemsicht auf die Hardware- oder Softwaresysteme, sodass diese streng beschrieben und analysiert werden können.

Werkzeuge:
Die entsprechenden Softwarewerkzeuge unterstützen den Ansatz der Formalen Methoden in den verschiedenen Entwicklungsphasen des Produktes. Der **Correct-By-Construction**-Ansatz überprüft das ordnungsgemäße Funktionieren des Systems. Die Verifizierung und Validierung von Spezifikationen überprüft die Einhaltung der Erwartungen an das System. Ein wichtiger Unterschied zwischen formalen Methoden und traditionellen Prüftechniken ist der Schwerpunkt der formalen Methoden auf der Analyse von (idealerweise) allen möglichen Ausführungen des Systems, und nicht nur einigen wenigen (Blackbox- und Whitebox-Testing).

Methoden:
Die effektive Integration der Formalen Methoden in die industrielle Praxis durch die Abbildung von Richtlinien und Normen gestattet eine konsequente Überwachung der ordnungsgemäßen Abläufe dieser Systeme.

4.1 Umgebungsmodellierung

Die Laufzeitumgebungen der KI-/ML-basierten Systeme sind häufig sehr komplex. Insbesondere ist selten sicher, wie die Laufzeitumgebung der Cyber-Physikalischen Systeme gestaltet ist, also welche und wie viele Agenten (weitere autonome Systeme – menschlich oder aus Software) sich in der Umgebung befinden. Beispiele dieser Umgebungen sind die Modelle von Verkehrssituationen oder industrieller Großanlagen. Aus der Perspektive der formalen Methoden ist es sehr schwer, realistische Umgebungsmodelle zu erstellen, mit denen man eine Verifikation oder Synthese durchführen kann. Zusammengefasst lassen sich die folgenden drei Hauptherausforderungen identifizieren:

Unbekannte Variablen:
Traditionell werden Dinge wie Cache-Kohärenz (CPU-Caches in Mehrprozessorsystemen), Gerätetreiber (Schnittstellen zwischen Steuerungssystemen und den zu steuernden Geräten) formal verifiziert. Es geht also um die wohldefinierte Schnittstelle zwischen dem System S und seiner Umgebung E. Für KI-basierte Systeme, wie z. B. das des autonomen Systems aus Abbildung Prozesse im Produktionsumfeld , kann es jedoch unmöglich sein, alle Variablen (Eigenschaften) der Umgebung genau zu definieren. Selbst in eingeschränkten Szenarien, in denen die Umgebungsvariablen (Agenten) bekannt sind, gibt es einen eklatanten Mangel an Informationen über deren Verhalten, insbesondere zur Entwurfszeit. Außerdem ist die Modellierung von Sensoren wie LiDAR, die die Schnittstelle zur Umgebung darstellen, an sich schon eine große technische Herausforderung.

Modellierung mit der richtigen Abstraktion:
In traditionellen Anwendungen der formalen Verifikation ist es normalerweise akzeptabel, die Umgebung als nicht-deterministischen Prozess zu modellieren. Typischerweise wird ein solches Umgebungsmodell als **überapproximiert** bezeichnet. Das bedeutet, dass es (viel) mehr Verhaltensweisen der Umgebung enthalten kann, als realistisch sind (Garavel and Graf 2013). Eine *überapproximierte Umgebungsmodellierung* erlaubt es also, eine solide Verifikation ohne ein detailliertes Umgebungsmodell durchzuführen. Für KI-basierte Autonomie führt jedoch eine rein nicht-deterministische Modellierung zu stark überapproximierten Modellen. Der resultierende Verifikationsprozess ist in der Praxis unbrauchbar, weil er zu vielen falschen Fehlerberichten führt. Außerdem treffen viele KI-basierte Systeme Verteilungsannahmen über die Umgebung, was eine probabilistische Modellierung erforderlich macht. Das Problem der zugrundeliegenden Verteilungen kann durch das Lernen eines probabilistischen Modells aus den zur Verfügung stehenden Daten gelöst werden. Die Modellparameter (z. B. Übergangswahrscheinlichkeiten) können dann allerdings nur zu Schätzungen und nicht für eine exakte Darstellung des Umgebungsverhaltens herangezogen werden. Daher können die Verifikationsalgorithmen das resultierende probabilistische Modell nicht als *perfekt* annehmen.

Modellierung des menschlichen Verhaltens:
Für viele KI-basierte Systeme sind menschliche Agenten ein wichtiger Teil der Umgebung und/""oder des Systems. Forscher haben versucht, Menschen als nicht-deterministische oder stochastische Prozesse zu modellieren, mit dem Ziel, die Korrektheit des Gesamtsystems zu verifizieren (Sadigh et al. 2014). Solche Ansätze müssen jedoch mit der Variabilität und Unsicherheit im menschlichen Verhalten umgehen. Man könnte einen datengesteuerten Ansatz wählen, der auf maschinellem Lernen basiert (z. B. per Reinforcement Learning (Ng, Ng, and Russell 2000)). Ein solcher Ansatz ist jedoch abhängig von der Aussagekraft der vom ML-Modell verwendeten Merkmale und der Qualität der Daten. Um eine verifizierte KI für solche *Human-in-the-Loop*-Systeme zu erreichen, muss man sich natürlich über die Grenzen der aktuellen menschlichen Modellierungstechniken und deren Unsicherheit im Klaren sein.

4.2 Formale Spezifikation

Die formale Verifikation hängt entscheidend davon ab, dass eine formale Spezifikation vorliegt – eine präzise, mathematische Aussage darüber, was das System tun soll. Die Herausforderung, eine qualitativ hochwertige formale Spezifikation zu erstellen, wird bei KI-basierten Systemen noch verschärft und lässt sich wieder in drei Hauptprobleme unterteilen:

Spezifikation schwer formalisierbarer Aufgaben:
Betrachten wir das Wahrnehmungsmodul (Perzeptron) in der Anlagensteuerung aus Abbildung Prozesse im Produktionsumfeld . Es muss Prozesszustände erkennen und klassifizieren, um eine Vorhersage treffen zu können, ob die spezifizierte Produktqualität erreicht wird. Spezifizierte Produktqualität bedeutet, dass das CPS im Zustandsraum immer einen Weg erlaubter Anlagenzustände durchläuft. Korrektheit für dieses Modul im Sinne klassischer formaler Methoden erfordert eine formale Definition jeder der möglichen Anlagenzustandstypen, was extrem schwierig, wenn nicht unmöglich ist. Ähnliche Probleme ergeben sich auch für andere Aufgaben, die Wahrnehmung und Kommunikation beinhalten (wie z. B. die Verarbeitung natürlicher Sprache). Wie also spezifiziert man die Korrektheit der Eigenschaften für ein solches Modul? Wie sollte die Spezifikationssprache aussehen und welche Werkzeuge kann man verwenden, um eine Spezifikation zu konstruieren?

Quantitative vs. Boolesche Spezifikation:
Üblicherweise bilden formale Spezifikationen das Systemverhalten boolesch auf wahr oder falsch ab. In KI und ML werden Spezifikationen jedoch oft als Zielfunktionen von Kosten oder Belohnungen angegeben. Optional müssen mehrere Ziele gemeinsam erfüllt oder gegeneinander abgewogen werden. Ziel ist es nun die booleschen und quantitativen Ansätze in der Spezifikation zusammenzuführen und Formalismen zu entwickeln, die Eigenschaften von KI-Komponenten wie Robustheit und Fairness auf eine einheitliche Weise erfassen.

Daten vs. formale Anforderungen:
Beim maschinellen Lernen ist die Betrachtung von *Daten als Spezifikation* üblich. Man bezeichnet die (eventuell von Hand) gelabelten Testdaten als *Ground Truth*, also das richtige Ergebnis, das man später mit der automatischen Klassifizierung erreichen möchte. Im Beispiel der LiDAR-Anwendung beim autonomen Fahren lässt man mit diesen (gelabelten) Daten dann eine *Clutter*-Klassifikation (also eine Stördaten-Klassifikation[1]) laufen, und vergleicht das Ergebnis mit dem *Ground Truth*. Zur Beurteilung kann dann z. B. die Prozentzahl zum Verhältnis der richtig erkannten Fläche zur Gesamtfläche oder zum Quotienten aus Anzahl richtig erkannter Datenpunkte und Gesamtzahl der Datenpunkte herangezogen werden. Auf der anderen Seite ist eine Spezifikation in formalen Methoden eine mathematische Beschreibung, die die Menge der korrekten Verhaltensweisen definiert. Diese Lücke gilt es zu überbrücken.

Die nächste Herausforderung besteht dann darin, effektive Methoden zu entwickeln, um gewünschte und unerwünschte Eigenschaften von KI- oder ML-basierten Systemen zu spezifizieren. Oft sind alleine die generellen Sicherheitsanforderungen an das System (*System Level Requirements*) unmöglich zu formulieren, weil das CPS einfach zu komplex ist (siehe Abbildung Risiken und Sicherheitsanforderungen).

[1] Siehe z. B. Wikipedia: Clutter.

Die Beschreibung von Risiken und Sicherheitsanforderungen auf der Systemebene sind häufig sehr komplex.

4.3 Modellierung Lernender Systeme

In der traditionellen Anwendung der Formalen Methoden wird ein genau bekanntes System *S* beschrieben (z. B. ein Softwareprogramm oder eine Schaltung), das in einer Programmiersprache oder Hardwarebeschreibungssprache modelliert wird. Beim Problem der Systemmodellierung geht es in erster Linie darum, die primäre Größe von *S* durch Abstraktion von irrelevanten Details auf eine besser handhabbare Größe zu reduzieren. KI-basierte Systeme führen zu einer ganz anderen Herausforderung für die Systemmodellierung, die sich vor allem aus dem Einsatz des maschinellen Lernens ergibt:

Hochdimensionaler Eingaberaum:
Die Eingaberäume von ML-Komponenten, die für die Wahrnehmung verwendet werden, haben in der Regel sehr viele Dimensionen. Abbildungen von Industrie 4.0 Produktionsanlagen (IIoT) generieren mittels ihrer immer größer werdenden Anzahl von Sensoren riesige Datenmengen (Mohammadi et al. 2018) und die Eingaberäume der optischen Beispiele aus dem Bereich der selbstfahrenden Autos bestehen aus vielen Millionen Elementen - jedes RGB-Eingabebild mit der Dimension 1000×600 Pixel, enthält 2561000×600×3 Elemente (Seshia, Sadigh, and Shankar Sastry 2020). Im Allgemeinen besteht der Dateneintritt aus einem Strom solcher hochdimensionaler Tensoren. Hinzu kommt, das die Struktur der Eingaberäume nicht vollständig boolesch, sondern hybride ist und sowohl diskrete als auch kontinuierliche Variablen enthalten.

Hochdimensionaler Parameter-/Zustandsraum:
ML-Komponenten wie Deep Neural Nets haben tausende bis Millionen von Modellparametern und *Primitives* (Algorithmen und Bibliotheken, der DNN-Schichten). Daraus ergibt sich ein riesiger Suchraum für die Verifikation, der eine sorgfältige Abstraktion erfordert.

Online-Evolution der lernenden Systeme:
Reinforcement-Learning-Systeme entwickeln sich weiter, wenn sie auf neue Daten und Situationen stoßen. Für solche Systeme muss die *Verifikation zur Entwurfszeit* entweder zukünftige Änderungen im Verhalten des Systems berücksichtigen oder inkrementell und in Echtzeit durchgeführt werden, während sich das lernende System weiterentwickelt.

Modellierung des Gesamtsystems im Kontext:
Für viele KI-/ML-Komponenten ist ihre Spezifikation nur durch den Kontext definiert. Um zum Beispiel die Robustheit eines DNNs wie in Abbildung Prozesse im Produktionsumfeld zu verifizieren, muss ein Modell

der Umgebung entwickelt werden. Man benötigt Techniken, um ML-Komponenten zusammen mit und in ihrem Kontext zu modellieren, so dass semantisch sinnvolle Eigenschaften verifiziert werden können.

4.4 Effizienter und skalierbarer Entwurf zur Verifikation von Modellen und Daten

Die Effektivität formaler Methoden in den Bereichen Hardware und Software wurde durch Fortschritte in den zugrundeliegenden *Berechnungswerkzeugen* vorangetrieben – z. B. SAT-Solver[2], SMT-Solver[3], numerische Simulation und Model Checking. Angesichts der Komplexität von KI-/ML-Systemen und ihrer Umgebungen sowie der neuen Herangehensweise bei der Erstellung von Spezifikationen, werden auch neue Computerprogramme (*Computational Engines*) für effizientes und skalierbares Training, Testen, Entwerfen und Verifizieren der KI-basierten Systemen benötigt. Im Folgenden werden die größten Herausforderungen noch einmal im Einzelnen beschrieben.

Datengenerierung:
Daten sind der grundlegende Ausgangspunkt für maschinelles Lernen. Um die Qualität eines ML-Systems zu verbessern, muss die Datenqualität verbessert werden. Für die formalen Methoden sollte ein Ziel sein, die Daten systematisch auszuwählen, zu gestalten und zu erweitern. Die für die KI-/ML-Systeme generierten Daten müssen den realen, durch die Sensoren generierten Daten entsprechen. Dazu werden statistische Ensembles mit der entsprechenden Verteilung erzeugt. Darüber hinaus muss die Datengenerierung selektiv sein, z. B. um die Anforderungen an die Größe und Vielfalt des Datensatzes für effektives Training und Generalisierung zu erfüllen.

Quantitative Verifikation:
Cyber-Physikalische Systeme sind fast immer auch sicherheitskritische Anwendungen (David J. Smith and Kenneth G.L. Simpson 2020). Dabei ist es unerheblich, ob sie KI-basiert sind oder nicht. Zusätzlich zur Skalierung der Systeme, gemessen an traditionellen Metriken (Dimension des Zustandsraums, Anzahl der Komponenten etc.), kommen noch Rahmenbedingungen der Interaktion mit ihrer Umgebung hinzu. Außerdem müssen Agenten der Umgebung (Menschen, Schnittstellen, Umweltanforderungen) als probabilistische Prozesse modelliert werden. Schließlich können die Anforderungen nicht nur traditionelle boolesche Spezifikationen für *Sicherheit* (nichts Schlimmes wird geschehen) und *Liveness* (irgendetwas Gutes wird schließlich herauskommen) sein, sondern es können auch quantitative Anforderungen an die Robustheit und Leistungsfähigkeit des Systems gestellt werden. Die meisten existierenden Verifikationsmethoden sind jedoch auf die Beantwortung boolescher statt quantitativer Verifikationsfragen ausgerichtet.

Compositional Verification:
Ein wichtiger Schlüssel zur Skalierbarkeit der Verifizierung großer Cyber-Physikalischer Systeme ist eine Art *Divide and Conquer*, die Unterteilung der Verifikation in kleinere Komponenten (z. B. Prozeduren). So ergeben die Komponentenspezifikationen zusammen die Spezifikation auf Systemebene (Pasareanu et al. 2018). Da es jedoch dazu kommt, dass Wahrnehmungskomponenten schlecht oder gar nicht spezifiziert werden können, müssen die Techniken für *kompositionale Schlussfolgerungen* in der Lage sein, mit nicht vollständigen kompositionalen Spezifikationen zu arbeiten. Zusätzlich muss mehr Arbeit geleistet werden, um die Theorie und Anwendung des *Compositional Reasoning* auf probabilistische Systeme und Spezifikationen zu erweitern (Seshia 2017).

[2] Eine Software zur Lösung des Erfüllbarkeitsproblem der Aussagenlogik SAT, siehe Erfüllbarkeitsproblem der Aussagenlogik .

[3] Die Satisfiability Modulo Theories sind eine Verallgemeinerung von SAT. Zur Lösung werden u.a. auch SAT-Solver herangezogen, vergleiche Satisfiability modulo theories .

4.5 Intelligente Systeme durch Correct-by-Construction

In einer idealen Welt sollte die Verifikation in den Designprozess integriert werden, so dass das System *Correct-by-Construction* ist. In der Hardwaremodellierung von integrierten Schaltkreisen wird dieser Ansatz beispielsweise durch eine Abstraktionsebene, die sogenannte Registertransferebene (*Register Transfer Level*, RTL) erreicht. Beim Entwurf auf dieser Ebene wird das System durch den Signalfluss zwischen den Registern spezifiziert. Dies könnte ebenfalls ein Ansatz für KI-basierte Systeme sein.

Spezifikationsgetriebener Entwurf von ML-Komponenten:
Eine Option ist, die ML"=Komponenten (das Modell) von Grund auf zu entwerfen, sodass nachweislich eine formale Spezifikation erfüllt wird. Dazu muss natürlich eine formale Spezifikation existieren und es müssen die folgenden Ebenen erfüllt werden:

- Entwurf des Datensatzes,
- Synthese der Struktur des Modells,
- Generierung eines guten Satzes von Merkmalen,
- Synthese der Hyperparameter (Steuerparameter des Lernprozesses, wie z. B. die Lernrate) oder anderer Einflussgrößen auf die Modellauswahl (z. B. Hidden-Layer-Größe),
- automatisierte Techniken zum Debuggen der ML-Modelle oder der Spezifikationen, wenn die Synthese fehlschlägt.

Kompositorische Design-Theorien:
Das CPS wird in mehrere lernende und nicht lernende Komponenten unterteilt. Für die digitalen Schaltungen und eingebetteten Systeme (Embedded Systems) existieren Theorien zum *kompositorischen Design* (z. B. (Nuzzo et al. 2014)). Allerdings existieren diese Theorien für KI-basierte Systeme noch nicht. Stellt man sich z. B. vor, dass zwei ML-Modelle für die Wahrnehmung auf zwei verschiedene Arten von Sensordaten (z. B. LiDAR und visuelle Bilder) angewendet werden und dabei jeweils einzeln ihre Spezifikationen unter bestimmten Annahmen erfüllen, stellt sich die Frage unter welchen Bedingungen können sie zusammen verwendet werden, damit die Zuverlässigkeit des Gesamtsystems verbessert wird?

Verbindung von Entwicklungs- und Laufzeitphase für eine belastbare KI:
Aufgrund der Komplexität von KI-basierten Systemen und ihrer Umgebungen, kann selbst nachdem ihre Spezifikation und Verifikation gelungen ist, ein sicherer Betrieb nicht gewährleistet werden. Daher müssen zur Laufzeit Fehlertoleranz- und Fehlerresilienztechniken eine entscheidende Rolle spielen. Dabei gibt es noch kein systematisches Verständnis darüber, was zur Entwurfszeit erreicht werden kann, wie der Entwurfsprozess zum sicheren und korrekten Betrieb des KI-Systems beitragen kann.

5 Das Prinzip von verifizierter KI

Nachdem im vorherigen Abschnitt die Herausforderungen zusammengetragen und beschrieben worden sind, werden nun fünf Prinzipien zur Umsetzung der oben beschriebenen Herausforderungen im Design-Verifizierungsprozess diskutiert:

1. Verwendung eines introspektiven, datengesteuerten und probabilistischen Ansatzes zur Modellierung der Umgebung;
2. Kombination der formalen Spezifikationen des Ende-zu-Ende-Verhaltens mit einem hybriden booleschen/""quantitativen Formalismus für lernende Systeme und Wahrnehmungskomponenten sowie Verwendung einer Spezifikationserfassung zur Überbrückung der Daten-Eigenschafts-Lücke;
3. Entwicklung von Abstraktionen, Erklärungen und semantischen Analysetechniken für die ML-Komponenten;
4. Schaffung einer neuen Klasse von kompositionellen, randomisierten und quantitativen formalen Methoden zur Datengenerierung, zum Testen und Verifizieren;
5. Entwicklung von Techniken zur formalen induktiven Synthese von KI-basierten Systemen und zum Entwurf von sicheren ML-Systemen, unterstützt durch Techniken zur Laufzeitsicherung.

5.1 Umgebungsmodellierung: Selbstbeobachtung, Wahrscheinlichkeit, Daten

Im Abschnitt Umgebungsmodellierung wurden die folgenden drei Herausforderungen zur Modellierung der Umgebung E eines KI-basierten Systems S beschrieben:

1. unbekannte Variablen,
2. Modelltreue,
3. Modellierung des Menschen.

Mögliche Antworten auf die Herausforderungen sollen nun im folgenden skizziert werden:

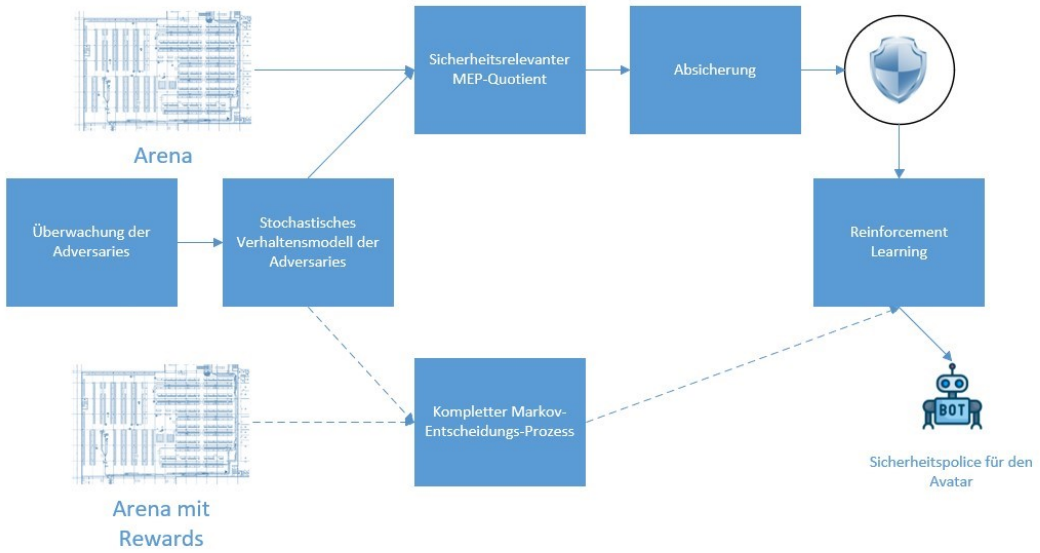
Introspektive Umgebungsmodellierung:
Das Problem der unbekannten Variablen kann durch die Entwicklung von introspektiven Entwurfs- und Verifikationsmethoden gelöst werden. Unter introspektiver Umgebungsmodellierung versteht man die durch Selbstbeobachtung des Systems entwickelten (synthetisierten) algorithmischen Annahmen A über die Umgebung, unter denen das System S einen korrekten Betrieb garantieren kann. Unter einem korrekten Betrieb versteht man die Erfüllung der Spezifikation φ (IEM 2019). Die Annahmen müssen so gewählt werden, dass sie zur Entwurfszeit effizient generierbar sind und zur Laufzeit mittels der verfügbaren Sensoren und anderer Informationsquellen ihre Umgebung überwachen können. Auf diese Weise können bei einer Verletzung der Annahmen entsprechende Korrekturmaßnahmen ergriffen werden. Darüber hinaus sollte das System S für die menschlichen Operatoren erklärbar sein. Es muss in der Lage sein, die Annahmen in einen Klartext zu übersetzen, sodass eine für den Menschen verständliche Begründung verfügbar ist. Die Erklärung beschreibt, warum die Spezifikation φ nicht erfüllt wurde. Der Umgang mit Anforderungen sowie die Notwendigkeit guter Sensormodelle macht die introspektive Umgebungsmodellierung zu einer nicht-trivialen Aufgabe, die weitere erhebliche Fortschritte in der Forschung erfordert (IEM 2019).

Aktive datengesteuerte Modellierung:
Ein kritischer Aspekt der Modellierung menschlicher Agenten ist die Erfassung menschlichen Bestrebens, seiner Intention. Hier könnte ein dreigleisiger Ansatz erfolgreich sein.

- Zuerst die Definition von Modellvorlagen/""Merkmalen auf Basis von Expertenwissen,

- dann die Nutzung von Offline-Lernen zur Vervollständigung des Modells, um eine Verwendung während der Entwurfszeit zu ermöglichen. Und schließlich
- das Lernen und Aktualisieren von Umgebungsmodellen zur Laufzeit mittels der Überwachung und der Interaktion mit ihrer Umgebung. Zum Beispiel die Datenaggregation bei Experimenten menschlicher Probanden in Fahrsimulatoren zur Generierung der Modelle des menschlichen Fahrerverhaltens. Diese Modelle können dann beispielsweise für die Verifikation und Steuerung von autonomen Fahrzeugen verwendet werden.

Probabilistische formale Modellierung:
Modelltreue kann erreicht werden, indem die Formalismen der probabilistischen und nicht-deterministischen Modellierung kombiniert werden. *Probabilistische Modellierung* kann dort verwendet werden, wo Wahrscheinlichkeitsverteilungen zuverlässig spezifiziert oder geschätzt werden können. Ein Beispiel dafür ist das Markov-Chain-Monte-Carlo-Verfahren (MCMC)[4]. In anderen Fällen kann die *nicht-deterministische Modellierung* verwendet werden, um das Verhalten der Umgebung zu überapproximieren. Das bedeutet, dass die Verifikation einseitige Irrtümer zulässt, also fehlerhafte Modelle immer, aber eventuell auch schon mal ein korrektes Modell ablehnt. Während Formalismen wie der Markow-Entscheidungsprozess (MEP, englisch *Markov Decision Process* oder MDP)[5] bereits eine Möglichkeit bieten, Wahrscheinlichkeit und Nicht-Determinismus zu kombinieren, bieten Techniken, die Wahrscheinlichkeit und logische oder automatentheoretische Formalismen (z. B. probabilistische Programmierung) zusammenführen, eine programmatische Methode zur Modellierung von Umgebungen. Diese probabilistischen Programme müssen natürlich (zum Teil) aus Daten gelernt/""synthetisiert werden. Der Formalismus des Markov-Entscheidungsprozesses bietet hier eine Möglichkeit die Unsicherheit in den Werten der gelernten Übergangswahrscheinlichkeiten darzustellen. Als Beispiel sei ein Fabrikflächenplan mit mehreren Korridoren angenommen. Die Knoten des Areals beschreiben Kreuzungen, die Kanten die Korridore mit Maschinen. Die Gegenspieler (Adversaries) sind (möglicherweise autonome) Transporter, die Teile innerhalb der Fabrik bewegen. Der Avatar modelliert eine Serviceeinheit, die sich umherbewegt und Maschinen, bei denen ein Problem aufgetreten ist, inspiziert und dabei das Verhalten der *Gegner* berücksichtigt (Jansen et al. 2020).



Workflow der Entwicklung einer Sicherheitspolice.

[4] MCMC-Verfahren
[5] Markow-Entscheidungsproblem

5.2 Ende-zu-Ende Spezifikation, Hybride Spezifikation und Spezifikationssuche

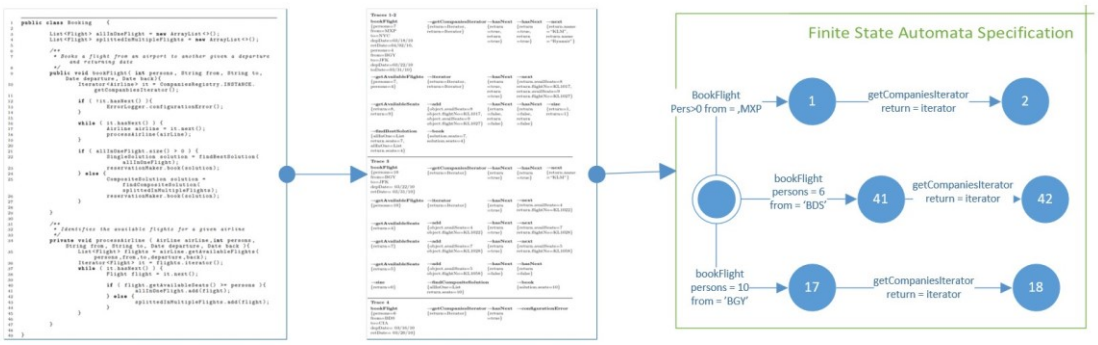
Das Schreiben formaler Spezifikationen für KI-/ML-Komponenten ist, wie schon mehrfach bemerkt, schwierig vor allem wenn die Komponente eine menschliche Wahrnehmungsaufgabe imitiert. Ein Ansatz zur Lösung ist die Befolgung der drei im Abschnitt Formale Spezifikation beschriebenen Leitprinzipien.

End-to-End/System-Level-Spezifikationen:
Um die Herausforderung der Spezifikation für die Wahrnehmung anzugehen, wird eine leichte Abänderung des Problems vorgeschlagen. Der Vorschlag sieht vor, sich zunächst auf die präzise Spezifikation des Ende-zu-Ende-Verhaltens des KI-basierten Systems zu konzentrieren. Ende-zu-Ende bedeutet, dass die Spezifikation für das gesamte, geschlossene System (siehe Abbildung Prozesse im Produktionsumfeld) oder ein genau spezifiziertes Teilsystem inklusive der KI-/ML-Komponente enthält. Das Teilsystem darf jedoch nicht ausschließlich die KI-Komponente sein. Eine solche Spezifikation wird auch als *Spezifikation auf Systemebene* bezeichnet. Für das Industriebeispiel bedeutet dies, eine Spezifikation φ, die der Einhaltung eines Mindestabstands zu den gefährdenden Objekten in Abschnitt Umgebungsmodellierung während der Bewegung, entspricht. Von dieser *Spezifikation auf Systemebene* (Ende-zu-Ende) werden Rahmenbedingungen für die Eingabe-Ausgabe-Schnittstelle der Wahrnehmungskomponente abgeleitet. Diese Rahmenbedingungen dienen als eine Detailspezifikation, unter der die Wahrnehmungskomponente analysiert werden kann.

Hybride quantitativ-boolesche Spezifikationen:
Boolesche und quantitative Spezifikationen haben beide ihre Vorteile. Einerseits sind boolesche Spezifikationen einfacher zu verfassen. Andererseits eignen sich Zielfunktionen zur Verifikation und Synthese sowie zur Definition einer feineren Granularität der Eigenschaften und ihrer Erfüllung. Ein Ansatz zur Überbrückung dieser Lücke ist der Übergang zu *quantitativen Spezifikationssprachen* durch Optimierung. Beispiele dafür sind Logiken mit sowohl boolescher als auch quantitativer Semantik (z. B. STL (Roehm et al. 2016)) oder das gewichteter Automaten.[6] Ein anderer Ansatz besteht in der Kombination der beiden booleschen und quantitativen Spezifikationen. Beispielsweise werden in einem Regelbuch (Censi et al. 2019) Spezifikationen in einer Hierarchie organisiert, verglichen und aggregiert. Zusätzlich werden neuartige Formalismen entwickelt, die Ideen aus formalen Methoden und maschinellem Lernen verbinden, um verschiedene Varianten der Eigenschaften wie Robustheit (einschließlich Vorstellungen von semantischer Robustheit (T. Dreossi, Ghosh, et al. 2019)), Fairness und Privatsphäre zu modellieren.

Specification Mining:
Um die Lücke zwischen Daten und formalen Spezifikationen zu schließen, wird das sogenannte *Specification-Mining* angewendet. Specification-Mining versucht Spezifikationen aus dem Systemverhalten, dem Sourcecode, den Verhaltensmustern und anderen Artefakten abzuleiten (David Lo, Siau-Cheng Khoo, Jiawei Han 2011). Solche Methoden könnten für ML-Komponenten im Allgemeinen oder Wahrnehmungskomponenten im Speziellen verwendet werden, da hier eben keine exakte oder eine für den Menschen lesbare Spezifikation vorliegt. Methoden des Specification-Minings könnten auch verwendet werden, um aus komplexeren Formen der Interaktion zwischen mehreren Agenten (Menschen oder Roboter), auf die *Absicht* und andere Eigenschaften zu schließen.

[6] Gewichteter Automat



Eindruck für die Anwendung von Software-Specification Mining

5.3 Systemmodellierung: Abstraktionen, Erklärungen, und Semantische Merkmalsräume

Im Abschnitt Modellierung Lernender Systeme wurden die Probleme und die Herausforderungen der Modellierung von selbstlernenden Systemen beschrieben. Selbstlernend bedeutet *Lernen aus Erfahrung*. Im Folgenden werden drei Ansätze beschrieben, um hier voran zu kommen:

Automatisierte Abstraktion:
Speziell zur Anwendung formaler Methoden auf große Hard- und Softwaresysteme kommt man nicht ohne die Abstraktion der Systeme aus. Insbesondere sollte die Abstraktion automatisiert geschehen. Gleiches gilt, um die Herausforderungen von sehr hochdimensionalen hybriden Zustands- und Eingaberäumen für ML-basierte Systeme zu meistern. Es müssen effektive Techniken entwickelt werden, um ML-Modelle in einfachere Modelle zu abstrahieren, die sich besser für die formale Analyse eignen. Einige Fortschritte in dieser Hinsicht verspricht die Verwendung von abstrakten Interpretationen zur Analyse von Deep Neural Networks (z. B. (Gehr et al. 2018)). Hier werden Abstraktionen zur Falsifikation von CPS mit ML-Komponenten genutzt. Das gleiche gilt für die Entwicklung von Garantien der ML-Algorithmen durch probabilistische Logiken.

Generierung von Erklärungen:
Die Modellierung eines ML-Systems ist offensichtlich einfacher, wenn der Anlerner seine Vorhersagen erklärt. Dies wird erklärungsbasierte Generalisierung genannt. So ergibt die Vorhersage zusammen mit den Daten plus Hintergrundwissen das gewünschte maschinelle Wissen. In letzter Zeit steigt das Interesse an der Verwendung von Logik, um die Ausgaben der Machine-Learning und Cyber-Physikalischen Systeme zu erklären (siehe (Mitchell, Keller, and Kedar-Cabelli 1986)). Ansätze zur Erzeugung von Erklärungen (XAI) in Kompatibilität zu den in formalen Methoden verwendeten Modellierungssprachen können die Aufgabe der Systemmodellierung für die Verifikation erheblich erleichtern. Das Hinzufügen kausaler und kontrafaktischer Schlussfolgerungen könnte die Robustheit, die Erklärbarkeit und die Ursache-Wirkung der Systeme erklären. Also die klassische Was wäre wenn-Fragestellung, die für die menschliche Intelligenz so wichtig ist (Pearl 2018).

Semantische Feature-Räume:
Um das Wettrüsten zwischen Angreifern und Verteidigern zu beenden, lautet ein Vorschlag, mehr Softwarewerkzeuge zur Verifizierung der Machine-Learning-Modelle zu entwickeln (Goodfellow, McDaniel, and Papernot 2018). Die Verifikation und kontradiktorische Analyse von ML-Modellen ist aussagekräftiger, wenn die generierten Eingaben und Gegenbeispiele eine semantische Bedeutung im Kontext der Cyber-Physikalischen Systeme haben. Es besteht also ein Bedarf an Techniken, die ML-Modelle im Laufzeitkontext des CPS analysieren können, indem z. B. eine *semantische adversariale Analyse* durchgeführt wird. Dazu wird der semantische Umgebungsraum des Modells in Abhängigkeit vom Eingaberaum des Systems betrachtet. Dies folgt der Intuition, dass der semantisch bedeutsame Teil des konkreten Merkmalsraumes (z. B. Bilder von Verkehrsszenen) einen viel niedriger-dimensionalen latenten Raum im Vergleich zum vollständigen

konkreten Merkmalsraum bildet. Als Beispiel dient hier erneut die Abbildung Prozesse im Produktionsumfeld. Nun sieht man sehr gut, dass der semantische Merkmalsraum der Prozessparameter weniger Dimensionen besitzt, als der konkrete Merkmalsraum der hochdimensionalen realen Pixel- und Parameterräume. Der niedriger-dimensionale semantische Merkmalsraum kann jetzt mit einem *Renderer* durchsucht werden. Der Renderer bildet dabei jeden Punkt im semantischen Merkmalsraum auf einen Punkt im konkreten Merkmalsraum ab. Sinnvoller Weise sollte der Renderer differenzierbar sein, um die Anwendung formaler Methoden und eine zielgerichtete Suche im semantischen Merkmalsraum zu erleichtern.

5.4 Kompositionelle und quantitative Methoden zum Entwurf und zur Verifikation von Modellen und Daten

Im Folgenden werden drei Ansätze vorgestellt, um die im Abschnitt Effizienter und skalierbarer Entwurf zur Verifikation von Modellen und Daten beschriebenen Herausforderungen der Entwicklung von Software für skalierbares Training, Testen und Verifizieren von Cyber-Physikalischen Systemen zu bewältigen.

Kontrollierte Randomisierung in formalen Methoden:
Das Problem des Datensatzdesigns – d. h. der systematischen Erzeugung von Trainingsdaten für die ML"=Komponente des CPS – benötigt zur Lösung mehrere Schritte:

1. Zunächst wird der Raum der gültigen Eingaben definiert, um valide Beispiele gemäß der Anwendungssemantik zu erhalten.
2. Nun werden die Grenzen der *Realität* gesetzt (z. B. eine Metrik für die Ähnlichkeit zu realen Daten).
3. Um Garantien für die Konvergenz des Lernalgorithmus zu bewahren, kann es notwendig sein, Beschränkungen für die Verteilung der generierten Beispiele vorzugeben.

Wie passen Formale Methoden in das Bild? Die Antwort liegt vermutlich in einer neuen Klasse *randomisierter formaler Methoden*. Also einer Klasse von randomisierten Algorithmen zur Erzeugung von Testeingaben, die den oben beschriebenen formalen Beschränkungen und Verteilungsanforderungen unterliegen. Genauer wird dies in einer Klasse von Techniken namens *Control Improvisation* beschrieben (Fremont et al. 2017). Ziel ist es in einer Spezifikationssprache einen *Improvisator* zu erzeugen, einen probabilistischen Algorithmus, der zufällig Wörter in der Spezifikationssprache generiert und drei Bedingungen erfüllt:

1. eine harte Randbedingung, die den Raum der gültigen Testeingaben x definiert,
2. eine weiche Randbedingung, die eine Metrik definiert. Eine Metrik, die generierte Testeingaben x in quantitativen Bezug zu realen Beispielen setzt,
3. eine Zufälligkeitsbedingung, die eine Beschränkung der Ausgabeverteilung definiert.

Anwendungsbeispiele sind hier Roboterüberwachung, Musikimprovisation oder randomisierte Varianten von Steuerungssystemen. Während die Theorie der Kontrollimprovisation, **Control Improvisation**, noch in den Anfängen steckt, stützt sich die Improvisation auf Berechnungsstrategien wie **Constrained Random Sampling**, **Model Counting** und **probabilistische Programmierung** (Seshia, Sadigh, and Shankar Sastry 2020). Constrained Sampling und Counting sind zwei grundlegende Ansätze der künstlichen Intelligenz. Beim Constrained Sampling besteht die Aufgabe darin, eine Zufallsstichprobe aus der Menge der Lösungen von Eingabebeschränkungen zu ziehen, während das Problem des eingeschränkten Zählens darin besteht, die Anzahl der Lösungen zu zählen. Beide Probleme haben zahlreiche Anwendungen, u. a. in der Wahrscheinlichkeitsrechnung, im maschinellen Lernen, in der Planung, in der statistischen Physik, im ungenauen Rechnen und in der eingeschränkten Zufallsverifikation (Meel et al. 2015). Die probabilistische Programmiersprache wird für den Entwurf und die Analyse von Cyber-Physischen Systemen, insbesondere solcher, die auf maschinellem Lernen basieren, eingesetzt. Sie bietet einen Ansatz, die Probleme der Erzeugung von Trainingsdatensätzen für seltene Ereignisse, der Fehlersuche und des Testens der Leistung unter verschiedenen Bedingungen zu vereinfachen. Die probabilistische Programmiersprache hilft das

Problem zu lösen, indem sie Verteilungen zur Kodierung der relevanten Eingabetypen spezifiziert und diese scannt. Ziel ist es, so spezielle Trainings- und Testmengen zu erzeugen. Zusammengefasst können die probabilistischen Sprachen für CPS und Robotik verwendet werden, um Umgebungsmodelle zu spezifizieren. Dies ist eine wesentliche Voraussetzung für die formale Analyse (Fremont et al. 2019).

Quantitative Verifikation im semantischen Merkmalsraum:
Eine Verifikation von quantitativen Anforderungen erfordert einen Verifizierer, der statt JA/NEIN einen numerischen Wert berechnet. Die Komplexität und Heterogenität von KI-basierten Systemen bedeutet, dass im Allgemeinen die formale Verifikation von Spezifikationen, ob boolesch oder quantitativ, nicht entscheidbar ist (zum Beispiel ist nicht einmal entscheidbar, ob ein Zustand eines linearen hybriden Systems, also mit linearen Zustandsübergängen, erreichbar ist). Spezifikationsformalismen, die sowohl die boolesche als auch die quantitative Semantik kennen, sind zum Beispiel die Formalismen der *metrischen temporalen Logik*. Temporale Logiken oder Zeitlogiken sind Erweiterungen der Logik, durch die zeitliche Abläufe erfasst werden können. Es handelt sich um Anwendungen der Modallogik, die auf einer Vorher-Nachher-Beziehung zwischen Zeitpunkten basieren. Ob daraus eine dichte oder diskrete Zeitordnung entsteht, ist von der Bestimmung genau dieser Relation abhängig. In Formalismen wie der metrischen temporalen Logik, ist die Formulierung der Verifikation als Optimierung entscheidend. Ziel ist die Vereinheitlichung der Berechnungsmethoden der formalen Methoden mit denen aus der Optimierungsliteratur (z. B. bei der simulationsbasierten Falsifikation in temporaler Logik) ((Dreossi et al. 2015)). Aus Effizienzgründen werden die formalen Methoden auf den semantischen Merkmalsraum angewendet (T. Dreossi, Ghosh, et al. 2019). Die Falsifikationstechniken können natürlich auch für die systematische, adversariale Generierung von Trainingsdaten für ML Komponenten verwendet werden. Techniken zur probabilistischen Verifikation (z. B. *Probabilistic Model Checking*) (Kwiatkowska, Norman, and Parker 2011), sollten mittels traditioneller Formalismen, wie Markov-Ketten oder Markov-Entscheidungsprozesse, entwickelt werden. So können die probabilistischen Programme über die semantischen Merkmalsräume verifiziert werden. In ähnlicher Weise geht man beim SMT-Solving (Satisfiability Modulo Theories) vor, um das SMT-Solving mit Optimierungsmethoden zu kombinieren.

Compositional Reasoning:
Compositional Reasoning ist eine Technik, mit der die Auswirkungen eines Zustandsraums mit hoher Dimension vereinfacht behandelt werden können. Im Wesentlichen ersetzt sie das Reasoning auf dem globalen Zustandsraum eines Programms durch ein lokalisiertes Schlussfolgern. Dabei wird jede Komponente separat analysiert. Wie bei allen Anwendungen formaler Methoden wird Modularität für die skalierbare Verifikation von KI-basierten Systemen entscheidend sein.

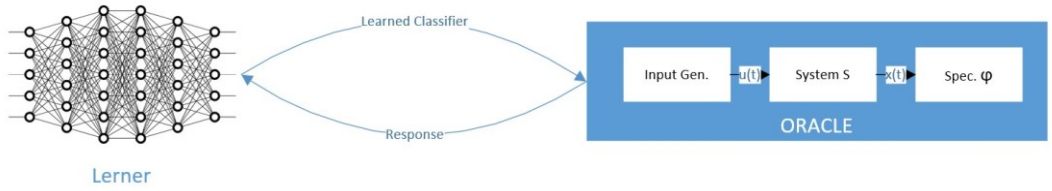
5.5 Formale induktive Synthese, Sicheres Lernen und Run-Time Assurance

Die Entwicklung einer *correct-by-construction* Entwurfsmethodik für KI-basierte Systeme (inklusive zugehöriger Werkzeuge) ist vielleicht die schwierigste Herausforderung. Voraussetzung dafür ist das Lösen der vier zuvor beschriebenen Herausforderungen. Ein Ansatz für die Methodik *Design for Verification* kann hier die Arbeit an den anderen vier Herausforderungen durchaus erleichtern.

Formale induktive Synthese:

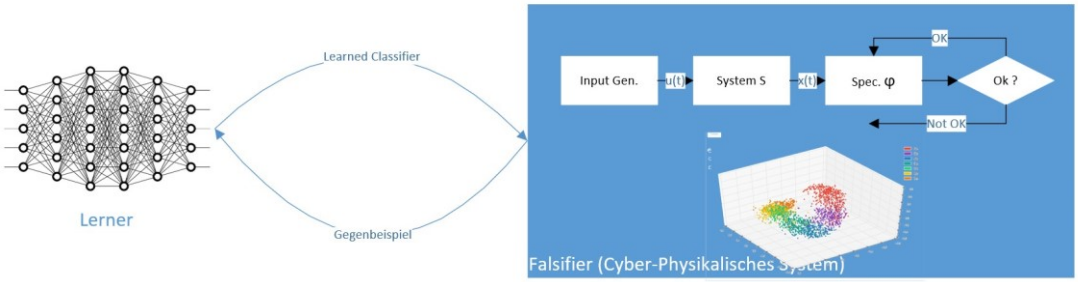
Zunächst betrachten wir das Problem der Synthese von *correct-by-construction* Lernkomponenten. Eine Lösung bietet die noch relativ neue Theorie der *formalen induktiven Synthese* (Jha and Seshia 2016). Die formale induktive Synthese ist die Synthese aus Beispielen von Programmen, die selbst formale Spezifikationen erfüllen. In Bezug auf maschinelles Lernen synthetisiert man die Modelle/""Klassifikatoren zu denen eine formale Spezifikation vorliegt. Also *induktive Synthese* bedeutet das Lernen von Beispielen und *formale induktive Synthese* bedeutet das Lernen von Beispielen, die einer formalen Spezifikation genügen. Üblicherweise wird zum Lösen eines formalen induktiven Syntheseproblems das sogenannte OGIS-Verfahren (*Oracle-Guided Inductive Synthesis*) verwendet. Bei OGIS wird der Lernalgorithmus, kurz

Lerner, mit der Befragung eines *Orakels* kombiniert. Die Menge der Abfrage/""Antwort-Rückgaben wird durch die Orakel-Schnittstelle definiert.



Oracle-Guided Inductive Synthesis (OGIS).

Für das Beispiel in Abbildung Prozesse im Produktionsumfeld kann das Orakel auch ein Falsifikator sein, der im Merkmalsraum Gegenbeispiele sucht und zurückgibt. Diese zeigen, wie ein Fehler der gelernten Komponente die Spezifikation auf Systemebene verletzt.

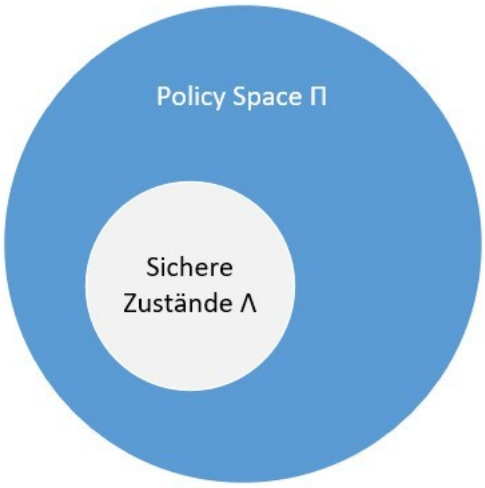


Counterexample-Guided Training of DNN (CEGIS).

Dieser Ansatz, auch bekannt als *gegenbeispiel-geführte induktive Synthese* und in Abbildung Counterexample-Guided Training of DNN veranschaulicht, hat sich in vielen Szenarien bewährt. Im Allgemeinen sind orakel-geführte induktive Synthesetechniken vielversprechend für die Synthese von gelernten Komponenten in Kombination mit menschlichem Expertenwissen, induktivem Lernen und deduktivem Schlussfolgerern.

Sicheres Lernen durch Design:
Hierbei geht es darum, Sicherheit in einen Lernprozess eines teilweise unbekannten Zustandsraummodells so zu integrieren, dass die unbekannte Dynamik als eine additive, begrenzte Störung betrachtet werden kann. Nach einigen Lernzyklen kann die Störung basierend auf realen Daten aktualisiert werden. Zu diesem Zweck wird eine Gaußsche Prozessregression der gesammelten Störungsproben durchgeführt (Fedorov and Candelieri 2018).

Ein prominentes Beispiel ist die sichere lern-basierte Prozesssteuerung. Die Steuerungsregeln (Control Policies) müssen hier direkt aus Experimenten lernen, ohne zuvor ein exaktes physikalisches Modell des Systems abzuleiten. Basierend auf einem endlichen zeitlichen Horizont und einer sicheren Zustandsmenge als Untermenge sämtlicher Systemzustände, muss die optimale Steuerungsregel zuvor berechnet werden, wie in Abbildung Alle Steuerungsregeln und die Menge der sicheren Regeln veranschaulicht. So wird sichergestellt, dass die Systemzustände immer innerhalb der Menge der sicheren Zustände bleiben. Bei diesem Ansatz wird eine Sicherheitshüllkurve vorberechnet und ein Lernalgorithmus verwendet, um eine Steuerung innerhalb dieser Hüllkurve abzustimmen. Es werden Techniken zur effizienten Berechnung solcher Sicherheitshüllkurven benötigt, die z. B. mit Hilfe der Lösung des Erreichbarkeitsproblems (Stichwort: Hamilton-Jacobi-Isaacs (HJI)-Erreichbarkeitsanalysen der Spieltheorie).



Alle Steuerungsregeln und die Menge der sicheren Regeln.

In diesem Zusammenhang können auch verschiedene Methoden für sicheres Reinforcement Learning in Betracht gezogen werden (siehe (García and Fernández 2015)). Und schließlich ist auch die Verwendung von *Theorem Provers* für die Sicherstellung der Korrektheit von ML-Trainingsalgorithmen ein wichtiger Schritt zur Verbesserung der Sicherheit in ML-basierten Systemen.

Absicherung zur Laufzeit:
In den meisten Fällen ist eine Verifikation nicht entscheidbar. Hinzu kommt das Problem der Umgebungsmodellierung. Nimmt man beide Probleme zusammen, so wird klar, dass es wohl nicht möglich ist, ein KI-basiertes System korrekt zu synthetisieren. Um einen korrekten Betrieb der Cyber-Physikalischen Systeme formal zu verifizieren, benötigen wir also eine Kombination aus Verifikation zur Entwurfszeit mit einer Laufzeitsicherung.

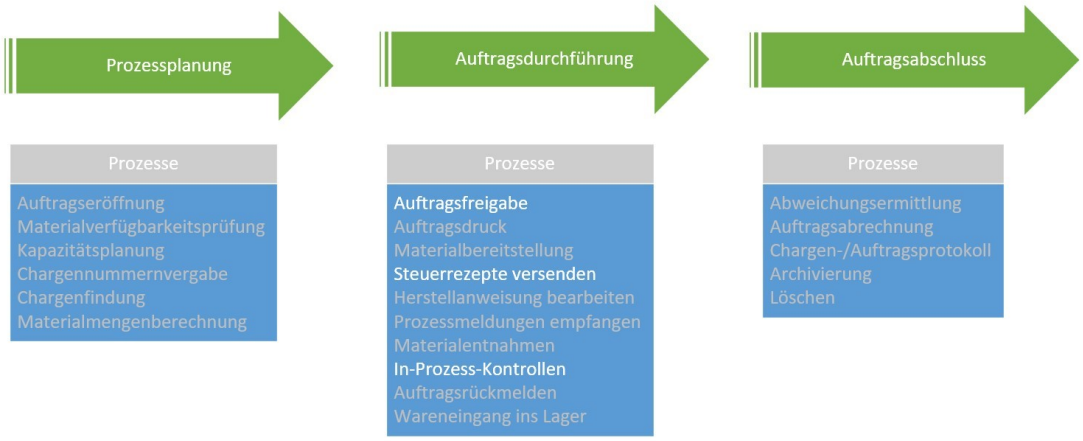
Ein Ansatz ist hier die Simplex-Technik (Sha 2001). Der Ansatz kombiniert ein komplexes, aber fehleranfälliges Modul mit einem sicheren, formal verifizierten Backup-Modul. Neuere Techniken zur Kombination von Entwurfs- und Laufzeitsicherungsmethoden (z. B. (Desai, Dreossi, and Seshia 2017)) haben gezeigt, wie unverifizierte Komponenten, einschließlich solcher, die auf KI und ML basieren, in ein Laufzeitsicherungs-Framework eingebunden werden können und so einen sicheren Betrieb garantieren.

6 Werkzeuge

Nachdem in den vorangegangenen Abschnitten vor allem die Motivation und die theoretischen Grundlagen im Vordergrund standen, werden in den folgenden Kapiteln mögliche Werkzeuge anhand eines konkreten Beispiels einer Kunststoffproduktion beschrieben. Das Beispiel ist dem Industrie 4.0-Kontext entliehen. Üblicherweise werden die Produktionsprozesse über Prozessleitsysteme gesteuert. In unserem Beispiel wird das Cyber-Physikalische-System durch ein KI-Modell (z. B. ein Neuronales Netz) unterstützt. Gemeinsam mit dem Ansatz die Blackbox des Modells per XAI erklärbar zu machen, gilt es nun mithilfe der Formalen Methoden die Steuerung abzusichern, indem der Trainingsdatensatz um mögliche Problemszenarien, bei dem das NN falsch reagiert, erweitert wird. Wir konzentrieren uns auf die Falsifizierung, weil das Hinzufügen erfolgreicher aber gegebenenfalls überflüssiger Regressoren zu einer Überanpassung (Overfitting) führen kann.

6.1 Motivationsbeispiel

In der Abbildung Lebenszyklus eines Prozessauftrag werden die allgemeinen Prozessanforderungen an eine Produktion der chemischen oder pharmazeutischen oder Lebensmittelindustrie abgebildet (Doller and Woelken 2014).

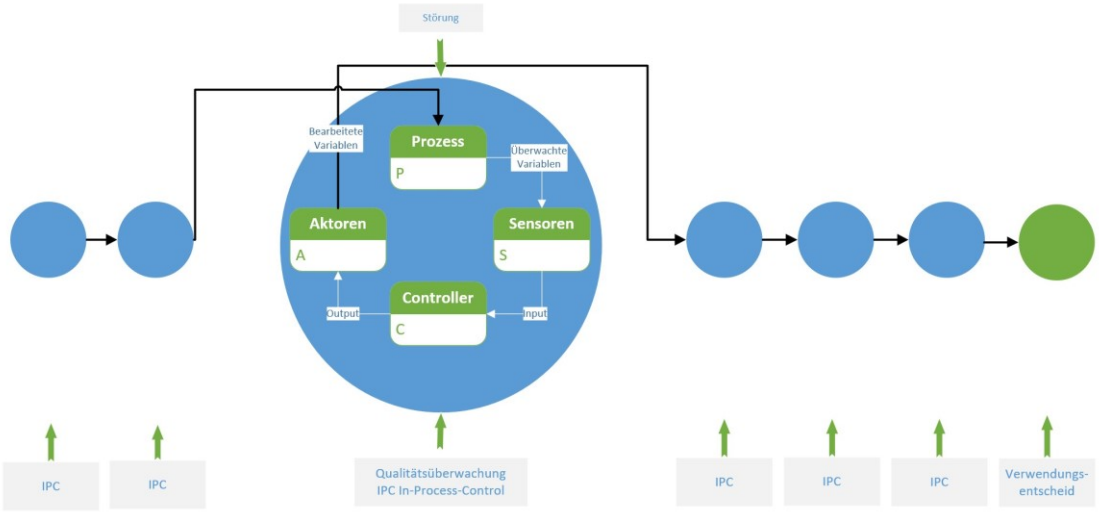


Lebenszyklus eines Prozessauftrag.

Kunststoffe sind Polymere (lange Ketten aus Monomeren) + Additive (z. B. Weichmacher und ölige Verbindungen). Um das Beispiel einfach zu halten, soll im Folgenden lediglich eine vereinfachte Kunststoffproduktion berücksichtigt werden. Nur die drei hervorgehobenen Prozessschritte sind relevant und das zu produzierende Kunststoffprodukt wird lediglich durch seine Viskosität spezifiziert. Die Einstellung der Viskosität wiederum wird gesteuert durch die Konzentration der Weichmacher und dem Anlagendurchsatz. Es gilt während der Produktion die Qualität zu garantieren, also die kundenspezifische Spezifikation einzuhalten. Dazu werden regelmäßige Proben (sogenannte IPC = *Inprocess-Control-Proben*) und eine abschließende Freigabeprobe genommen und ausgewertet.

6.1.1 Mathematische Darstellung

Die Prozesse der Industrie sind klassische Zustandsautomaten, die durch Zustandübergangsdiagramme (z. B. Petri Netze) dargestellt werden können.



Produktionsprozess.

Die in der Abbildung Produktionsprozesse abgebildete Abfolge von Systemzuständen lässt sich in der Linearen Temporalen Logik relativ einfach darstellen. Es lässt sich sogar zwischen kontinuierlicher und Batch-Fahrweise unterscheiden. Ein Batchprozess (englisch *batch*: Stapel) kommt aus der Reaktionstechnik. Es handelt sich dabei um Prozesse, die streng nacheinander abgearbeitet werden. Die Batchproduktion ist ein Spezialfall der diskontinuierlichen Produktion. Ein Batch oder eine Charge wird zum Beispiel durch das Fassungsvermögen eines Produktionsgefäßes (z. B. Reaktor, Mischer) begrenzte Materialmenge als Ganzes dem Arbeitssystem zugeführt und ihm als Ganzes nach Abschluss des Produktionsprozesses entnommen. Das Gegenstück zur chargenweisen Handhabung ist ein kontinuierlicher Prozess. Während in der kontinuierlichen Fahrweise der Prozess die Qualität ständig einhalten muss, reicht bei der Batch-Fahrweise, dass die Qualität zum Schluss des Batches erreicht wird. Mathematisch lässt sich das wie folgt ausdrücken:

Kontinuierliche Produktion:	$\blacksquare \psi R \phi$	Ist WAHR zu allen zukünftigen Momenten bis eine neue Produktion begonnen worden ist.
Batch-Produktion:	$\Diamond \psi U \phi$	ψ muss irgendwann WAHR sein und dann halten, bis ϕ eintritt.

ψ und ϕ sind Spezifikationen für aufeinanderfolgende Produktionen.

6.2 Sprachen

Wichtige Herausforderung bei der Entwicklung der CPS bzw. allgemeiner der ML-basierten Wahrnehmungssysteme sind:

- das Training des Systems in der Weise, dass es korrekt auf nur selten vorkommende Ereignisse reagiert,
- die Durchführung von Systemtests unter einer Vielzahl von Bedingungen, insbesondere ungewöhnlichen Bedingungen,
- die Fehlersuche im System, um die Ursache eines Fehlers zu verstehen und zu beseitigen.

Der traditionelle ML-Ansatz für diese Probleme ist das Erfassen von mehr Daten aus der Umgebung und damit jeweils ein Nachtrainieren des Systems, bis die Leistung des Modells angemessen ist. Die Hauptschwierigkeit dabei ist, dass das Erfassen von Daten aus der realen Welt langsam und teuer sein kann, da sie vor der Verwendung vorverarbeitet und korrekt beschriftet werden müssen. Außerdem kann es schwierig oder unmöglich sein, Daten zu Spezialfällen zu sammeln. Spezialfälle treten zwar selten auf, müssen aber dennoch trainiert und getestet werden (z. B. ein Autounfall oder Spezifikationsverletzungen). Aus diesem Grund werden in jüngerer Zeit synthetisch erzeugte Daten zum Trainieren und Testen der CPS genutzt. Synthetische Daten können in großen Mengen direkt mit korrekten Bezeichnungen erzeugt

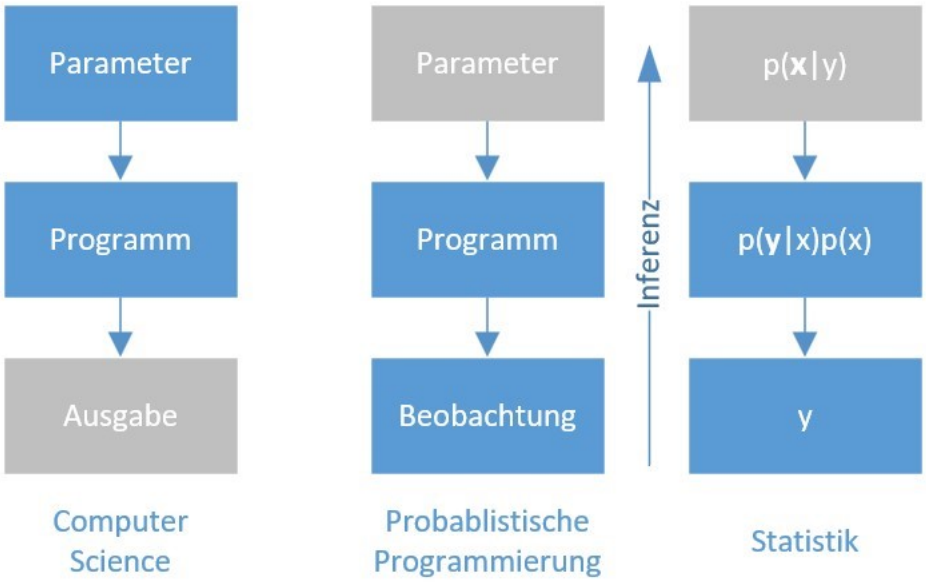
werden und geben dem Entwickler automatisch die volle Kontrolle über die statistische Verteilung der Daten. Ein Weg, die eingangs genannten Herausforderungen mit synthetischen statistisch verteilten Daten anzugehen, ist die Nutzung von probabilistischen Programmiersprachen für den Entwurf und die Analyse von Wahrnehmungssystemen.

6.2.1 Probabilistische Programmierung

Probabilistische Programmierung (PP) ist ein Programmierparadigma, bei dem statistische Modelle als Parametergrundlage genommen werden (Gordon et al. 2014; Van De Meent et al. 2018). Ein statistisches Modell E ist ein Tripel $E = (X, A, P)$

- einer Grundmenge$X$, die alle möglichen Ergebnisse des Experiments enthält,
- einer σ-Algebra A auf der Grundmenge X und
- einer Menge P von Wahrscheinlichkeitsmaßen auf (X, A).

Statistische Modelle beschreiben die Abbildung von Stichproben X in den Messraum (X, A). Programmiersprachen, deren Grundlage die probabilistische Programmierung ist, werden als *probabilistische Programmiersprachen* (PPL) bezeichnet. PP führt die Inferenz für die entwickelten Modelle automatisch durch. Sie stellt einen Versuch dar, probabilistische Modellierung und traditionelle Allzweckprogrammierung zu vereinheitlichen, um die klassische Programmierung zu vereinfachen und auf einer breiteren Basis insbesondere für KI-/ML-Systeme anwendbar zu machen. Sie kann eingesetzt werden, um Systeme zu erstellen, die helfen, Entscheidungen trotz Unsicherheit zu treffen. Die derzeit noch übliche Herangehensweise ist es, ein Programm zu schreiben, das die Argumente zu einem Ergebnis verarbeitet und ausgibt, wie in Abbildung Probabilistische Programmierung im linken Teil skizziert.



Motivation: Probabilistische Programmierung.

Der rechte Block in der Abbildung Probabilistische Programmierung veranschaulicht den statistischen, modellbasierten Ansatz. Beginnend mit der Ausgabe, also den Beobachtungen bzw. den Zielgrößen y der Daten, die dann in einem mathematischen abstrakten, generativen Modell $p(x, y)$ verarbeitet werden, um schließlich mittels Algebra und Inferenztechniken die posteriore Verteilung $p(x|y)$ und die unbekannten Größen im Modell zu bestimmen. Man spricht hier also über eine bayessche Interferenz. Schematisch für probabilistische Programmierung zeigt das folgende Listing Beispiel für ein probabilistisches Programm in einem Lisp-Dialekt:

```
(let [prior (beta a b)
      x (sample prior)
      likelihood (bernoulli x)
      y 1]
(observe likelihood y)
x))
```

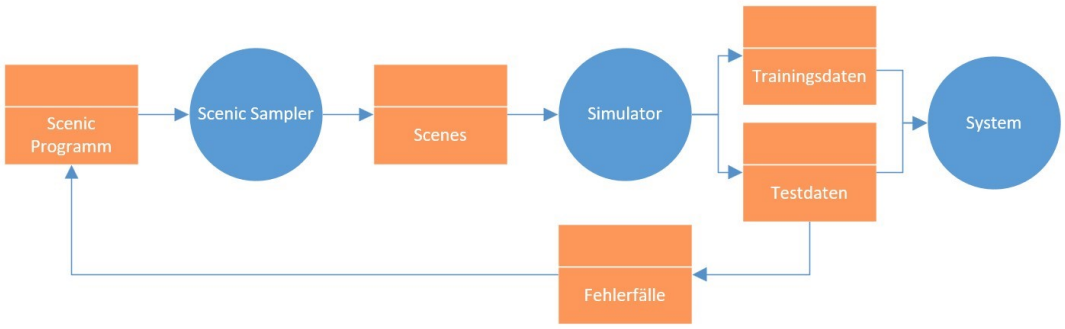
6.2.2 SCENIC

Scenic[7] ist eine domänenspezifische probabilistische Programmiersprache zur Modellierung der Umgebungen von Cyber-Physikalischen Systemen, wie sie in Industrie 4.0 vorkommen. Beispiele sind unter anderem Smart Devices, Roboter und selbstfahrende Autos. Ein Scenic-Programm definiert eine stochastische Verteilung über Szenarien, Konfigurationen von physischen Objekten und Agenten; Stichproben dieser Verteilung beschreiben konkrete, zu simulierende Szenarien, um Trainings- oder Testdaten zu erzeugen. Scenic ist in der Lage (probabilistische) Richtlinien für dynamische Agenten zu definieren, wodurch Szenarien modelliert werden können, in denen Agenten als Reaktion auf den Zustand der Welt im Laufe der Zeit Maßnahmen ergreifen. Als probabilistische Programmiersprache erlaubt Scenic die Zuweisung von Verteilungen zu Merkmalen der Szene, sowie die deklarative Auferlegung harter und weicher Beschränkungen für die Szene.

Synthetische Daten haben sich sowohl beim Training als auch beim Testen von Modellen des maschinellen Lernens wie z. B. neuronalen Netzen als zunehmend nützlich erwiesen. Das Hauptproblem bei der Generierung synthetischer Daten besteht darin, aussagekräftige Daten zu erzeugen, die nicht einfach nur zufällig gewählt sind, sondern Eigenschaften realer Daten widerspiegeln oder bestimmte Fälle von Interesse abdecken. Der traditionelle ML-Ansatz besteht sicher darin, solange mehr Daten aus der Umgebung zu sammeln, bis das System adäquat reagiert. Das Hauptproblem ist jedoch, dass an manche Daten nicht einfach zu gelangen ist. Reale-Welt-Daten sind häufig schwer oder unmöglich zu beschaffen. Ein Beispiel sind hier die Unfälle industrieller Anlagen. Der Einsatz synthetischen Daten bietet einen Ausweg, also das Trainieren und Testen der Systeme damit. Scenic zeigt, wie eine probabilistische Programmiersprache verwendet werden kann, um die Datensynthese zu steuern, indem Domänenwissen darüber, welche Daten sinnvoll sind, kodiert wird. Speziell konzentriert sich Scenic auf Datensätze, die aus Szenen, also Konfigurationen von physischen Objekten, entstehen. Beispielsweise Bilder von Autos auf einer Straße. Die Syntax von Scenic ist darauf ausgerichtet, komplexe Beziehungen zwischen den Positionen und Ausrichtungen von Objekten zu spezifizieren. Als probabilistische Programmiersprache erlaubt Scenic die Zuordnung von Verteilungen zu Merkmalen der Szene sowie die deklarative Auferlegung harter und weicher Randbedingungen für die Szene. Ein Scenic-Szenario definiert somit implizit eine Verteilung über Szenen, und formuliert das Problem des Samplings aus dieser Verteilung als Szenenimprovisation. Der implementierte Improvisator wird auf die Scenic-Szenarien angewendet. Ein Scenic-Szenario definiert also eine Verteilung über einzelne Szenen.

Die Generierung von Szenen aus einem Scenic-Programm erfordert Stichproben aus der darin implizit definierten Wahrscheinlichkeitsverteilung. Diese Aufgabe ist eng verwandt mit dem Inferenzproblem für imperative PPLs auf Basis von Beobachtungen (siehe Abbildung Probabilistische Programmierung). Die Idee besteht darin, PPLs zu verwenden, um allgemeine Laufzeitszenarien zu formalisieren und dann aus diesen Verteilungen Stichproben zu ziehen. Anhand der Stichproben wiederum können konkrete Umgebungskonfigurationen erzeugt werden (Fremont et al. 2019).

[7] Dokumentation unter Scenic's documentation.



Zusammenarbeit der Werkzeuge, um SCENIC für eine Umgebung zu trainieren, zu testen und zu debuggen.

Durch die Einspeisung der Konfigurationen in einen Simulator erhält man nun Bilder oder andere Sensordaten, die zum Testen und Trainieren des Cyber-Physikalischen Systems verwendet werden können. In Abb. Zusammenarbeit ist die allgemeine Vorgehensweise skizziert. Die Trainings- bzw. Testdatensätze müssen dabei nicht rein synthetisch sein, sondern können vorhandene, reale Daten ergänzen.

6.2.3 OCaml

OCaml[8], früher Objective Caml ist eine universelle, multiparadigmatische Programmiersprache[9], die den Caml-Dialekt der ML um objektorientierte Eigenschaften erweitert. Das Akronym CAML stand ursprünglich für *Categorical Abstract Machine Language* (kategorische abstrakte Maschinensprache). OCaml verzichtet auf diese abstrakte Maschine. OCaml ist ein freies und quelloffenes Softwareprojekt, das hauptsächlich vom französischen Institut für Forschung in Informatik und Automatisierung (INRIA) verwaltet und gepflegt wird. In den frühen 2000er Jahren wurden Elemente von OCaml von vielen Sprachen übernommen, insbesondere von F# und Scala. OCaml vereinigt funktionale, imperative und objektorientierte Programmierung unter einem ML-ähnlichen Typsystem. Daher müssen Programmierer mit dem rein funktionalen Sprachparadigma nicht besonders vertraut sein, um OCaml zu verwenden.

6.2.4 Einsatz der probabilistischen Sprachen

Im Folgenden werden die drei, im Zusammenhang mit formaler Verifizierung auftretenden Designprobleme, näher erläutert.

- **Testen verschiedener Bedingungen:** Das einfachste Problem ist das der Bewertung der Systemleistung unter verschiedenen Bedingungen. Wir können einfach Szenarien entwickeln, die systematisch jede Bedingung bestimmen und jeweils einen Testdatensatz erzeugen. Die Leistung des Systems kann dann unter diesen Bedingungen bewertet werden. Auch Bedingungen, die in der realen Welt nur selten auftreten stellen kein zusätzliches Problem dar. PPLs kodieren beliebige Bedingungen und können damit beliebig viele Instanzen generieren.
- **Training für seltene Ereignisse:** In Erweiterung des vorangegangenen Abschnitts können wir dieses Verfahren verwenden, um sicherzustellen, dass das System auch für ungewöhnliche Umstände Ergebnisse liefert. Es werden Instanzen für Szenarien seltener Ereignisse zur Erweiterung der ursprünglichen Trainingsdaten entwickelt. Das Hervorheben dieser Fälle in der Trainingsmenge kann die Leistung des Systems im schwierigen Fall verbessern, ohne die Leistung im typischen Fall zu beeinträchtigen.
- **Debugging von Fehlern:** Schließlich kann man das gleiche Verfahren verwenden, um Fehler im System zu verstehen und zu beheben. Wenn eine Umgebungskonfiguration gefunden wird, bei der das System versagt, kann ein entsprechendes Szenario entwickelt werden, das diese bestimmte Konfiguration

[8] OCaml
[9] Multiparadigmatische Programmierung

reproduziert. Da die Konfiguration als Programm kodiert wurde, ist es dann möglich, die Umgebung in einer Vielzahl von verschiedenen Richtungen zu erkunden. Dabei werden einige Aspekte der Szene festgehalten und andere variiert. Dies kann Aufschluss darüber geben, welche Merkmale der Szene für den Fehler relevant sind, um so die Grundursache zu identifizieren. Diese Ursache kann dann wiederum selbst in einem Szenario kodiert werden, das den ursprünglichen Fehler verallgemeinert. So kann ein erneutes Training ohne Überanpassung an das spezielle Gegenbeispiel ermöglicht werden.

Zusammengefasst zeigt sich, dass Probabilistische Programmiersprachen bei den Trainingsproblemen der cyber-physikalischen Systeme helfen können. Leistungen unter verschiedenen Bedingungen, seltene Ereignisse und Fehlversuche werden durch Verteilungen spezifiziert. Die entstehenden Umgebungsmodelle bilden also eine wesentliche Voraussetzung für jede Formale Analyse.

6.3 Motivationsbeispiel 1: Scenic

Im folgenden wird eine Fallstudie beschrieben. Als PPL wird eine an Scenic (Abschnitt Scenic) angelehnte Syntax verwendet, die durch eine domänenspezifische Struktur einen guten Zugang zu den Problemen bietet. Scenic erlaubt die Zuweisung von Verteilungen zu Merkmalen der Szene sowie die deklarative Auferlegung von harten und weichen Beschränkungen für die Szene. Als Motivation soll hier der zunehmende Einsatz von cyber-physikalischen Systemen in sicherheitskritischen Situationen dienen. Sicherheitskritisch ist nicht nur der Einsatz dieser Systeme in selbstfahrenden Autos sondern auch z. B. bei der Steuerung von Prozessleitsystemen in der Industrie. Wie bereits beschrieben, wird von CPS erwartet, dass sie mit wenig oder gar ohne menschliche Überwachung zuverlässig arbeiten. Sollten solche Systeme versagen, kann es zu katastrophalen Problemen kommen. Sie zuverlässig zu machen, ist aufgrund der Komplexität ihrer Umgebungen extrem schwierig. Eine Umgebung wie zum Beispiel einen Verkehrsfluss oder die Zustände eines Produktionsprozesses abzubilden, ist zunächst häufig eine Wahrnehmungsaufgabe, die nicht auf Pixel- oder Bitebene betrachtet werden kann, sondern als sogenannte Szenen behandelt werden. Szenen enthalten Objekte, Einstellungen, Positionen, Ausrichtungen usw. und damit die Grundwahrheiten für die notwendigen Trainings- und Testdatensätze. Das Sammeln der Szenen ist sehr aufwändig, da zusätzlich zur Erfassung noch eine manuelle Klassifizierung stattfinden müsste. Jedes Objekt, jede Einstellung hat zusätzlich noch eine große Reihe einstellbarer Parametern und deren Permutationen. Der Szenenraum ist also ziemlich groß und zufällig abgetastet erhält man bedeutungslose Szenen. Gleichzeitig unterliegen die Zustände jedoch geometrischen Einschränkungen durch Spezifikationen. Autos fahren typischerweise in Fahrspuren, Produktionsprozesse werden in verfahrenstechnischen Anlagen betrieben, Batchprozesse basieren auf Rezepturen. Die Arbeitsweise der Systeme in ihrer Umgebung kann durch Zufallsverteilungen kleiner Abweichungen von der Spezifikation beschrieben werden. Dies ist eine ideale Voraussetzung die Spezifikation in einem formalen Umgebungsmodell zu kodieren. Die probabilistische Programmierung hat das Ziel, diese Aufgabe zu vereinfachen. Ein probabilistisches Programm kann als Spezifikationen einer Wahrscheinlichkeitsverteilung auf seinen Ausgabewerten angesehen werden. Die Stichproben seiner Verteilungen enthalten perfekte Szenen, die dann in Simulationen und der Generierung von ML-Testdaten weiterverarbeitet werden können. Erkannte Fehler füllen Lücken in bestehenden Testdatensätzen ohne die Gefahr von Overfitting.

Durch den domänenspezifischen Ansatz ist es möglich, präzise Szenarien zu generieren mit der Möglichkeit fixierte und variable Prozessgrößen zu kontrollieren, wie in der Abbildung Spektrum der Szenarien schematisiert. Außerdem kann die Szene durch die Vorgaben von Verteilungen gesteuert werden.

Spektrum der Szenarien von allgemein bis spezifisch.

6.4 Plattformen

6.4.1 Imandra

Imandra[10] ist eine Programmiersprache und Argumentationsmaschine, mit der man die Eigenschaften von Programmen analysieren und verifizieren kann. Imandra ist eine Cloud-native Engine für automatisiertes *Reasoning* auf Basis der probabilistischen Programmiersprache ReasonML/OCaml. Insbesondere wird von Imandra die formale Verifikation und das symbolisches Schlussfolgern unterstützt. Imandra verfügt über viele fortschrittliche Funktionen, darunter berechenbare Gegenbeispiele, symbolische Modellprüfung, Unterstützung für polymorphe rekursive Funktionen höherer Ordnung, automatisierte Induktion, eine leistungsstarke Vereinfacher- und symbolische Ausführungs-Engine mit lemma-basierter bedingter Umschreibung und Vorwärtsverkettung, erstklassige Zustandsraumzerlegungen, Entscheidungsprozeduren für algebraische Datentypen, Gleitkommaarithmetik und vieles mehr.

Der Prozess ähnelt in gewisser Weise dem Schreiben von Tests, geschieht jedoch durch symbolische mathematische Analyse des Programmcodes und nicht einfach durch konkrete Testfallausführung. Der Entwicklungsprozess ist leistungsfähiger und erlaubt mathematisch zu beweisen, dass eine Behauptung für alle möglichen (eventuell unendlich vielen) Eingaben gilt, im Vergleich zu üblicherweise angelegten Unit-Tests oder zufällig generierten Eingaben.

6.4.2 VERIFAI

Prinzipiell ist VERIFAI[11] anwendbar auf eine Vielzahl von KI-Systemen. In diesem Rahmen konzentrieren wir uns auf KI-basierte cyber-physische Systeme (CPS) und insbesondere auf simulationsbasierte Verifikation, bei der der Simulator als Blackbox behandelt wird, um eine breite Anwendbarkeit auf das Spektrum der in der Industrie verwendeten Simulatoren zu gewährleisten. VERIFAI erwartet ein *Closed-Loop-CPS-Modell*, zusammengesetzt aus dem zu verifizierenden KI-basierten CPS-System und dem Umgebungsmodell. Das KI-basierte CPS umfasst typischerweise eine Wahrnehmungskomponente (nicht notwendigerweise auf ML basierend), einen Controller und die Anlage (d. h. das zu kontrollierende System). Unter diesen Voraussetzungen bietet VERIFAI die folgenden Anwendungsfälle (T. Dreossi, Fremont, et al. 2019):

[10] Imandra
[11] VerifAI

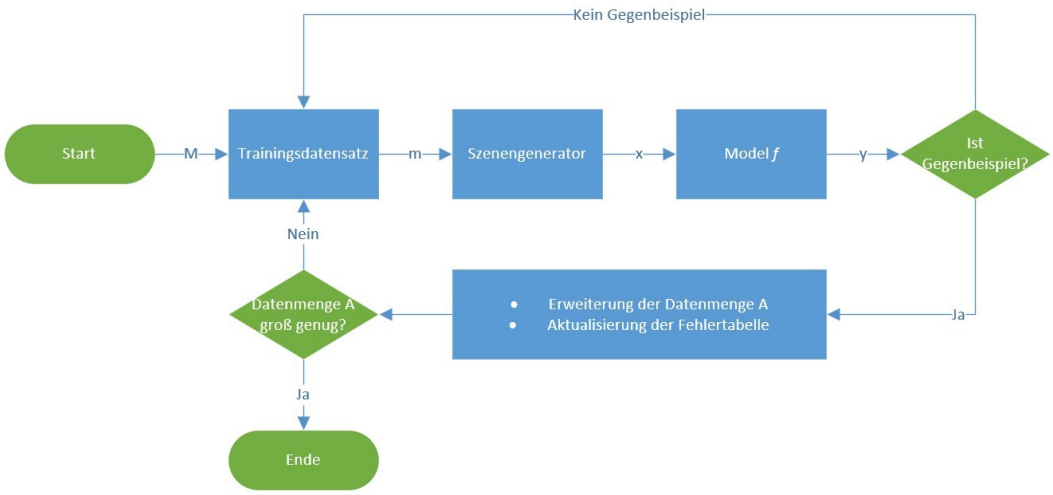
1. Temporal-logische Falsifikation
2. Modellbasiertes Fuzzy-Testing
3. Gegenbeispiel-geführte Datenerweiterung
4. Gegenbeispiel-(Fehlertabellen-)Analyse
5. Hyperparametersynthese
6. Modell-Parameter-Synthese

6.5 Motivationsbeispiel 2: VerifAI

Im oben genannten Anwendungsfall der Entwicklung von Gegenbeispielen sind die folgenden Punkte wichtig:

- Erweiterung des Datensatzes durch Gegenbeispiele, indem nur falsch klassifizierte Beispiele iterativ zu Trainingsmengen hinzugefügt werden;
- ein Generator für synthetische Prozessabläufe (Szenen), der realistische Gegenbeispiele erzeugt;
- Fehlertabellen, die Informationen über Gegenbeispiele speichern und deren Analyse Erklärungen liefert sowie die Erzeugung von Gegenbeispielsbildern erleichtert.

Die Abbildung Gegenbeispiel-getriebenes Verfahren fasst das Verfahren zur Generierung von Gegenbeispielen schematisch zusammen.



Gegenbeispiel-getriebenes Verfahren zur Augmentation.

Ziel ist es, die vorhandene Datenmenge um noch nicht identifizierte bzw. nicht im Trainingsdatensatz enthaltene Gegenbeispiele zu erweitern. Das Verfahren nimmt als Eingabe einen Modifikationsraum M, den Raum der möglichen Konfigurationen des verwendeten Szenengenerators. Der Raum M wird auf der Grundlage von Domänenwissen als ein Raum *semantischer Modifikationen* konstruiert, d. h. jede Modifikation muss in der Anwendungsdomäne, in der das maschinelle Lernmodell eingesetzt wird, eine Bedeutung haben. Auf diese Weise kann man eine sinnvollere Datenerweiterung vornehmen als nur durch die Erzeugung negativer Daten mittels Störung eines Eingabevektors (z. B. durch die negative Auswahl und Änderung einer kleinen Anzahl von Eingangsgrößen in einer Rezeptur).

In jeder Schleife wählt der Probenehmer eine Modifikation m aus M. Die Probe wird durch ein Stichprobenverfahren bestimmt, das durch eine vorberechnete Fehlertabelle beeinflusst wird, eine Datenstruktur, die Informationen über Bilder speichert, die durch das Modell falsch klassifiziert wurden. Die abgetastete Modifikation wird durch den Bildgenerator in ein Bild umgewandelt. Das Bild x wird als

Eingabe in das Modell f gegeben, das die Vorhersage $\hat{y}$ liefert. Dann wird geprüft, ob ein Gegenbeispiel gefunden wurde, also die Vorhersage $\hat{y}$ dazu falsch ist. Wenn dies der Fall ist, wird x zur Erweiterungsmenge A hinzugefügt und die Informationen über x (z. B. m, $\hat{y}$) in der Fehlertabelle gespeichert, die der Sampler bei der nächsten Iteration verwenden wird. Dieser Zyklus wird so lange wiederholt durchlaufen, bis die Erweiterungsmenge A groß genug ist (oder M hinreichend abgedeckt wurde).

7 Formale Methoden für künstliche neuronale Netzwerke

Die Verifikation von Modellen des maschinellen Lernens basierend auf künstlichen neuronalen Netzwerken (NN) ist immer noch ein wesentliches Gebiet aktueller Forschung, wie z. B. dem Lyra-Projekt (Lyra oder dem NASA Projekt SafeDNN (SafeDNN). Für NN werden Verifikationsziele definiert, die besonders mit der Sicherstellung der Robustheit des Netzes gegenüber Störungen der Eingaben und der Gewährleistung der Sicherheit der Ergebnisse verbunden sind. Man kann zwischen vollständigen und unvollständigen Verifikationsmethoden unterscheiden. Vollständige Methoden sind oft nur auf bestimmte neuronale Netzarchitekturen und Aktivierungen begrenzt, z. B. neuronale Netze mit stückweise linearen Schichten und ReLU-Aktivierungen, und skalieren nur eingeschränkt mit der Größe des Netzes. Die Vorgehensweise sieht die Kodierung des neuronalen Netzes zusammen mit den Randbedingungen, die die gewünschte Sicherheitseigenschaft der Ausgabe des Netzes kodieren, zum Beispiel, dass die Ausgabe immer innerhalb bestimmter Sicherheitsgrenzen liegt, vor. Die Kodierungen werden dann an einen Black-Box SMT-Solver weitergeleitet (Pulina and Tacchella 2010, 2012; Scheibler and others 2015).

Das SW-Werkzeug **Planet** (Planet) von R. Ehlers kodiert das NN in der Form einer Kombination von linearen Constraints, die zusammen mit der Negation der gewünschten Sicherheitseigenschaft formal verifizierbar ist (Ehlers 2017).

Von Katz et al. (2017) (Katz and others 2017) wurde ein spezialisierter SMT-Solver, genannt **Reluplex** (ReluplexCav2017), vorgeschlagen, um Sicherheitseigenschaften von *Feed-forward Fully-connected Neural Networks* mit ReLU-Aktivierungen zu überprüfen. Dieser Code wurde zuvor für die Verifikation des ACAS Xu (Julian and others 2016), eines prototypischen Kollisionsvermeidungssystem für unbemannte Luftfahrzeuge, angewendet. Reluplex benötigt mehrere Stunden für die Überprüfung (in einem Fall über zwei Tage).

Marabou (Marabou) ist ein von Katz et al. (2019) (Marabou) entwickeltes Nachfolgeprogramm von Reluplex, das auch in paralleler Ausführung laufen kann. Marabou arbeitet mit symbolischen Grenzen für alle Neuronen, die als Linearkombinationen der Eingangsneuronen ausgedrückt werden (FSA 2018b). Eigene, engere Grenzen für den Eingaberaum können somit bestimmt werden. Marabou hat Implementierungsverbesserungen gegenüber Reluplex erhalten, die mehrere Vorteile mit sich bringen. Zum Beispiel, integriert es einen eigenen Simplex-Löser. Experimentelle Auswertungen zeigen, dass Marabou (im Divide-and-Conquer-Modus, der auf vier Kernen läuft) im Allgemeinen sowohl Planet (Ehlers 2017) als auch Reluplex übertrifft und es ermöglicht, alle ACAS Xu Benchmarks innerhalb einer Stunde pro Benchmark zu überprüfen.

Die Überprüfung von Eigenschaften tiefer neuronaler Netze stellt besondere Herausforderungen. Binäre NN (BNN) können in Form einer booleschen Formel dargestellt und mittels eines **SAT-Solvers** überprüft werden (Narodytska and others 2018). Dieser Ansatz konnte Robustheit und Äquivalenzeigenschaften für neuronale Netze, trainiert mit MNIST und CIFAR-10 Datensets, erfolgreich überprüfen.

Einen Ansatz für die formale Überprüfung der Sicherheitseigenschaften von DNNs ohne Verwendung eines SMT-Solvers haben Shiqi Wang et al. (FSA 2018b) unter dem Namen **ReluVal** (ReluVal) veröffentlicht. Es wird Intervall-Arithmetik benutzt, um strenge Grenzen für die DNN-Ausgaben zu berechnen. Eine symbolische Intervallanalyse und mehrere Optimierungen können die Einschätzung der Ausgabegrenzen sehr genau und viel effektiver als Reluplex berechnen. Stehen nur vier CPU-Kerne zur Parallelisierung zur Verfügung, ist ReluVal auch dem Marabou-Programm überlegen, dass wiederum mit 64 verfügbaren Kernen die Führung übernimmt.

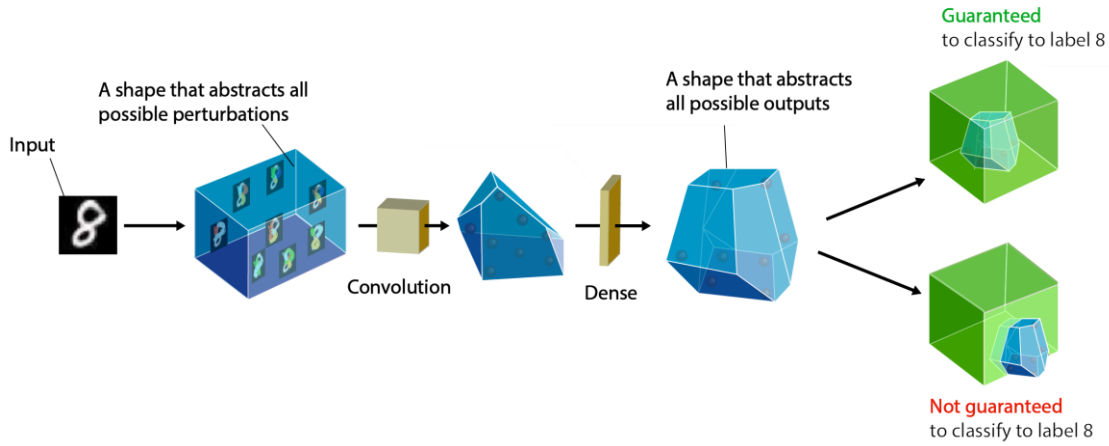
Als verbesserten Nachfolger von ReluVal haben Wang et al. (Wang and others 2018a) das Werkzeug **Neurify** (Neurify) implementiert und vorgestellt. Unter anderem auf dem auf neuronalen Netzen basierenden ACAS Xu erprobt erwies sich Neurify als wesentlich schneller als ReluVal und um drei Größenordnungen schneller als Reluplex.

Innovative neuere Ansätze für die Analyse und Verifikation von DNN benutzen so genannte *Star Sets* für die symbolische Darstellung von Sätzen von System-Zuständen. *Stern*-basierte Algorithmen werden für die Verifikation der Robustheit von ML-Methoden, zum Beispiel implementiert als DNN, eingesetzt (Tran and others 2020). Ein für diese Methode entwickeltes *Neural Network Verification Tool* (**nnv**) ist im Sourcecode unter nnv verfügbar. Der in diesem Software-Werkzeug implementierte stern-basierte Erreichbarkeitsproblem-Solver berechnet alle erreichbaren Zustände, und deshalb den ganzen Satz von Gegenbeispielen, die eine Sicherheitseigenschaft verletzen. Die ACAS Xu Netzwerke können hiermit hundertmal schneller und komplett verifiziert werden.

Eine weitere Implementierung eines stern-basierten Tools in Python steht mit der Bibliothek **nnenum** von Stanley Bak unter nnenum im Sourcecode zur Verfügung, die die Berechnung aller ACAS Xu Benchmarks auf einem Standard-Laptop-Computer in weniger als 10 Minuten ermöglicht (Bak 2021).

Auch Ansätze basierend auf MILP (*Mixed Integer Linear Programming*) erlauben die Implementierung von NN-Verifikationswerkzeuge, die Robustheit auch für faltende und residual Netze ermöglicht (Tjeng, Xiao, and Tedrake 2019): das Open-Source Paket **MIPVerify** für die Programmiersprache Julia (MIPVerify.jl) ist um zwei bis drei Größenordnungen schneller als Reluplex bei der Entdeckung von Adversarial-Beispielen bei gleicher Größenordnung der Störung der Eingabe.

Singh et al. (BRCNN 2019c) haben ein sogenanntes *RefineZono*-Verfahren, eine Kombination aus abstrakter Interpretation und (Mixed Integer) linearer Programmierung, vorgeschlagen, um lokale Robustheit gegen Adversial-Störungen (Szegedy and others 2014) für neuronale Netzwerke mit stückweise linearen Lagen (vollständig verbundene, Max-Pooling- und Faltungs-Lagen) und ReLU-Aktivierungen zu beweisen. Der Ansatz von (BRCNN 2019c) macht Gebrauch von einer abstrakten Domäne von Zonotopen (Ghorbal, Goubault, and Putot 2009), ausgerüstet mit spezialisierten Transformern zur Über-Approximation der Aktivierungen des neuronalen Netzes (Singh and others 2018). Die Veröffentlichung begleitender, in Python programmierter Code mit Bilder-Datensatz ist unter eran als ETH-Robustheits-Analyseprogramm für neuronale Netzwerke (**ERAN**, Abbildung Veranschaulichung der Verifikation) verfügbar.



Veranschaulichung der Verifikation eines NN mit ERAN.

Unvollständige Formale Methoden benutzen zur Beschreibung der Netzwerke eine abstrakte Spezifikationssprachen-Semantik und abstrakte Domänen und Intervalle, die auch als Zonotopen und Polyhedra in der Literatur bekannt sind. Diese Methoden leiden an Genauigkeitsverlust (Tradeoff zwischen Präzision und Skalierbarkeit), können jedoch viel besser mit der Größe der Netze skalieren und müssen sich nicht auf einen Typ von Netzwerk begrenzen. Bekannter Nachteil der Methoden ist, dass sie falsche Positive liefern. Zu diesen Methoden gehört das **AI^2**-Framework, vorgeschlagen von Gehr et al. (Gehr and others 2018). AI^2 unterstützt abstrakte Domänen von Intervallen, Zonotopen und Polyedern, ebenso wie ihre begrenzenden Potenzmengen[12].

[12] Wikipedia: Potenzmenge

Um Störungen (Perturbationen), die bei Adversial-Angriffen auf neuronale Netzwerke bei den Eingabe-Pixeln vorgenommen werden, zu quantifizieren, wird gewöhnlich eine Abstandsmetrik entsprechend einer L_p-Norm, mit Werten von 0, 2 oder ∞ für p, verwendet. Die L_∞-Norm misst das Maximum der Änderung irgendeines der Pixel (Ponakala 2019).

In der empirischen Bewertung des AI^2-Verfahrens wurde die Robustheit gegenüber Pixel"=Aufhellungsstörungen (bis hin zu einer Robustheitsgrenze von $\delta = 0{,}085$) von Bildklassifizierungs-Netzwerken, die mit den Datensätzen MNIST (Le Cun, Cortes, and Burges 1998) und CIFAR-10 (Krizhevsky 2009) trainiert wurden, getestet. Die untersuchten Netzwerke bestanden aus bis zu 53.000 künstlichen Neuronen (Le Cun and others 1989). Die Ergebnisse zeigen, dass Domänen von Zonotypen einen guten Kompromiss zwischen Genauigkeit und Skalierbarkeit der Analyse bieten. Es dauert im Durchschnitt weniger als 10 Sekunden, um die lokale Robustheit für neuronale Feed-Forward-Netze mit bis zu 18.000 versteckten Neuronen, die auf MNIST trainiert wurden, zu verifizieren. Bei den größten neuronalen Faltungsnetzen, die mit CIFAR-10 trainiert wurden, ist jedoch der Unterschied in der Präzision bezogen auf den weniger aussagekräftigen Bereich der Intervalle weniger signifikant.

DeepZ von Singh et al. (Singh and others 2018) kann mit ReLU-Aktivierungen ebenso wie mit Sigmoid- und Tanh-Aktivierungen umgehen. In der experimentellen Auswertung wurden Störungen bezüglich des L_∞-Abstands (bis hin zu $\epsilon = 0{,}3$) bei neuronalen Netzwerken zur Bildklassifikation (mit bis zu 88.500 internen Neuronen), die ebenfalls mit den MNIST- und CIFAR-10-Bilderdatensätzen trainiert wurden, betrachtet. Die Ergebnisse zeigen, das DeepZ signifikant präziser und schneller ist als AI^2. Darüber hinaus ist DeepZ in der Lage, lokale Robustheitseigenschaften der größten neuronalen Faltungs-Netzwerke in Minutenschnelle nachzuweisen.

DeepPoly und seine präzisere Version **k-ReLU**, die beide von Singh et al. (ADCNN 2019a, BRCNN 2019b) vorgeschlagen worden sind, ordnen jedem Neuron eines neuronalen Netzes eine konkrete untere und obere Schranken sowie symbolische Schranken, ausgedrückt als Linearkombinationen der Aktivierungen der Neuronen in der vorhergehenden Schicht des Netzes, zu. In einer auf arXiv zur Diskussion gestellten Arbeit haben Müller et al. (Müller and others 2020) Algorithmen zur effizienten Ausführung von DeepPoly auf einer GPU, genannt **GPUPoly**, vorgestellt. Die experimentelle Auswertung ergab, dass damit eine Analyse mit DeepPoly bis zu 170 Mal schneller abläuft als bei der Ausführung nur auf der CPU. Mit diesem Algorithmus skaliert DeepPoly auch gut auf große neuronale Netze mit bis zu 967.000 Neuronen, die auf den CIFAR-10 Datensatz trainiert worden sind. Die Verfahren GPUPoly, k-ReLU, DeepPoly und DeepZ sind alle in dem oben vorgestellten, ERAN genannten Robustheits-Analyse-Werkzeug für NN der ETH Zürich implementiert und können mit dem auf eran im Sourcecode nebst Anleitung verfügbaren Softwarepaket erprobt werden.

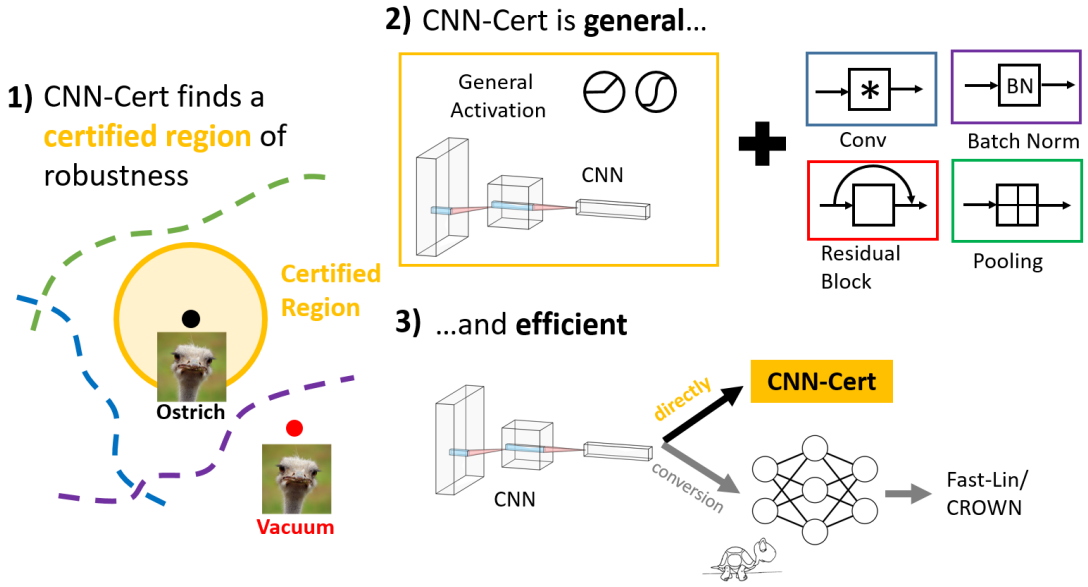
Andere Algorithmen basieren auf der direkten Berechnung von schichtweisen unteren und oberen Robustheitsgrenzen für jedes Neuron unter Verwendung symbolischer linearer Approximationen ähnlich wie bei Neurify (Wang and others 2018a) oder durch Begrenzung der lokalen Lipschitz-Konstante des neuronalen Netzes, wie die **Fast-Lin**- und **Fast-Lip**-Algorithmen, die von Weng et al. (Weng and others 2018) für vollständig verbundene neuronale Netze mit Vorwärtskopplung (*feed-forward fully-connected NN*) und ReLU-Aktivierungen vorgeschlagen wurden. Fast-Lin und Fast-Lip sind über 10.000 Mal schneller als Reluplex bei der Berechnung der minimalen adversen Störung, während die unteren Schranken gegen L_∞-Störungen, die von Fast-Lin und Fast-Lip für ein auf MNIST trainiertes Netzwerk gefunden wurden, nur zwei- bis dreimal größer als die von Reluplex gefundenen, exakten minimalen adversen Störungen, sind.

Eine Verallgemeinerung von Fast-Lin, die mit vorwärts gerichteten, vollständig verbundenen neuronalen Netzen mit beliebigen Aktivierungen (z. B., ReLU-, Sigmoid-, Tanh- und Arctan"=Aktivierungen) arbeiten kann, ist das von Zhang et al. (Zhang and others 2018) vorgeschlagene SW-Tool CROWN13, dessen experimentelle Bewertung mit den Trainingsdaten MNIST (Le Cun, Cortes, and Burges 1998) und CIFAR-10

[13] Der Python-Sourcecode mit Dokumentation liegt auf Crown.

(Krizhevsky 2009)) gezeigt hat, dass es kleinere untere Schranken als die von Fast-Lin ermittelten finden kann auf Kosten einer bescheidenen Zunahme der Rechenzeit.

CNN-Cert[14] ist eine CROWN-Erweiterung, die die Untersuchung von faltenden NN ermöglicht (Boopathy and others 2019), vergleiche Abbildung Prinzip der Verifikation eines NN.



Prinzip der Verifikation eines NN mit CNN-Cert.

Schließlich unterstützt **POPQORN**[15] (**P**ropagated-**o**ut**p**ut **Q**uantified R**o**bustness for **RN**Ns) rekurrente neuronale Netze, LSTM und Gated Architekturen. Die Autoren Ching-Yun Ko et al. (Ko and others 2019) konzentrieren sich auf die Berechnung einer garantierten oder zertifizierten nicht-trivialen unteren Schranke der minimalen adversen Störung unter Verwendung einer modifizierten Punktezahl, genannt *CLEVER-RNN*, um sequenzielle Eingaben zu berücksichtigen. In ihren Experimenten entspricht die Perturbation der Bildstörung eines Einzelbildes der Datensequenz.

Für die Verifizierung des Verhaltens neuronaler Netzwerke sind viele Aspekte zu berücksichtigen und ein Sensitivitätskriterium wie die oben diskutierten unteren Schranken ist nur eines von vielen. Deep Reinforcement Learning benötigt zum Beispiel einen Formalismus zur Codierung von Entscheidungsfindungsaufgaben in wechselnden Umgebungen mit variierenden Einzelbild-Qualitäten. In diesem Fall ist es die Sicherheit der Agenten-Policy, die bewertet werden sollte, was eine Reihe von Eigenschaften und deren mögliche Verletzungsrate umfasst. Es ist nicht möglich abzuleiten, welche Ausgabe-Aktion vom Agenten gewählt wird, wenn bei der Verifikation nur Schranken für die Ausgaben berechnet werden, ohne die Form (und die Beziehung) der Ausgangsfunktionen insgesamt zu berücksichtigen. **ProVe**[16] verifiziert die Beziehung zwischen zwei oder mehr Ausgaben unter Verwendung der Intervallalgebra von Moore und berechnet die Propagation für eine Teilmenge des Eingabebereichs, um eine genaue Schätzung der Ausgabeform zu erhalten (Corsi, Marchesini, and Farinelli 2021).

ProVe wurde mit dem ACAS Xu Benchmark, der Bahnkurvengenerierung für einen Roboter-Manipulator und kartenloser Navigation getestet. Es konnte gezeigt werden, dass die 15 Eigenschaften von ACAS, die als Standardmetrik zur Bewertung der Sicherheit in neueren Arbeiten verwendet werden (Wang and others 2018a; Katz and others 2017), nicht informativ genug sind, um die Kollisionsvermeidung zu garantieren.

[14] Sourcecode in Python und eine ausführliche Anleitung werden auf CNN-Cert bereit gestellt.
[15] Die Veröffentlichung begleitender Sourcecode, implementiert in Python, ist unter POPQORN verfügbar, ebenso wie Dokumentation zu einigen Experimenten mit POPQORN.
[16] Eine Python-Implementierung steht auf https://github.com/d-corsi/ProVe nebst Dokumentation zur Verfügung.

Anstatt den kompletten Satz unsicherer möglicher Aktionen zu verifizieren, haben Corsi et al. (Corsi, Marchesini, and Farinelli 2021) ProVe angewandt, um die Verletzungsrate bei nur zwei grundlegenden, wie sie es nennen, Entscheidungseigenschaften, die zur maximalen Belohnung für den Agenten beitragen, zu ermitteln. Es konnte eine starke Korrelation zwischen dieser Verletzungsrate als neuer Metrik und der Kollisionsabschätzung nachgewiesen werden. Die NN-Sicherheit konnte ohne viele Informationen über die Umgebung ausgewertet werden. ProVe erwies sich als über 20x schneller als Neurify.

Dieser Abschnitt hat einen Überblick der formalen Methoden für künstliche neuronale Netzwerke gegeben, kann aber bei Weitem nicht das Thema erschöpfen. Zumal die formale Verifikation von neuronalen Netzen ein Feld intensiver aktueller Forschung ist und sich laufend weiter entwickelt.

Literaturverzeichnis

Bak Stanley. 2021. “nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement.” In NASA Formal Methods Symposium, 19–36. doi:10.1007/978-3-030-76384-8_2.

Boopathy Akhilan, and others. 2019. “CNN-Cert: An Efficient Framework for Certifying Robustness of Convolutional Neural Networks.” In Proceedings of the AAAI Conference on Artificial Intelligence, doi:10.1609/aaai.v33i01.33013240.

Censi Andrea, Konstantin Slutsky, Tichakorn Wongpiromsarn, Dmitry Yershov, Scott Pendleton, James Fu and Emilio Frazzoli. 2019. “Liability, Ethics, and Culture-Aware Behavior Specification using Rulebooks.” In 2019 International Conference on Robotics and Automation (ICRA), pages 8536–8542. doi:10.1109/ICRA.2019.8794364.

Corsi Davide, Enrico Marchesini and Alessandro Farinelli. 2021. “Formal Verification of Neural Networks for Safety-Critical Tasks in Deep Reinforcement Learning.” In Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence, 161:333–343. PMLR. https://proceedings.mlr.press/v161/corsi21a.html.

De Lemos, Rogério. 2005. “The conflict between self-* capabilities and predictability.” In Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3460 LNCS:219–228. Springer Verlag. doi:10.1007/11428589_15.

Desai Ankush, Tommaso Dreossi and Sanjit A Seshia. 2017. “Combining Model Checking and Runtime Verification for Safe Robotics.” In Lahiri, S. and Reger, G., editors, 17th International Conference on Runtime Verification (RV), volume 10548 of Runtime Verification, pages 172–189. Springer International Publishing. doi:10.1007/978-3-319-67531-2_11.

Doller Andreas and Jan Woelken. 2014. “Produktionsplanung mit SAP in der Prozessindustrie.” SAP PRESS. ISBN 978-3-8362-2892-3.

Dreossi Tommaso, Thao Dang, Alexandre Donzè, James Kapinski, Xiaoqing Jin and Jyotirmoy V. Deshmukh. 2015. “Efficient guiding strategies for testing of temporal properties of hybrid systems.” In Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9058:127–42. Springer Verlag. doi:10.1007/978-3-319-17524-9_10.

Dreossi Tommaso, Alexandre Donzé, Sanjit A Seshia. 2018. “Compositional Falsification of Cyber-Physical Systems with Machine Learning Components.” In Barrett, C., Davies, M., and Kahsai, T., editors, NASA Formal Methods, pages 357–372. arXiv:1703.00978.

Dreossi Tommaso, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte and Sanjit A Seshia. 2019. “VERIFAI: A Toolkit for the Design and Analysis of Artificial Intelligence-Based Systems.” doi:10.1007/978-3-030-25540-4_25.

Dreossi Tommaso, Shromona Ghosh, Alberto Sangiovanni-Vincentelli and Sanjit A Seshia. 2019. “A Formalization of Robustness for Deep Neural Networks.” In Proceedings of the AAAI Spring Symposium Workshop on Verification of Neural Networks (VNN). http://people.eecs.berkeley.edu/~sseshia/pubs/b2hd-dreossi-vnn19.html.

Ehlers Rüdiger. 2017. “Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks.” In Automated Technology for Verification and Analysis. ATVA 2017, 269–86. doi:10.1007/978-3-319-68167-2_19.

Fedorov Stanislav, and Antonio Candelieri. 2018. “Reachability-based safe learning for optimal control problem.” arXiv:1811.04006.

Fremont Daniel J., Alexandre Donzé and Sanjit A. Seshia. 2017. "Control Improvisation." arXiv:1704.06319.

Fremont Daniel J., Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli and Sanjit A. Seshia. 2019. "Scenic: A Language for Scenario Specification and Scene Gen-eration." In PLDI 2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 63–78. doi:10.1145/3314221.3314633.

Friston Karl. 2005. "A theory of cortical responses." Philosophical Transactions of the Royal Society B: Biological Sciences. doi:10.1098/rstb.2005.1622.

Friston Karl, James Kilner and Lee Harrison. 2006. "A free energy principle for the brain." Journal of Physiology Paris 100 (1-3): 70–87. doi:10.1016/j.jphysparis.2006.10.001.

"Funktionale Sicherheit – Wikipedia." 2020. https://de.wikipedia.org/wiki/Funktionale_Sicherheit. [Online; Stand 11. September 2021].

Garavel Hubert, and Susanne Graf. 2013. "Formal Methods for Safe and Secure Computer Systems." BSI Study 875, 362. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/formal_methods_study_875/formal_methods_study_875.html

García Javier and Fernando Fernández. 2015. "A Comprehensive Survey on Safe Reinforcement Learning." Journal of Machine Learning Research, 16(42):1437–1480. https://jmlr.org/papers/v16/garcia15a.html.

Gehr Timon, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri and Martin Vechev. 2018. "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation." In Proceedings - IEEE Symposium on Security and Privacy, 2018-May:3–18. Institute of Electrical; Electronics Engineers Inc. doi:10.1109/SP.2018.00058.

Ghorbal Khalil, Eric Goubault and Sylvie Putot. 2009. "The Zonotope Abstract Domain Taylor1+." In Computer Aided Verification, 627–33. doi:10.1007/978-3-642-02658-4_47.

Goodfellow Ian J, Jonathon Shlens and Christian Szegedy. 2015. "Explaining and harnessing adversarial examples." In 3rd International Conference on Learning Representations, Iclr 2015 - Conference Track Proceedings. https://github.com/lisa-lab/pylearn2/tree/master/pylearn2/scripts/.

Goodfellow Ian, Patrick McDaniel and Nicolas Papernot. 2018. "Making machine learning robust against adversarial inputs: Such inputs distort how machine-learningbased systems are able to function in the world as it is." Communications of the ACM 61 (7): 56–66. doi:10.1145/3134599.

Gordon Andrew D., Thomas A. Henzinger, Aditya V. Nori and Sriram K. Rajamani. 2014. "Probabilistic programming." In Future of Software Engineering, Fose 2014 - Proceedings, 167–81. doi:10.1145/2593882.2593900.

Jansen Nils, Bettina Könighofer, Sebastian Junges, Alex Serban and Roderick Bloem. 2020. "Safe Reinforcement Learning Using Probabilistic Shields." In Konnov, I. and Kovács, L., editors, 31st International Conference on Concurrency Theory (CONCUR 2020), volume 171 of Leibniz International Proceedings in Informatics (LIPIcs), pages 3:1–3:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CONCUR.2020.3.

Jha Susmit, and Sanjit A Seshia. 2016. "A Theory of Formal Synthesis via Inductive Learning." Acta Informatica, 54(7):693–726. doi:10.1007/s00236-017-0294-5.

Julian Kyle and others. 2016. "Policy Compression for Aircraft Collision Avoidance Systems." In 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), 1–10. doi:10.1109/DASC.2016.7778091.

Katz Guy and others. 2017. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks." In Computer Aided Verification – 29th International Conference, CAV 2017, 269–286. doi:10.1007/978-3-319-63387-9_5.

Katz Guy and others. 2019. "The Marabou Framework for Verification and Analysis of Deep Neural Networks." In Computer Aided Verification – 31st International Conference, CAV 2019, 443–52. doi:10.1007/978-3-030-25540-4_26.

Ko Ching-Yun and others. 2019. "POPQORN: Quantifying Robustness of Recurrent Neural Networks." In Proceedings of the 36th International Conference on Machine Learning, 97:3468–3477. PMLR. https://proceedings.mlr.press/v97/ko19a.html.

Krizhevsky Alex. 2009. "Learning Multiple Layers of Features from Tiny Images." Tech report, Computer Science. University of Toronto, Canada. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Kwiatkowska Marta, Gethin Norman and David Parker. 2011. "PRISM 4.0: Verification of probabilistic real-time systems." In Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6806 LNCS:585–91. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-22110-1_47.

Le Cun Yann, Corinna Cortes and Christopher J. C. Burges. 1998. "The MNIST Database of Handwritten Digits." http://yann.lecun.com/exdb/mnist/.

Le Cun, Yann and others. 1989. "Handwritten Digit Recognition: Applications of Neural Network Chips and Automatic Learning." IEEE Communications Magazine 27 (11): 41–46. doi:10.1109/35.41400.

Lo David, Siau-Cheng Khoo, Jiawei Han, Chao Liu. 2011. "Mining Software Specifications: Methodologies and Applications ". doi:10.1201/b10928.

Meel Kuldeep S., Moshe Y. Vardi, Supratik Chakraborty, Daniel J. Fremont, Sanjit A. Seshia, Dror Fried, Alexander Ivrii and Sharad Malik. 2015. "Constrained Sampling and Counting: Universal Hashing Meets SAT Solving *." In Proceedings of Workshop on Beyond NP(BNP). https://www.comp.nus.edu.sg/~meel/publications.html

Mitchell Tom M., Richard M. Keller and Smadar T. Kedar-Cabelli. 1986. "Explanation-Based Generalization: A Unifying View." Machine Learning 1 (1): 47–80. doi:10.1023/A:1022691120807.

Mohammadi Mehdi, Ala Al-Fuqaha, Sameh Sorour and Mohsen Guizani. 2018. "Deep Learning for IoT Big Data and Streaming Analytics: A Survey." IEEE Communications Surveys & Tutorials, 20(4):2923–2960. doi:10.1109/COMST.2018.2844341.

Müller Christoph and others. 2020. "Neural Network Robustness Verification on GPUs." Computing Research Repository (CoRR). arXiv:2007.10868.

Narodytska Nina and others. 2018. "Verifying Properties of Binarized Deep Neural Networks." In Proceedings of the AAAI Conference on Artificial Intelligence, 32:6615–6624. https://ojs.aaai.org/index.php/AAAI/article/view/12206.

Ng Andrew Y. and Stuart Russell. 2000. "Algorithms for Inverse Reinforcement Learning." In ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning, pages 663–670. doi:10.1.1.41.7513.

Nuzzo Pierluigi, John B. Finn, Antonio Iannopollo and Alberto L. Sangiovanni-Vincentelli. 2014. "Contract-based design of control protocols for safety-critical cyber-physical systems." In 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 1–4.EDAA. doi:10.7873/date.2014.072.

Pasareanu Corina S, Divya Gopinath, Huafeng Yu and Nasa Ames. 2018. "Compositional Verification for Autonomous Systems with Deep Learning Components White Paper." In Yu, H., Li, X., Murray, R. M., Ramesh, S., and Tomlin, C. J., editors, Safe, Autonomous and Intelligent Vehicles, chapter 10, pages 187–197. Springer International Publishing. doi:10.1007/978-3-319-97301-2_10.

Pearl Judea. 2018. "The Seven Tools of Causal Inference with Reflections on Machine Learning." Communications of the ACM, 62:54–60. doi:10.1145/3241036.

Ponakala Rajasekhar. 2019. "Testing Deep Neural Networks for Classification Tasks Through Adversarial Perturbations on Test Datasets." MSc, Jawaharlal Nehru Technological University Hyderabad, Telagana, India. Thesis Commons https://thesiscommons.org/r7wcn/download.

Pulina Luca and Armando Tacchella. 2010. "An Abstraction-Refinement Approach to Verification of Artificial Neural Networks." In Computer Aided Verification. CAV 2010, 243–257. doi:10.1007/978-3-642-14295-6_24.

Pulina Luca and Armando Tacchella. 2012. "Challenging SMT solvers to verify neural networks." AI Communications 25 (2): 117–135. doi:10.3233/AIC-2012-0525.

Roehm Hendrik, Jens Oehlerking, Thomas Heinz and Matthias Althoff. 2016. "STL Model Checking of Continuous and Hybrid Systems." In Artho, C., Legay, A., and Peled, D., editors, Automated Technology for Verification and Analysis, pages 412–427. doi:10.1007/978-3-319-46520-3_26.

Sadigh D., K. R. Driggs-Campbell, A. Puggelli, W. Li, V. Shia, R. Bajcsy, A. L. Sangiovanni-Vincentelli, S. S. Sastry and S. A. Seshi. 2014. „Data-driven probabilistic modeling and verification of human driver behavior." In 2014 AAAI Spring Symposium, Stanford University, volume SS-14-02, pages 56–61. https://escholarship.org/uc/item/3rc2d5f6.

Scheibler Karsten and others. 2015. "Towards Verification of Artificial Neural Networks." In Proceedings of the 18th Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV), 30–40. https://ira.informatik.uni-freiburg.de/~wimmer/pubs/scheibler-et-al-mbmv-2015.pdf.

Seshia Sanjit A. 2017. "Compositional Verification without Compositional Specification for Learning-Based Systems." Technical report, EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-164.html.

Seshia Sanjit A. 2019. "Introspective Environment Modeling." In Finkbeiner, B. and Mariani, L., editors, Runtime Verification, pages 15–26. doi:10.1007/978-3-030-32079-9_2.

Seshia Sanjit A., Dorsa Sadigh and S. Shankar Sastry. 2020. "Towards Verified Artificial Intelligence." arXiv:1606.08514.

Sha, Lui. 2001. "Using Simplicity to Control Complexity." IEEE Software, 18(4):20–28. doi:10.1109/MS.2001.936213.

Singh Gagandeep and others. 2018. "Fast and Effective Robustness Certification." In Proceedings of the 32nd International Conference on Neural Information Processing Systems of Adv. Neural Inf., 31:10825–10836. https://proceedings.neurips.cc/paper/2018/hash/f2f446980d8e971ef3da97af089481c3-Abstract.html.

Singh, G. and others 2019a. "An Abstract Domain for Certifying Neural Networks." In Proceedings of the Acm on Programming Languages (Popl), 3:1–30. doi:10.1145/3290354.

Singh, G and others 2019b. "Beyond the Single Neuron Convex Barrier for Neural Network Certification." In Proceedings of the 33rd International Conference on Neural Information Processing Systems, 32:15072–15083. Adv. Neural Inf. Process. Syst. https://proceedings.neurips.cc/paper/2019/hash/0a9fdbb17feb6ccb7ec405cfb85222c4-Abstract.html.

Singh, G. and others 2019c. "Boosting Robustness Certification of Neural Networks." In International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=HJgeEh09KQ.

Smith David J. and Kenneth G.L. Simpson. 2020. The Safety Critical Systems Handbook. Elsevier. doi:10.1016/c2019-0-00966-1.

Szegedy Christian and others. 2014. "Intriguing Properties of Neural Networks." In International Conference on Learning Representations, ICLR 2014. arXiv:1312.6199.

Tjeng Vincent, Kai Y. Xiao and Russ Tedrake. 2019. "Evaluating Robustness of Neural Networks with Mixed Integer Programming." In International Conference on Learning Representations. https://openreview.net/forum?id=HyGIdiRqtm.

Tran Hoang-Dung and others. 2020. "Verification of Deep Convolutional Neural Networks Using ImageStars." In Computer Aided Verification. CAV 2020, 18–42. doi:10.1007/978-3-030-53288-8_2.

Van De Meent Jan-Willem, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. "An Introduction to Probabilistic Programming." arXiv:1809.10756.

Vassev Emil. 2016. "Safe artificial intelligence and formal methods." Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9952 LNCS: 704–13. doi:10.1007/978-3-319-47166-2_49.

Vinge V. 1993. "The coming technological singularity: How to survive in the post-human era." Talk delivered at the VISION-21 Symposium, March 30–31, 1993.

Wahlster Wolfgang. 2017. "Künstliche Intelligenz als Grundlage autonomer Systeme." Informatik-Spektrum 40 (5): 409–418. doi:10.1007/s00287-017-1049-y.

Wang Shiqi and others. 2018a. "Efficient Formal Safety Analysis of Neural Networks." In Proceedings of the 32nd International Conference on Neural Information Processing Systems, 31:6367–6377. Adv. Neural Inf. Process. Syst. https://proceedings.neurips.cc/paper/2018/hash/2ecd2bd94734e5dd392d8678bc64cdab-Abstract.html

Wang Shiqi and others. 2018b. "Formal Security Analysis of Neural Networks using Symbolic Intervals." In 27th USENIX Security Symposium (USENIX Security 18), 1599–1614. https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi.

Weng Lily and others. 2018. "Towards Fast Computation of Certified Robustness for ReLU Networks." In Proceedings of the 35th International Conference on Machine Learning, 80:5276–5285. PMLR. https://proceedings.mlr.press/v80/weng18a.html.

Zhang Huan and others. 2018. "Efficient Neural Network Robustness Certification with General Activation Functions." In Proceedings of the 32nd International Conference on Neural Information Processing Systems, 31:4944–4953. Adv. Neural Inf. Process. Syst. https://proceedings.neurips.cc/paper/2018/hash/d04863f100d59b3eb688a11f95b0ae60-Abstract.html.