Wie sicher ist KI?

Grundlagen und Verifikation von Künstlicher Intelligenz

Klaus Mainzer[1]

Abstract: In Zeitalter der Digitalisierung nimmt die Künstliche Intelligenz eine Schlüsselstellung ein. Was ist aber Künstliche Intelligenz? Was kann sie heute und was kann sie nicht? Im Unterschied zu den logischen Formalismen der klassischen symbolischen KI wird das aktuelle Machine Learning durch statistisches Lernen dominiert, das die Bewältigung großer Datenmengen mit leistungsstarken Rechnern in Technik und Wirtschaft verspricht. Wie sicher sind aber statistische Korrelationen? Was sind und was können demgegenüber Kausalmodelle leisten? Kausalanalysen sind erkenntnistheoretisch und ethisch mit Verantwortungsfragen eng verbunden. Nur wenn diese Grundlagen der KI-Technologie klar verstanden sind, lassen sich auch ihre Anwendungen beurteilen und ethisch-rechtlich bewerten. Daher plädiert dieser Beitrag für eine Kombination von Machine Learning mit kausalem Lernen und zertifizierten KI-Programmen durch Beweisassistenten.

1 Formale Verifikationsprogramme der symbolischen KI

In ihren Anfängen wurde Künstliche Intelligenz als Automatisierung logischen Denkens und Beweisens verstanden. Man spricht in diesem Fall auch von symbolischer KI [KM20].

1.1 Von Logik und Beweistheorie zur SAT-Verifikation

David Hilbert hatte in den 1920er Jahren einen Formalismus entwickelt, um mathematische Konzepte in der formalen Sprache der Prädikatenlogik zu formulieren. Damit stand ein formales Verfahren zur Verfügung, um die Allgemeingültigkeit (bzw. Widersprüchlichkeit) einer Formel mechanisch nachzuweisen. Eine Formel kann nur dann widersprüchlich sein, wenn es ein endliches Anwendungsbeispiel gibt, das den Widerspruch aufzeigt. Außerdem gibt es zu jeder Formel ein „allgemeinstes“ (i.A. unendlich großes) Anwendungsbeispiel.

Dieses nach dem Logiker Jacques Herbrand benannte Beispiel besteht aus aussagenlogischen Formeln, die man systematisch und sukzessiv aus den in der Formel vorkommenden Symbolen konstruieren kann. Diese Formeln sind generische Namen für Individuen, die in einem beliebigen Anwendungsbeispiel nach Maßgabe der prädikatenlogischen Formel

[1] TU München und Carl Friedrich von Weizsäcker-Zentrum, Universität Tübingen, Deutschland, k.mainzer@outlook.com

doi:10.18420/inf2020_63

notwendigerweise vorkommen müssen. Jeder endlich große Anfang des Beispiels stellt nun eine Konjunktion von aussagenlogischen Formeln dar, deren Erfüllbarkeit (oder Widersprüchlichkeit) man mit rein aussagenlogischen Mitteln in endlicher Zeit entscheiden kann (z.B. mit Wahrheitstabellen).

Mit der Verfügbarkeit von Computern in den 1950er Jahren begannen mehrere Forscher in den USA das genannte Beweisverfahren konkret auszuformen und auf verschiedene Arten zu implementieren. Das Kernproblem war dabei, zunächst ein effizientes Verfahren für die aussagenlogische Erfüllbarkeit (SAT-Solving) zu finden, die wiederholt für endliche Anfänge des kombinatorisch stark anwachsenden Herbrand Beispiels nachzuweisen war. Der Philosoph Willard V. O. Quine hatte dazu Wahrheitstabellen vorgeschlagen, die aber exponentielles Wachstum aufweisen [Qu55]. Paul Gilmore hatte stattdessen eine disjunktive Form (der Art $A \wedge B \wedge C \cdots \vee E \wedge F \wedge G \cdots \vee \ldots$) benutzt und auf einer IBM 704 implementiert, aber auch dieser Ansatz war schon auf kleinen Beispielen ineffizient [Gi60]. IBM produzierte von 1954 bis 1960 das Modell 704 mit Schaltkreisen aus Röhren, etwa 18KB bis 144KB Hauptspeicher aus Magnetkernen sowie Magnetbandspeicher von je 5MB. Martin Davis und Hilary Putnam fassten 1960 die Gesamtsituation zusammen und präsentierten zudem ein völlig neues Verfahren für die Erfüllbarkeitsprüfung, das im Kern auf der Idee der Resolution aufbaut [DP60]. Sie konnten damit von Hand ein Beispiel von Gilmore rechnen, bei dem dessen Programm abbrechen musste.

Danach implementierten Davis, Logemann und Loveland das neue Verfahren ebenfalls auf einer IBM 704 und entdeckten sofort, dass es an einer entscheidenden Stelle verändert werden musste, um nicht ebenfalls ein explosives Wachstum der Formeln auszulösen. In ihrer 1962 publizierten Arbeit wird der heute als „DPLL“ bekannte rekursive Algorithmus beschrieben, der erstmalig ein praktikables Verfahren zum SAT-Solving darstellt und bis in die 1990er Jahre hinein Bestand hatte [Da+62]. Das Beispiel von Gilmore wurde damit in nur 2 Minuten automatisch gelöst.

Im Jahr 1965 publizierte J. A. Robinson eine neue Methode zum Beweisen in Prädikatenlogik. Dieses „Resolution“ genannte Verfahren erlaubt es, direkt aus der Menge der prädikatenlogischen Klauseln neue logisch implizierte Klauseln abzuleiten, ohne wie zuvor einen Umweg über die Aussagenlogik zu nehmen. Falls die ursprüngliche Formel (bzw. deren Klauselmenge) widersprüchlich ist, kann man immer in endlicher Zeit die „leere“ Klausel ableiten, die nicht erfüllt werden kann und so den Widerspruch zu Tage treten lässt. Die aussagenlogische Variante wurde bereits erwähnt. Prädikatenlogisch ist die Sache komplizierter, denn die beiden Klauseln $\{P(x)\}$ und $\{\neg P(f(y))\}$ widersprechen sich auf den ersten Blick noch nicht. Allerdings sind x und y implizit als allquantifiziert zu verstehen, sodass aus $\forall x.P(x)$ auch $\forall x.P(f(x))$ folgt und damit ein Widerspruch zu $\neg P(f(y))$.

Zum Vergleich: in der „alten“ Methode aus den 1950er Jahren hätte man den Widerspruch durch sukzessives und im Prinzip zielloses Aufbauen des Herbrand-Beispiels wie folgt ermitteln müssen: Soll es ein nicht-leeres Beispiel geben, so muss mindestens ein Individuum „a“ vorhanden sein und es muss eine Funktion „f“ geben. Damit muss es auch ein „$f(a)$“

geben, und weiter „$f(f(a))$“, „$f(f(f(a)))$“, usf. Wir prüfen nun, ob durch Einsetzen von a (in x und y) alleine ein Widerspruch zu Tage tritt; das ist nicht der Fall. Auch durch Einsetzen von a für x und $f(a)$ für y ergibt sich kein Widerspruch. Erst durch Einsetzen von $f(a)$ für x und a für y tritt der Widerspruch zu Tage. Durch geschicktes Einsetzen kommt man also (viel) schneller zum Ziel als durch ungeschicktes Einsetzen, und nach jedem Einsetzen braucht man eine erneute Erfüllbarkeitsprüfung. Demgegenüber vergleicht Robinson die prädikatenlogischen Formeln systematisch und stellt die Gleichheit durch gezieltes Einsetzen („Unifikation“) her. Im Beispiel sehen wir, dass wir im alten Verfahren für x ein Individuum „$f(a)$“ einsetzen müssen, wenn wir für y „a“ einsetzen, oder eben für x ein „$f(f(a))$“ und für y dann ein „$f(a)$“ etc.

Prädikatenlogische Resolution ist deutlich effizienter als das vorhergehende Verfahren; die Gilmore Formel wird nach nur 5 einfachen Resolutionsschritten (Ableitung neuer Klauseln) als widersprüchlich erkannt, was auch von Hand problemlos zu bewältigen ist. Im Gegenzug waren die ersten 24 Einsetzungen von Davis & Putnam jeweils erfüllbar, und erst die 25. Einsetzung erzeugte einen Widerspruch.

Aufgrund der deutlich besseren Effizienz wendete sich die damalige KI sofort der Resolutionsmethode als neuem Werkzeug für das zentrale Problem des „Reasoning“ (Herstellen von Schlussfolgerungen) zu. Ein unabhängig agierender intelligenter Akteur oder Agent muss seine Umwelt wahrnehmen können, daraus Schlussfolgerungen für sein intendiertes Handeln ableiten und diese sodann in Aktionen übersetzen. Hieraus ergaben sich Teilgebiete der KI wie Bilderkennung („Computer Vision“) und Sprachverstehen einerseits und Robotik andererseits sowie als zentrale Komponente Wissensrepräsentation mit intelligentem Schlussfolgern und Planen. Nach der herkömmlichen wissenschaftlichen Methode modelliert man die Umwelt mathematisch um durch Berechnungen Vorhersagen treffen können. Da die Mathematik nach Hilbert prädikatenlogisch gefasst werden kann, liegt es nahe, eine prädikatenlogische Repräsentation und einen automatischen Beweiser als zentrales Werkzeug vorzusehen.

Ein wichtiger Schritt war das Lehrbuch „Problem solving methods in AI“ von Nils Nilsson [Ni71, Ni80]. Neben verschiedenen Algorithmen um Gewinnstellungen in Lösungsräumen z.B. von Brettspielen zu suchen, widmen sich 3 von 8 Kapiteln dem Automatischen Beweisen mit der Resolutionsmethode und ihren Anwendungsmöglichkeiten. Hier geht es nun nicht mehr nur um den effizienten Beweis eines logischen Widerspruchs, sondern vor allem auch um die Extraktion von konkreten Handlungsanweisungen und Antworten aus dem Widerspruchsbeweis. Ein Beispiel ist die Extraktion eines Plans zum Erfüllen eines Ziels bis hin zum automatischen Schreiben eines Programms.

Eine eingeschränkte Form von Klauseln (sog. Horn-Klauseln) wurden zur Grundlage der neuen Programmiersprache PROLOG. Grundidee ist hier, sowohl Daten als auch Berechnungsregeln als Menge von Klauseln zu schreiben und einen universellen Beweiser für die konkreten Berechnungen zu nutzen. Dies sollte transparente, leicht wartbare Programme ermöglichen ohne die Notwendigkeit komplexer Schleifenkonstrukte und

Verzweigungen. Einer der ersten Prolog Compiler wurde um 1978 am „Department for Artificial Intelligence“ der Universität Edinburgh fertiggestellt [CM81].

In Deutschland begann eine kleine Gruppe um Jörg Siekmann mit der Programmierung eines Resolutionsbeweisers. Ein Ziel sollte es sein, die Beweise in einem Lehrbuch der Berechnungstheorie automatisch nachzuvollziehen. Diese Gruppe organisierte 1983 die führende Tagung „International Joint Conference of AI (IJCAI-83)“ in Karlsruhe. Themen waren System Support, Theorem Proving, Cognitive Modelling, Automatic Programming, Planning and Search, Knowledge Representation, Learning and Knowledge Acquisition, Logic Programming, Natural Language, Expert Systems, Vision, Robotics.

Aber in Deutschland war die KI nicht unumstritten. Einen eigenen Fachbereich gab es dafür anders als in Edinburgh nicht. Erst 1988 wurde das Deutsche Forschungszentrum KI (DFKI) gegründet. Ein führendes Lehrbuch der Informatik sprach noch von der „sogenannten künstlichen Intelligenz“. Allerdings erwiesen sich viele kühne Ziele der KI auch als schwieriger als zunächst vermutet. Nicht nur erwies sich Intelligenz als ein unerwartet komplexes Phänomen. Bereits mathematische Logik und ihre Beweise sind nicht einfach zu bewältigen, besonders wenn es effizient und in realisierbarer Zeit geschehen soll. Bereits 1962 hatten Davis, Logemann und Loveland resümiert: „We hoped that some mathematically meaningful and, perhaps nontrivial, theorems could be solved. The actual achievements in this direction were somewhat disappointing“.

Bis in die 1990er Jahre trat das SAT-Solving der Aussagenlogik in den Hintergrund, da es für Beweise in der Prädikatenlogik nicht mehr gebraucht wurde. Dann aber wurden mikroelektronische Schaltungen immer komplexer, deren Schaltpläne im Wesentlichen in Boole'scher Algebra (Schaltalgebra) vorliegen. Ein weit beachtetes Ereignis war 1994 ein Fehler in der Gleitkommadivision des Pentium 5 Prozessors von Intel. Im Zuge des neu erwachten Interesses an der Verifikation kombinatorischer Schaltkreise gelang Joao Markes-Silva [Ma95] und Karem Sakallah 1996 ein richtungsweisender Durchbruch beim SAT-Solving: die intelligente Kombination der DPLL Suchprozedur mit dem Deduktionsverfahren Resolution [MS96, MS99].

Deduktionsverfahren häufen ständig neues Wissen an, indem sie dynamisch neue Klauseln erzeugen und leiden in der Folge unter diesem Speicherverbrauch. Reines DPLL-Suchen hat keinen dynamischen Speicherverbrauch, kann aber unter langen Laufzeiten leiden, wenn es durch backtracking wiederholt auf dieselbe Weise bei der Lösung derselben Teilmenge von Formeln scheitert. Das conflict driven clause learning (CDCL) besteht nun darin, dass man aus den Konflikten lernt, auf die man bei der DPLL Suche stößt. Ein Konflikt besteht darin, dass eine der Klauseln unter der momentan untersuchten Variablenbelegung falsch (false) wird. Dieser Fall tritt niemals direkt durch eine *true/false* Entscheidung auf, sondern immer durch weitere Wertepropagationen aufgrund einer Entscheidung.

Solche Propagationen werden immer durch Klauseln erzwungen: z.B. erzwingt eine Klausel $\{x, y\}$ die Propagation von $y = true$ sobald $x = false$ gesetzt wird. Falls eine Klausel false

wird, so liegt das daran, dass sie die umgekehrte Propagation erzwingen würde, im Beispiel etwa $\{x, \neg y\}$. In dieser Situation gibt es aber immer eine Resolvente, im Beispiel wäre das $\{x\}$. Als Resolvente ist die Klausel eine logische Konsequenz der zu lösenden Formel und kann als zusätzliche Klausel „gelernt" werden. In unserem Beispiel würde sie verhindern, dass nach backtracking jemals wieder eine Lösung mit x=false versucht wird, denn $\{x\}$ erzwingt zuvor sofort $x = \mathit{true}$.

Eine in der Praxis wichtige Konsequenz des CDCL Solving besteht darin, dass der Solver im Fall einer unlösbaren Formel nun die leere Klausel „lernt" und dazu einen Resolutionsbeweis konstruiert. Nun kann das bislang lapidare Ergebnis „UNSAT" durch einen nachvollziehbaren (und unabhängig nachprüfbaren) Beweis begründet werden. In Fällen wo „UNSAT" auf einen Fehler in der Anwendung (z.B. einem Schaltkreis) zeigt, kann aus dem Beweis oft eine Anleitung zur konkreten Lokalisation des Fehlers und zu seiner Korrektur entnommen werden.

Neben dem Lernen wurden für moderne CDCL Solver weitere wichtige Komponenten entwickelt, wie z.B. hoch effiziente Implementierungen der Wertepropagation und gute Heuristiken zur Auswahl der nächsten zu belegenden Variable. Außerdem entstanden um das CDCL-Solving weitere verwandte Algorithmen wie die Berechnung von Primimplikanten (wichtig für die Minimierung von Formeln) oder zur Berechnung maximaler lösbarer Teilmengen von unlösbaren Klauselmengen (wichtig für Optimierungszwecke).

Einige wesentliche industrielle Anwendungsgebiete für das SAT Solving sind heute die formale Verifikation von Mikroelektronik und von Software sowie die Lösung von Konfigurationsproblemen. Im Falle von Software übersetzt man ein Programm vollautomatisch in eine Boole´sche Formel. Im Falle von Konfigurationsproblemen (z.B. plugin Konfiguration im Eclipse Programm-Editor) codiert man die Randbedingungen („wenn A dann nur wenn auch B aber nicht C oder D") als Klauseln und lässt den Solver eine gültige Konfiguration berechnen. Auch den Konfiguratoren der Automobilindustrie, die man im Internet benutzen kann, liegen ähnliche Randbedingungen zugrunde. Viele Automobilhersteller benutzen heute SAT-Solver um ihre hochkomplexen Konfigurationsprobleme zu lösen bzw. die Formelsysteme zu analysieren und zu validieren [KS00]. In der Mikrobiologie bestehen Anwendungsmöglichkeiten dadurch, dass sowohl Proteine als auch Gene durch die Kombination von endlich vielen Grundbausteinen gebildet werden, was man durch Boole'sche Variablen codieren kann.

Moderne CDCL Solver funktionieren äußerst effizient und robust, ohne händische Hilfen, und bewältigen Anwendungsprobleme mit (Hundert-)Tausenden von Variablen und Klauseln. Da das Erfüllbarkeitsproblem ein NP-vollständiges Problem ist (es ist das prototypische NP-vollständige Problem) ist kein Algorithmus bekannt, der es auch im verzwicktesten Fall in weniger als exponentieller Zeit (in der Anzahl der Variablen) löst; außerdem ist die Existenz eines solchen Algorithmus sehr unwahrscheinlich. Andererseits können somit auch alle anderen NP-vollständigen Probleme im Kern durch SAT-Solver gelöst werden [Kn16, BG18].

Insgesamt gibt es viele wichtige Anwendungsprobleme, bei denen die Struktur der Klauseln ein praktisch effizientes CDCL Solving erlaubt. Selbst wenn also durch die KI Forschung auf dem Gebiet des Reasoning bis jetzt nicht alle Erwartungen und Hoffnungen erfüllt werden konnten, so hat schon alleine das daraus entstandene SAT-Solving zu vielen konkreten Anwendungen in Industrie und Wissenschaft geführt.

1.2 Von Logik und Beweistheorie zu maschinellen Beweisassistenten

Die moderne Beweistheorie eröffnet die Möglichkeit, interaktive und automatische Beweisassistenten zu entwickeln, die sowohl Beweise in der Mathematik als auch Softwareprogramme in der Informatik überprüfen können. Die zunehmende Komplexität von menschlichem Wissen und menschlicher Technik macht Verifikation zu einem Schlüsselproblem zukünftiger Entwicklungen, insbesondere auf dem Gebiet der Künstlichen Intelligenz. Bemerkenswert ist aber auch, wie tief diese aktuellen Entwicklungsfragen in den Grundlagen von Logik, Mathematik und Philosophie verwurzelt sind.

In der KI realisieren Algorithmen Wissensverarbeitung, indem aus Datenstrukturen (also Zeichenreihen) weitere Zeichenreihen abgleitet werden. Das entspricht dem Ideal des mathematischen Beweisens, das in der Mathematik seit der Antike vertreten wird. Euklid hatte gezeigt, wie aus als wahr vorausgesetzten Axiomen nur durch logische Schlüsse mathematische Lehrsätze abgeleitet und bewiesen werden konnten. In der KI stellt sich die Frage, ob mathematisches Beweisen auf Algorithmen übertragen und damit „automatisiert“ werden kann. Dahinter steht dann die grundlegende KI-Frage, ob und bis zu welchem Grad Denken automatisiert, also durch einen Computer ausgeführt werden kann.

Schauen wir uns dazu einen klassischen Beweis näher an: Um 300 v.Chr. bewies Euklid die Existenz unendlich vieler Primzahlen [AZ01, 3]. Euklid vermeidet den Begriff „unendlich“ und behauptet: „Es gibt mehr Primzahlen als jede vorgelegte Anzahl von Primzahlen“. Der Beweis wird in der Schule mit Widerspruch geführt. Man nimmt das Gegenteil der Behauptung an, schließt logisch unter dieser Annahme auf einen Widerspruch. Die Annahme war also falsch. Wenn wir nun voraussetzen, dass eine Aussage entweder war oder falsch ist, dann gilt das Gegenteil der Annahme: die Behauptung ist richtig.

Der Nachteil an diesem Beweis ist, dass wir die Existenz der Primzahlen konstruktiv nicht bewiesen haben. Wir haben nur gezeigt, dass das Gegenteil der Annahme zu Widersprüchen führt. Um die Existenz eines Objekts zu beweisen, benötigen wir einen Algorithmus, der ein Objekt erzeugt und beweist, dass die Aussage für dieses Beispiel richtig ist. Formal lautet eine Existenzaussage $A \equiv \exists x.B(x)$. In einer abgeschwächten Form könnten wir fordern, eine Liste von endlich vielen Zahlen $t_1, \ldots, t_n$ zu konstruieren, die Kandidaten für die Aussage B sind, so dass die Oder-Aussage $B(t_1) \vee \cdots \vee B(t_n)$ gilt, also die Aussage B für wenigstens eine der konstruierten Zahlen $t_1, \ldots, t_n$ wahr ist. Das könnte auch eine Maschine leisten. Wenn allgemein für alle x ein y mit $B(x, y)$ existieren soll, also formal $A \equiv \forall x \exists y.B(x, y)$ gilt, dann benötigen wir einen Algorithmus p, der für jeden x-Wert einen

Wert $y = p(x)$ konstruiert, so dass $B(x, p(x))$ für alle x gilt, also formal $\forall x B(x, p(x))$. In einer schwächeren Form wären wir zufrieden, wenn für den Suchprozess eines y-Wertes für einen gegeben x-Wert wenigstens eine obere Schranke $b(x)$ berechnet werden könnte, also formal $\forall x \exists y \leq b(x) B(x, y)$. Damit lässt sich der Suchprozess genau abschätzen.

Der amerikanische Logiker G. Kreisel hat deshalb gefordert, dass Beweise mehr als bloße Verifikationen sein sollen. Sie sind gewissermaßen „eingefrorene" Algorithmen. Man muss sie nur in den Beweisen entdecken und „herauswinden" (unwinding proofs) [Fe96]. Dann können sie auch Maschinen übernehmen.

Tatsächlich ist im erwähnten Beweis des Primzahlsatzes ein konstruktives Verfahren „versteckt". Es lässt sich nämlich für jede Position r einer Primzahl p_r (in der Aufzählung $p_1 = 2, p_2 = 3, p_3 = 5, \ldots$) eine obere Schranke $b(r)$ berechnen, also zu jeder vorgelegten Anzahl von Primzahlen eine weitere angeben, die allerdings unterhalb einer berechenbaren Schranke liegt. In Euklids indirektem Beweis werden ja endlich viele Primzahlen angenommen, die kleiner oder gleich einer Schranke x sind, also $p \leq x$. Damit wird dann eine Zahl $1 + \prod_{p \leq x} p$ konstruiert, aus der die Widersprüche abgeleitet werden. (Dabei bezeichnet $\prod_{p \leq x} p$ das Produkt aller Primzahlen, die kleiner als x sind.) Wir konstruieren daher zunächst die Schranke

$$g(x) := 1 + x! \geq 1 + \prod_{p \leq x} p.$$

Die Fakultätsfunktion $1 \cdot 2 \cdot \ldots \cdot x = x!$ lässt sich durch die sogenannte Stirling-Formel abschätzen. Wir zielen allerdings auf eine obere Schranke der $r + 1$-ten Primzahl p_{r+1}, die nur von der Position r in der Aufzählung der Primzahlen anstelle von der unbekannten Schranke $x \geq p_r$ abhängt. Euklids Beweis zeigt $p_{r+1} \leq p_1 \cdot \ldots \cdot p_r + 1$. Daraus lässt sich *für alle* $r \geq 1$ (durch vollständige Induktion über r) beweisen, dass $p_r < 2^{2^r}$. Die gesuchte berechenbare Schranke ist also $b(r) = 2^{2^r}$.

In Logik und Mathematik werden Formeln (also Zeichenreihen) Schritt für Schritt abgeleitet, bis der Beweis einer Behauptung abgeschlossen ist. Computerprogramme arbeiten im Grunde wie Beweise. Schritt für Schritt leiten sie nach festgelegten Regeln Zeichenfolgen ab, bis ein formaler Ausdruck gefunden ist, der für eine Lösung des Problems steht. Stellen wir uns z.B. die Montage eines Werkstücks auf einem Fließband vor. Das entsprechende Computerprogramm beschreibt, wie das Werkstück Schritt für Schritt aus vorausgesetzten Einzelteilen nach Regeln aufeinander aufbauend entsteht.

Ein Kunde wünscht von einem Informatiker ein Computerprogramm, das ein solches Problem löst. Bei einem sehr komplexen und unübersichtlichen Produktionsprozess, möchte er sicher vorher einen Beweis, dass das Programm auch korrekt arbeitet. Eventuelle Fehler wären gefährlich oder würden erhebliche Mehrkosten verursachen. Ein Informatiker beruft sich dazu auf eine Software, die den Beweis automatisch aus den formalen Eigenschaften des Problems extrahiert hat. So wie Software im „Data Mining" zur Suche von Daten oder Datenkorrelationen eingesetzt wird, so lässt sich passende Software auch zur automatischen

Suche von Beweisen einsetzen. Man spricht dann von „proof mining “ [Ko08, Kap. 2]. Das entspricht Georg Kreisels Ansatz, Algorithmen aus Beweisen herauszufiltern (unwinding proofs), nun allerdings automatisch durch Computerprogramme.

Dann entsteht allerdings die Frage, ob die Software zur Extraktion des Beweises selber zuverlässig ist. In einem genau vorgegebenen Rahmen lassen sich solche Zuverlässigkeitsbeweise für die zugrunde gelegte Software führen. Der Kunde kann dann sicher sein, dass das Computerprogramm für seine Problemlösung korrekt arbeitet. Dieses „automatische Beweisen“ hat also nicht nur erhebliche Bedeutung für die moderne Softwaretechnik. Sie führt auch zu philosophisch tiefen Fragen, wieweit nämlich (mathematisches) Denken automatisiert werden kann: Die Beweisfindung ist automatisch. Den Korrektheitsbeweis der dazu verwendeten Software führt aber ein Mathematiker. Selbst wenn wir diesen Beweis wieder automatisieren würden, entsteht eine grundlegende erkenntnistheoretische Frage: Führt uns das nicht in einen Regress, an dessen Ende immer der Mensch steht (stehen muss)?

Ein Beispiel ist das interaktive Beweissystem MINLOG, das aus formalen Beweisen automatisch Computerprogramme heraus extrahiert [Sc06, SW12, Kap. 7]. Es benutzt die Computersprache LISP. Ein einfaches Beispiel ist die Behauptung, dass für jede Liste v von Symbolen in LISP eine Umkehrliste w mit umgekehrter Anordnung der Symbole existiert. Das ist wieder eine Behauptung von der Form $A \equiv \forall v \exists w B(v, w)$. Der Beweis kann informal durch eine Induktion über den Aufbau der Listen v geführt werden. MINLOG extrahiert daraus automatisch ein passendes Computerprogramm. Aber auch für anspruchsvolle mathematische Beweise lässt sich diese Software benutzen. Ein allgemeiner Zuverlässigkeitsbeweis garantiert, dass die Software korrekte Programme liefert.

Um Widersprüche zu vermeiden, wie sie Anfang des 20. Jahrhunderts die Grundlagendiskussion der Mathematik auslösten, sollten Beweise konstruktiv sein. Hilbert wollte mathematische Theorien zunächst formalisieren und für diese Formalismen nachträglich konstruktive Widerspruchsfreiheits- und Vollständigkeitsbeweise führen. Die Mathematiker sollten also weiterhin ungestört ihre Arbeit tun und die Logiker würden sich danach um ihre Korrektheit und Widerspruchsfreiheit kümmern. Wie die Gödelschen Sätze zeigen, lässt sich das Hilbertsche Programm nur bedingt realisieren. Demgegenüber wollte Brouwers Intuitionismus von vornherein nur konstruktive Verfahren in der Mathematik zulassen. Mit Blick auf Sicherungsverfahren spielen intuitionistische Verfahren selbst bei maschinellen Computerprogrammen bis heute eine Rolle.

Die intuitionistische Interpretation der logischen Konstanten geht auf Brouwer, Heyting und Kolmogrov zurück [He34, Ko32, Ko08, 43]. Die Brouwer-Heyting-Kolmogorov (BHK)-Interpretation erklärt die Bedeutung logischer Konstanten mit Termen von Beweiskonstruktionen:

Es gibt keinen Beweis von $\perp$ („Falsch“).

i. Ein Beweis von $(A \wedge B)$ ist ein Paar (q, r) von Beweisen, wobei q ein Beweis von A und r ein Beweis von B ist.

ii. Ein Beweis $(A \vee B)$ von ist ein Paar (n, q), das aus einer ganzen Zahl n und einem Beweis q besteht, der A beweist, falls $n = 0$ und entsprechend B, falls $n \neq 0$.

iii. Ein Beweis p von $(A \rightarrow B)$ ist eine Konstruktion, die einen hypothetischen Beweis q von A in einen $p(q)$ von B transformiert.

iv. Ein Beweis p von $\forall x A(x)$ ist eine Konstruktion, die für jede Konstruktion c_d eines Elements d aus der Domäne des Allquantors einen Beweis $p(c_d)$ von $A(d)$ erzeugt.

v. Ein Beweis von $\exists x A(x)$ ist ein Paar (c_d, d), wobei c_d die Konstruktion eines Elements d aus der Domäne des Existenzquantors und q ein Beweis von $A(d)$ ist.

2 Typentheoretische Verifikationsprogramme in Mathematik und Informatik

Eine wichtige Gemeinsamkeit zwischen Computersprachen und logischen Formalismen ist die Unterscheidung von Typen unterschiedlicher Zeichen, Terme, Formeln und Beweise. B. Russell hatte die Typentheorie vorgeschlagen, um Widersprüche in logischen Formalismen und der Mengenlehre zu vermeiden. In einer Computersprache müssen die Typen der Symbole genau angegeben werden, damit die Maschine keine fehlerhaften Zordnungen durchführt.

2.1 Beweise als Programme mit intuitionistischen Typen

1969 beobachtete der Logiker W.A. Howard (wie vorher bereits H.B. Curry), dass Gentzens Beweissystem natürlichen Schließens in seiner intuitionistischen Version direkt als eine typisierte Variante der Berechnung in Form des Churchschen Lambda-Kalküls interpretiert kann [Ho69]. Man spricht daher auch vom Curry-Howard Isomorphismus. Nach Church bedeutet $\lambda a.b$ eine Funktion, die ein Element a auf den Funktionswert b mit $\lambda a.b[a] = b$ abbildet. Im Folgenden werden Beweise durch Terme $a, b, c, \ldots$, Aussagen durch $A, B, C, \ldots$ repräsentiert [Ma18, Kap. 9].

Beispiele:

$$\begin{array}{rc} & [A] \\ \lambda a.b & \vdots \\ (\rightarrow I) & \dfrac{B}{A \rightarrow B} \end{array}$$

$$\begin{array}{l} \lambda a.(\lambda b.a) \\ (\rightarrow I) \end{array} \begin{array}{c} [A] \\ \vdots \\ \dfrac{B \rightarrow A}{A \rightarrow (B \rightarrow A)} \end{array}$$

Ein Beweis ist ein Program, and die Formel, die er beweist, ist der Typ für dieses Programm. Nach Gentzens Kalkül des natürlichen Schließens lässt sich auch sein Sequenzenkalkül durch den Lambda-Kalkül charakterisieren.

Ein Beweis von $\Gamma \vdash \alpha$ bedeutet, dass man über ein Programm verfügt, das für gegebene Werte mit Typen aufgelistet in Γ ein Objekt vom Typ α herstellt. Ein Axiom korrespondiert zur Einführung einer neuen Variablen mit einem neuen und uneingeschränktem Typ, die $\rightarrow I$ Einführungsregel korrespondiert zur Funktionsabstraktion und die $\rightarrow E$ Eliminationsregel korrespondiert zur Funktionsanwendung.

$t : \alpha$ bedeutet sowohl „t beweist A“ als auch „t ist vom Typ α“.

Auch Aussagen lassen sich als Typen in der intuitionistischen Typentheorie verstehen. Nach dem Curry-Howard Isomorphismus von Aussagen als Typen ist $\Sigma x : A.B$ die disjunkte Summe der A-indizierten Familie von Typen B und $\Pi x : A.B$ ihr cartesisches Produkt [DP16].

Die kanonischen Elemente von $\Sigma x : A.B$ sind Paare (a, b) mit $a : A$ und $b : B[x := a]$ vom Typ, der durch Ersetzung aller frei vorkommenden x in B durch α entsteht. Die Elemente von $\Pi x : A.B$ sind (berechenbare) Funktionen f mit $fa : B[x := a]$, wobei $a : A$.

Wie lassen sich Theoreme als Typen nach dem Curry-Howard Isomorphismus verstehen? Als Beispiel betrachten wir das Theorem, wonach es beliebig große Primzahlen gibt, d.h. formal:

$$\forall m : \mathbb{N}.\exists n : \mathbb{N}.m < n \wedge Prime(n)$$

Nach dem Curry-Howard Isomorphismus wird daraus der Typ von Funktionen, die eine Zahl m auf ein Tripel $(n, (p, q))$ abbilden, wobei n eine Zahl ist, p ein Beweis, dass $m < n$ ist, und q ein Beweis, dass n eine Primzahl ist:

$$\Pi m : \mathbb{N}.\Sigma n : \mathbb{N}.m < n \times Prime(n)$$

Das ist ein Beispiel für das Beweis-als-Programm Prinzip: Danach wird ein konstruktiver Beweis, dass es beliebig große Primzahlen gibt, zu einem Programm, das für gegebene Zahlen eine größere Primzahl zusammen mit Beweisen erzeugt, dass sie in der Tat größer und Primzahl ist.

Zusätzlich zu den Typenformern des Curry-Howard Isomorphismus erweiterte der Logiker und Philosoph P. Martin-Löf die basale intuitionistische Typentheorie, die Heyting

Arithmetik der höheren Typen HA^{ω} und Gödels System T der primitiv-rekursiven Funktionen höheren Typs enthält, mit primitiven Identitätsttypen, wohl-geordneten Baumtypen, Hierarchien von Universen und allgemeinen Begriffen induktiver und induktiv–rekursiver Definitionen. Martin-Löfs beweistheoretische Erweiterung betrifft sowohl die Anwendung auf die Programmierung als auch auf die Formalisierung der Mathematik [Ma98].

Neben den gegebenen Regeln für Π gibt es analoge Regeln für andere Typenformer, die den logischen Konstanten der typisierten Prädikatenlogik entsprechen. In der intuitionistischen Typenarithmetik werden wie in der Peano-Arithmetik die natürlichen Zahlen mit 0 und der Nachfolgeroperation s erzeugt. Um Widersprüche mit unbegrenzten Begriffsbildungen wie dem Universum aller Typen zu vermeiden, führte Martin-Löf das Universum kleiner Typen ein. Das typentheoretische Universum U ist analog zu einem Grothendieck-Universum in der Mengenlehre. Das Grothendieck-Universum ist eine Menge von Mengen, die abgeschlossen unter allen Methoden ist, mit denen Mengen in der Zermelo-Fraenkel Mengenlehre gebildet werden können. Das Grothedieck-Universum deckt weite Möglichkeiten mathematischer Begriffsbildung ab.

In der intuitionistischen Typentheorie ist das Auswahlaxiom ein beweisbares Theorem. Es ist nämlich eine unmittelbare Folge der BHK-Interpretation der intuitionistischen Quantoren. In der Mengenlehre ist das Auswahlaxiom allerdings nicht immer konstruktiv. Typen sind im Allgemeinen keine konstruktiven Approximationen von Mengen im klassischen Sinn. Mit Blick auf Datenstrukturen in der Informatik ist bemerkenswert, wie sich wohlgeordnete Bäume und iterative Mengen in der intuitionistischen Typentheorie darstellen lassen. Wohlfundierte Bäume können auf ein typentheoretisches Modell der konstruktiven Mengenlehre erweitert werden [Ac78].

In der intuitionistischen Typentheorie werden induktive Typen durch Konstruktoren eingeführt:

Beispiele:

a) Typ $\mathbb{N}$ der natürlichen Zahlen mit Konstruktoren
   - $0 : \mathbb{N}$
   - $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$

b) Typ $\mathrm{List}(A)$ der endlichen Listen von Elementen des Typs A mit Konstruktoren
   - $\mathrm{nil} : \mathrm{List}(A)$ (leere Liste)
   - $\mathrm{cons} : A \to \mathrm{List}(A) \to \mathrm{List}(A)$ (Hinzufügung eines Elements am Anfang der Liste)
   - $\mathrm{app} : \mathrm{List}(A) \to \mathrm{List}(A) \to \mathrm{List}(A)$ (Verbindung zweier Listen)

Ein Induktionsprinzip beweist eine Behauptung für einen Typ, der durch seine Konstruktoren frei erzeugt ist.

2.2 Von der Typentheorie zur univalenten Mathematik

In der Tradition von Leibnizens Mathesis Universalis stellt sich die Frage, bis zu welchem Grad das Denken von (mengentheoretischen) Unendlichkeiten durch formale Codes („Typen“) dargestellt werden kann, die durch „Maschinen“ bearbeitbar sind. In der Informatik lassen sich Typen des funktionalen Programmierens als erste Schritte in diese Richtung verstehen.

Seit ihren Anfängen spielen Datentypen eine Schlüsselrolle in Computersprachen: Wieweit können mathematische Objekte mit Typen von Computersprachen repräsentiert werden? Die Homotopie Theorie stammt aus der algebraischen Topologie und homologischen Algebra mit Beziehungen zur höheren Kategorientheorie, die als abstraktes Fundament der Mathematik verstanden werden kann. Demgegenüber ist die Typentheorie ein Zweig der mathematischen Logik und theoretischen Informatik. Die Homotopie Typentheorie (Homotopy Type Theory = HoTT) verbindet beide Ansätze und interpretiert Typen als Objekte der abstrakten Homotopie Theorie. Daher versucht HoTT, sowohl eine universale („univalente“) Grundlegung der Mathematik als auch einen Kalkül für einen Beweisassistenten zu entwickeln.

In HoTT entsprechen Terme der intuitionistischen Typentheorie solchen der Homotopie Theorie: In der intuitionistischen Typentheorie kann ein Term $a : A$ als ein Element a vom Typ A oder als ein Beweis der Aussage A oder, in der Homotopie Typentheorie, als ein Punkt des Raums A verstanden werden. Beweise p der Identität zwischen zwei Elementen a, b vom Typ A werden geometrisch als Pfade veranschaulicht, die die entsprechenden Punkte verbinden.

HoTT verbindet die Typentheorie mit Beweistheorie, Mengenlehre und Homotopie Theorie [UFP13]: Mit höheren induktiven Typen lassen sich Geometrie und Topologie charakterisieren. Wie die bisherigen induktiven Typen sind höhere induktive Typen ein allgemeines Schema, um neue Typen zu definieren, die durch einige Konstruktoren erzeugt werden (Einführungsregeln). Im Unterschied zu gewöhnlichen induktiven Typen erzeugen diese Konstruktoren nicht nur Punkte dieses Typs, sondern auch Pfade zwischen diesen Punkten, Pfade zwischen Pfaden zwischen Punkten und weiter höhere Pfade in diesem Typ [AW09].

Grundlage von HoTT ist das von dem russischen Mathematiker W. Voevodsky eingeführte Univalenz Axiom (UA). Dabei bezieht sich Äquivalenz auf höher dimensionale Objekte von der Mengenlehre bis zur Kategorientheorie (z.B. gleiche Elemente, isomorphe Mengen, äquivalente Groupoide). UA besagt, dass „alles“ durch Äquivalenz erhalten wird. Insbesondere sind äquivalente Typen (z.B. isomorphe Strukturen) identisch.

Die Homotopie Typentheorie (HoTT) eröffnet grundlegende Zusammenhänge zwischen Beweisbarkeit, Konstruktivität und Berechenbarkeit. HoTT soll ermöglichen, mathematische Beweise in eine Programmiersprache für maschinelle Beweisassistenten sogar für höhere mathematische Kategorien mit “Isomorphismen als Gleichheit” (UA) zu übertragen (z.B. Coq). Daher lautet ein wesentliches Ziel von HoTT:

Typenprüfung $\Rightarrow$ Beweisprüfung in höheren Kategorien (d.h. für „schwierige Beweise“)

HoTT ist durch höhere induktiv definierte Strukturen (z.B. induktiv definierte Räume mit Kollektionen von Punkten, Pfaden, höheren Pfaden etc.) erweitert, die durch passende Induktionsprinzipien charakterisiert werden können. HoTT ist klassisch widerspruchsfrei mit Bezug auf ein semantisches Modell in der Kategorie von Kan Komplexen [Vo12, Jo02], kann aber auch konstruktiv begründet werden (T. Coquand).

Die Motivation, sich mit HoTT zu beschäftigen, hat mit einer neuen Art von Grundlagenkrise zu tun, der sich die moderne Mathematik zunehmend stellen muss. In der gegenwärtigen Mathematik könnten nämlich Beweise so kompliziert werden, dass ein einzelner Mathematiker sie kaum in allen Details prüfen kann: Man vertraut auf die Expertise seiner Kolleginnen und Kollegen. Die Situation ist ähnlich zur modernen industriellen Arbeitswelt. Nach dem französischen Soziologen Emile Durkheim (1858-1917) ist die moderne industrielle Produktion so komplex, das sie nur nach dem Prinzip der Arbeitsteilung ("job sharing") und des gegenseitigen Vertrauens in Expertise organisiert werden kann. Niemand hat dabei aber den totalen Überblick über alle Details.

Auf dem Hintergrund gravierender Fehler, die von Experten übersehen wurden, wollte der Wladimir Voevodsky (1966-2017, IAS Princeton, Fields Medaille) aber nicht länger dem Prinzip des "job-sharing" in der Mathematik trauen. Menschen könnten in Zukunft mit der wachsenden Komplexität der Mathematik überfordert sein. Sind Computer die einzige und letzte Lösung? Voevodskys Grundlagenprogramm univalenter Mathematik ist durch die Idee einer Software für Beweisprüfung inspiriert, um Vertrauen & Verifikation in der Mathematik zu garantieren. Diese Herausforderung der Beweisprüfung lässt auf eine Grundlagenkrise der modernen Software im allgemeinen erweitern. Wie soll die Sicherheit von immer komplexer werdenden Computerprogrammen (z.B. in der Künstlichen Intelligenz) garantiert werden? [Ma18, Kap. 2] Die folgenden Abschnitte legen dafür die ersten Schritte mit Beweisassistenten, die auf dem Formalismus von HoTT aufbauen.

2.3 Von der Typentheorie zu Beweisassistenten

Der Kalkül der Konstruktionen (Calculus of Constructions = CoC) ist eine Typentheorie von T. Coquand et al., die sowohl als typisierte Programmiersprache als auch als konstruktive Begründung der Mathematik dienen kann [CH88]. Ähnlich wie Martin-Löf erweitert dieser Kalkül den Curry-Howard Isomorphismus auf Beweise im ganzen intuitionistischen Prädikatenkalkül. Coc benötigt nur sehr wenige Konstruktionsregeln für Terme. Die Objekte von CoC sind Beweise (Terme mit Propositionen als Typen), Propositionen (kleine Typen), Prädikate (Funktionen, die Propositionen zurückgeben), große Typen (Typen von Prädikaten, z.B. P), T (Typ von großen Typen). Die Ableitungsregeln von CoC lassen sich wieder in einem Sequenzkalkül zusammenstellen. Logische Operatoren und Datenypen in CoC benötigen sehr wenige Basisoperatoren. Der einzige logische Operator zur Bildung weiterer Operatoren und Datentypen ist $\forall$ (z.B. $\forall C : P.(A \Rightarrow C)$ für die Negation $\neg A$).

Der Kalkül der induktiven Konstruktionen (Calculus of inductive Constructions = CiC) basiert auf CoC und verfügt über zusätzliche induktive und ko-induktive Definitionen, die durch folgende Regeln zur Konstruktion von Termen eingeführt werden [BC04]. In CiC wird ein induktiver Typ frei durch eine bestimmte Anzahl von Konstruktoren erzeugt [Pa93].

Induktive Beweise ermöglichen es, Behauptungen für unendliche Kollektionen von Objekten zu beweisen (z.B. ganze Zahlen, Listen, binäre Bäume), weil alle diese Objekte in einer endlichen Anzahl von Schritten konstruiert werden. Das Induktionsprinzip eines induktiven Typs beweist eine Behauptung für einen Typ, der frei durch seine Konstruktoren erzeugt ist.

Neben induktive Types gibt es ko-induktive Typen, die unendliche Objekte betreffen (z.B. potentiell unendliche Listen, potentiell unendliche Bäume mit unendlichen Ästen) [Gi96]. Terme erhält man durch wiederholten Gebrauch von Konstruktoren, wie bereits für induktiven Typen gezeigt wurde. Allerdings gibt es kein Induktionsprinzip und die Äste können unendlich sein.

In praktischen Anwendungen wie Telekommunikation, Energie oder Transport lassen sich diese (potentiell) unendlichen Objekte auch als Ströme (z.B. Daten-, Energie- und Warenströme) illustrieren. Sie entstehen in unbegrenzt vielen Schritten, die durch den Konstruktor Cons definiert werden. Im Unterschied zum induktiven Typ einer Liste gibt es keinen Konstruktor der leeren Liste. Daher können endliche Listen nicht konstruiert werden.

Auf dieser Grundlage lässt sich nun der Coq Beweisassistent einführen [BC04]. Coq implementiert eine Programmspezifikation, die auf dem Kalkül der induktiven Konstruktionen (CiC) beruht und beides eine Logik höherer Ordnung und eine reich typisierte funktionale Sprache verbindet. Die Befehle von Coq erlauben,

- Funktionen oder Prädikate zu definieren (die effizient evaluiert werden können)
- mathematische Theoreme und Software-Spezifikationen zu behaupten
- formale Beweise von diesen Theoremen interaktiv zu entwickeln
- diese Beweise durch eine relativ kleine Zertifikation maschinell zu prüfen
- zertifizierte Programme zu extrahieren (e.g., Objective Caml, Haskell, Scheme).

Coq liefert interaktive Beweismethoden, Entscheidungs- und Semi-Entscheidungsalgorithmen. Interaktiv bedeutet, dass der Kalkül ständig durch die extrahierten Beweisstrategien und Routinen erweitert wird, die dann automatisch für neue Problemlösungen zur Verfügung stehen. Verbindungen mit externen Theorembeweisern sind verfügbar. Coq ist also eine Plattform für die Verifikation sowohl von mathematischen Beweisen als auch von Computerprogrammen auf der Grundlage von CiC.

2.4 Verifikation von Schaltkreisen mit typentheoretischen Beweisassistenten

Ein Hardware oder Software Programm ist korrekt („zertifiziert durch Coq“), falls verifiziert werden kann, dass es einer gegebenen Spezifikation in CiC folgt. Eine entsprechende Verifikation wird nun am Beispiel eines Schaltkreises in der Elektrotechnik erläutert [CJ96]. Dabei kann die Struktur bzw. Architektur eines Schaltkreises und sein tatsächliches Verhalten mathematisch durch verschaltete endliche Automaten modelliert werden. In Schaltkreisen sind potentiell unendlich lange Zeitfolgen von Daten (Ströme) zu berücksichtigen.

Ein Schaltkreis ist korrekt genau dann, wenn unter bestimmten Bedingungen die Output-Ströme des Automaten, der die Schaltkreisstruktur darstellt, äquivalent sind zum Automaten, der das Verhalten des Schaltkreises beschreibt. Daher muss zunächst die Automatentheorie in CiC mit den ko-induktiven Typen von Strömen implementiert werden.

Die Korrektheit eines Schaltkreises wird also durch die Äquivalenz seiner Struktur und seines Verhaltens bewiesen. Dabei werden Struktur und Verhalten durch zwei zusammengesetzte Automaten modelliert. Die Äquivalenz von zusammengesetzten Automaten kann durch ein Äquivalenz-Lemma invarianter Relationen bewiesen werden [CJ96]. Dieses Lemma kann ebenfalls in CiC implementiert werden. Dieses Lemma ist ein Beispiel für eine Beweisroutine von Coq, mit der Verifikationen von Schaltkreisen unter bestimmten Bedingungen automatisch realisiert werden können.

Wie aktuell Software-Verifikationen sind, zeigt die jüngste Technikentwicklung. Inkorrektheit von Programmen kann zu Katastrophen führen, die von falschen Angaben von Strahlungsdosierungen in der Medizin, die Patienten schädigten, über die Explosion der Rakete Ariane V aufgrund eines Programmierfehlers bis zu Software- und Systemfehlern von Boing 737max in 2019 reichen. Solche Unfälle werfen ein Schlaglicht auf die Gefahren sicherheitskritischer Systeme ohne Software Verifikation. Dramatisch wird die Situation in den Programmen der Künstlichen Intelligenz, die wegen ihrer Komplexität zu “schwarzen Kästen” werden [Ma19]. Im mathematischen Idealfall könnten Beweisassistenten, wie sie in diesem Artikel beschrieben wurden, ihren exakten Betrieb garantieren. Davon sind wir allerdings in der Praxis noch weit entfernt. Selbst in der Mathematik sind Beweisassistenten bisher nur eine große Vision auf eine zukünftige sichere Welt, die allerdings theoretisch möglich wäre.

3 Grundlagen und Verifikation des Machine Learning

Im Unterschied zu den logischen Formalismen symbolischer KI orientiert sich das moderne Machine Learning an statistischen Lernverfahren neuronaler Netze. Der Anwendungserfolg des Machine Learning beruht darauf, dass große Datenmengen mit leistungsstarken Rechnern bewältigt werden können. Wie sicher ist aber statistisches Lernen im Unterschied zu formalen Verifikationsverfahren der symbolischen KI?

3.1 Vom Bayesschen Lernen zum Machine Learning

Eine zentrale Herausforderung für KI-Anwendungen ist die Komplexität der betrachteten Systeme – von zellulären Organismen über Robotiksysteme bis zum Internet. Die Bayessche Wissenschaftstheorie erklärt statistisch, wie sich unsere Modelle und Hypothesen über die Welt durch neue Erfahrungen verändern und wie wir aus Erfahrung lernen können. Nach dem Bayesschen Theorem kommt es zur Bestimmung der Wahrscheinlichkeit $P(M|D)$ eines Modells M unter Voraussetzung der Datenmenge D zunächst darauf an, die Datenplausibilität (Likelihood) $P(D|M)$ eines Modells und seine apriorische Wahrscheinlichkeit $P(M)$ einzuschätzen.

Sie kann daher als Rahmentheorie für Lernalgorithmen verstanden werden, mit denen Lernverfahren im Machine Learning automatisiert werden. Diese Automatisierung ist z.B. in der Bioinformatik, Robotik oder im Internet von zentraler Bedeutung, da die komplexen Datenmengen und ihre Verbindung mit großen Mengen von Modellparametern immer schwieriger zu durchschauen sind [Ma20, Kap. 4]. Im Machine Learning spielen neuronale Netze nach dem Vorbild des menschlichen Gehirns eine dominante Rolle. Der Durchbruch der KI-Forschung in der Praxis hängt wesentlich mit der Fähigkeit neuronaler Netze zusammen, große Datenmengen (Big Data) z.B. bei der Mustererkennung mit effektiven Lernalgorithmen anzuwenden. Praktische Anwendungen erfordern Tausende von Neuronen und Synapsen in mehrschichtigen neuronalen Netzen (deep learning), die statistisch mit endlich vielen Datensätzen von Inputs und Outputs trainiert werden.

Vom Bayesschen Standpunkt aus lassen sich neuronale Netze als graphische Modelle $M(w)$ mit bestimmten Parametern w auffassen. Diese Modelle erinnern vereinfacht an das menschliche Gehirn. In klassischen vorwärtslaufenden (feed-forward) neuronalen Netzen sind Neuronen als Knoten eines Graphen in Schichten angeordnet. Jedes Neuron einer Schicht ist mit allen Neuronen der nachfolgenden Schicht durch gerichtete Kanten (Synapsen) verbunden, die Neuronen einer Schicht untereinander aber nicht. Ausnahmen sind die Input-Schicht, deren Neuronen keine einlaufenden Verbindungen besitzen, und die Output-Schicht, deren Neuronen keine auslaufenden Verbindungen haben. Die Schichten zwischen Input- und Output-Schicht heißen „versteckt“ (hidden). Jede Verbindung/Kante ist im graphischen Modell eines neuronalen Netzes mit einer Zahl gewichtet, die der Intensität der synaptischen Verbindung entspricht. Diese Gewichte (englisch: weights) sind die Parameter w des graphischen Modells $M(w)$ eines neuronalen Netzes. Jedes Neuron ist durch eine Aktivierungsfunktion charakterisiert, mit der die Input-Output Relation für dieses Neuron definiert wird. In Analogie zum Gehirn sagt man, dass ein Neuron „feuert“ bzw. erregt ist, wenn die Summe der gewichteten Inputs seiner Nachbarzellen einen bestimmten Schwellenwert überschreitet.

3.2 Was sind und wozu brauchen wir Kausalmodelle?

Die zentrale Herausforderung sind die gewaltigen Datenmengen D mit Tausenden von „versteckten“ (englisch: hidden) Parametern H ihrer statistischen Modelle M. Die sich daraus ergebenden globalen Wahrscheinlichkeitsverteilungen $P(D, M, H)$ sind mathematisch häufig nicht berechenbar. Was sich im Detail tatsächlich unter diesen statistischen Datenwolken abspielt, bleibt verborgen.

Statistische Datenkorrelationen können Hinweise auf kausale Zusammenhänge liefern, müssen es aber nicht. Stellen wir uns eine Testreihe vor, bei der sich eine günstige Korrelation zwischen einer verabreichten chemischen Substanz und der Bekämpfung bestimmter Krebstumore ergibt. Ein aktuelles Beispiel ist auch die Entwicklung eines Impfstoffes gegen den Corona-Virus. Auch in diesem Fall entsteht Druck des betroffenen Unternehmens, mit einem entsprechenden Medikament in die Produktion zu gehen und Gewinne abzuschöpfen. Aber auch betroffene Patienten mögen darin ihre letzte Chance sehen. Tatsächlich erhalten wir ein nachhaltiges Medikament aber nur, wenn wir den zugrunde liegen kausalen Mechanismus des Tumorwachstums, also die Gesetze der Zellbiologie und Biochemie verstanden haben.

Statistisches Lernen und Schließen aus Daten reichen also nicht aus. Wir müssen vielmehr die kausalen Zusammenhänge von Ursachen und Wirkungen hinter den Messdaten erkennen. Diese kausalen Zusammenhänge hängen von den Gesetzen der jeweiligen Anwendungsdomäne unserer Forschungsmethoden ab, also den Gesetzen der Physik oder den Gesetzen der Biochemie und des Zellwachstums im Beispiel der Krebsforschung. Wäre es anders, könnten wir mit den Methoden des statistischen Lernens und Schließen bereits die Probleme dieser Welt lösen. Tatsächlich scheinen das einige kurzsichtige Zeitgenossen beim derzeitigen Hype der Künstlichen Intelligenz zu glauben.

Statistisches Lernen und Schließen ohne kausales Domänenwissen ist aber blind – bei noch so großer Datenmenge (Big Data) und Rechenpower. Galilei und Newton erkannten grundlegende Gesetze aus nur wenigen Beobachtungsdaten. Neben der Statistik der Daten bedarf es zusätzlicher Gesetzes- und Strukturannahmen der Anwendungsdomänen, die durch Experimente und Interventionen überprüft werden. Kausale Erklärungsmodelle (z.B. das Planetenmodell Newtons oder ein Tumormodell) erfüllen die Gesetzes- und Strukturannahmen einer Theorie (z.B. die Gravitationstheorie oder die Gesetze der Zellbiologie).

Beim kausalen Schließen werden Eigenschaften von Daten und Beobachtungen aus angenommenen Kausalmodellen, d.h. Gesetzesannahmen von Ursachen und Wirkungen, abgeleitet. Kausales Schließen ermöglicht damit, die Wirkungen von Interventionen oder Datenveränderungen (z.B. durch Experimente) zu bestimmen. Kausales Lernen versucht umgekehrt, ein Kausalmodell aus Beobachtungen, Messdaten und Interventionen (z.B. Experimente) abzuleiten, die zusätzliche Gesetzes- und Strukturannahmen voraussetzen.

Ein Kausalmodell besteht aus einem System von Zuordnungen von Ursachen zu Wirkungen mit eventuellen Störvariablen. Ursachen und Wirkungen werden durch Zufallsvariablen beschrieben. Ihre funktionalen Zuordnungen (unter Berücksichtigung von Störvariablen) werden durch Gleichungen definiert, also z.B. Wirkung $X_j = f(X_i, R)$ in funktioneller Abhängigkeit von Ursache X_i und Stör- und Rauschvariable R. Anschaulich kann das Netzwerk der Ursachen und Wirkungen durch einen Graphen von Knoten und Kanten dargestellt werden. Zufallsvariablen von Ursachen und Wirkungen entsprechen Knoten. Kausale Wirkungen entsprechen gerichteten Pfeilen: $X_i \rightarrow X_j$ bedeutet, dass Ursache X_i Wirkung X_j auslöst.

Es lässt sich beweisen, dass ein Kausalmodell eine eindeutige Wahrscheinlichkeitsverteilung der Daten einschließt, aber nicht umgekehrt: Für Kausalmodelle (z.B. Planetenmodell) müssen zusätzliche Gesetze (z.B. Gravitationsgesetz) angenommen werden [MJS13]. Um kausale Abhängigkeiten und Unabhängigkeiten von Ereignissen zu erkennen, muss die Abhängigkeit und Unabhängigkeit der sie darstellenden Zufallsvariablen ermittelt werden. Statistisch lässt sich die Unabhängigkeit der Resultate x und y zweier Zufallsvariablen (anschaulich Zufallsexperimente) X und Y dadurch ausdrücken, dass ihre Verbundwahrscheinlichkeit $p(x, y)$ faktorisierbar ist, d.h. $p(x, y) = p(x)p(y)$. Man spricht in diesem Fall auch von der Markov-Bedingung.

3.3 Vom statistischen zum kausalen Lernen

Ein hochaktuelles Anwendungsbeispiel sind selbstlernende Fahrzeuge: Um das Prinzip zu erläutern, können wir uns vereinfacht ein elektrisches Spielzeugauto vorstellen, das rund herum mit Sensoren ausgestattet ist. Die Sensoren (z.B. Nachbarschaft, Licht, Kollision) seien mit den Neuronen eines neuronalen Netzwerks verbunden. Werden benachbarte Sensoren bei einer Kollision mit einem äußeren Gegenstand erregt, dann auch die mit den Sensoren verbundenen Neuronen. So entsteht im neuronalen Netz ein Verschaltungsmuster, das den äußeren Gegenstand repräsentiert. Im Prinzip ist dieser Vorgang ähnlich wie bei der Wahrnehmung eines äußeren Gegenstands durch einen Organismus – nur dort sehr viel komplexer. Wenn wir uns nun noch vorstellen, dass dieses Automobil mit einem „Gedächtnis“ (Datenbank) ausgestattet wird, mit dem es sich solche gefährlichen Kollisionen merken kann, um sie in Zukunft zu vermeiden, dann ahnt man, wie die Automobilindustrie in Zukunft unterwegs sein wird, selbst-lernende Fahrzeuge zu bauen.

Hier zeigt sich aber eine grundlegende Schwäche des derzeitigen maschinellen Lernens: Wie viele reale Unfälle sind erforderlich, um selbstlernende (“autonome”) Fahrzeuge zu trainieren? Wer ist verantwortlich, wenn autonome Fahrzeuge in Unfälle verwickelt sind? Welche ethischen und rechtlichen Herausforderungen stellen sich? Bei komplexen Systemen wie neuronalen Netzen mit Tausenden oder sogar Millionen von Elementen erlauben zwar die Gesetze der statistischen Physik, globale Aussagen über Trend- und Konvergenzverhalten des gesamten Systems zu machen. Die Zahl der Parameter ist jedoch unter Umständen so groß, dass keine lokalen Ursachen ausgemacht werden können. Die neuronalen Netze sind

also eine Black Box, die mit Big Data trainiert wird, um gewünschtes Verhalten zu erzeugen. Keiner weiß im Einzelnen, was dort in der Black Box abgeht. Wenn aber Ursachen und Wirkungen nicht klar zu unterscheiden sind, lassen sich rechtliche und ethische Fragen der Verantwortung nicht klären. *Ehe wir also über Ethik und Recht sprechen, müssen wir unsere Hausaufgaben in der Grundlagenforschung des maschinellen Lernens machen.*

Tatsächlich ist das maschinelle Lernen häufig nur Statistik mit Lernalgorithmen und neuronalen Netzen - mathematisch keineswegs spektakulär wie in den Medien suggeriert. Jeder Anfänger der Statistik weiß, dass statistische Korrelationen keine kausalen Erklärungen ersetzen können: Wenn eine günstige statistische Korrelation zwischen einer chemischen Substanz und dem Abnehmen eines Krebstumors gefunden wurde, ist das noch keine Garantie für ein nachhaltiges Medikament. Dazu muss man das Grundlagenwissen über die kausalen Wachstumsgesetze eines Tumors und biochemische Grundgesetze kennen. Dasselbe gilt für einen Impfstoff gegen den Corona Virus SARS-CoV-2.

Mit diesem Beispiel verbinde ich eine grundsätzliche Feststellung für den heutigen KI-Hype: Einige glauben damit ja bereits auf Wasser gehen und alle Probleme dieser Welt in absehbarer Zeit mit „KI“ lösen zu können. Erfolgreich sind diese KI-Methoden aber nur dann, wenn sie mit Fachwissen und Theorie aus den jeweiligen Anwendungsgebieten (wie z.B. Physik, Medizin und Ingenieurwissenschaften) verbunden werden.

Statistisches Lernen und Schließens ist jedenfalls nur schwache KI, die jeder einfache Organismus in der Natur auch ohne statistische Formeln bewältigt: Selbst ein Wurm wird nach gehäuften Erfahrungen von gefährlichen Situationen davor zurückschrecken. Was über statistische Mustererkennung in Daten hinausgeht, ist die Fähigkeit zu kausalem Lernen und Schließen. Gibt es Algorithmen, mit denen sich kausale Modelle unter geeigneten Bedingungen finden lassen? Dieses kausale Lernen wäre ein erster Schritt in Richtung einer starken KI. Tatsächlich ist kausales Lernen mittlerweile Thema theoretischer Grundlagenforschung. Aber es bedarf noch vieler Forschung, bis einmal eine Software z.B. in einem biochemischen Datensatz automatisch ein kausales Erklärungsmodell entdecken und damit eine begründete medizinische Diagnose geben kann. Wie sicher ist jedoch ein Softwareprogramm, wenn es zunehmend mehr oder weniger intelligente Entscheidungen selbstständig treffen soll?

3.4 Sicherheit und Vertrauen von Machine Learning durch logische Beweise?

Ein Computerprogramm heißt korrekt bzw. zertifiziert, falls verifiziert werden kann, dass es einer gegebenen Spezifikation folgt. Praktisch angewendet werden Verifikationsverfahren mit unterschiedlichen Graden der Genauigkeit und damit der Verlässlichkeit [TB03]. Aus Zeit-, Aufwands- und Kostengründen begnügen sich viele Anwender allerdings nur mit Stichprobentests. Im Idealfall müsste ein Computerprogramm aber so sicher sein wie ein mathematischer Beweis. Dazu wurden Beweisprogramme („Beweisassistenten“) entwickelt, mit denen ein Computerprogramm automatisch oder interaktiv mit einem Nutzer auf Korrektheit überprüft wird.

Künstliche neuronale Netze sind zwar äußerst effektiv, um komplexe Probleme (real world problems) zu bearbeiten [Ma19]. Was aber fehlt, sind Spezifikationen und Standards für die Sicherheit ihrer Outputs. Dazu muss die Black Box neuronaler Netze besser verstanden, kontrolliert und verifiziert werden. Die Verifikation neuronaler Netze ist allerdings ein hartes Erkenntnisproblem: Selbst der Nachweis einfacher Eigenschaften erweist sich im Rahmen der Komplexitätstheorie als NP-vollständig. Gründe dafür sind die Größe der praktisch angewendeten Netze (Skalierung) und die nichtlinearen Aktivierungsfunktionen ihrer Neuronen, die von Menschen in diesem Umfang und mit dieser Geschwindigkeit nicht nachvollzogen werden können. Da neuronale Netze zudem der Dynamik komplexer Systeme unterliegen, sind sie häufig empfindlich gegen kleine Störungen und Veränderungen ihrer Inputs, die sich zu unkontrollierbaren Effekten aufschaukeln können. Robustheit und Stabilität der Netze hängt also mit ihrer Sicherheit eng zusammen.

Für unterschiedliche Klassen neuronaler Netze lassen sich unterschiedliche Verifikationen angeben, die aus verschiedenen Theorien der Logik und Mathematik ableitbar sind. Dazu wurden Verifikationsverfahren vorgeschlagen, die auf der Erfüllbarkeit von Formeln der Booleschen Aussagenlogik (SAT = Satisfiability Theories), Erfüllbarkeit von Formeln der Prädikatenlogik 1. Stufe (SMT = Satisfiability Modulo Theores), Reduktion auf lineare Probleme (MIP = Mixed Integer Linear Programming) und Robustheit von mehrschichtigen Perzeptron-Netzen (MLP = Multi-Layer Perceptron) beruhen. Bei SAT- und SMT-Verifikationen wird die klassische KI (symbolic AI) des automatischen Beweisens (automated reasoning) mit dem Machine Learning verbunden. MIP beruht auf der Logik und Algebra linearen Programmierens. Robustheitsuntersuchungen von MLP wenden Erkenntnisse aus der Theorie komplexer dynamischer Systeme im Machine Learning an.

SAT-Verifikationen stehen in der Tradition der klassischen KI. Sie sind mittlerweile nicht nur von theoretischem Interesse, sondern wurden für industrielle Anwendungen ausgebaut [FH07]. Neuronale Netze lassen sich durch verschiedene Rand- und Nebenbedingungen (constraints) charakterisieren. Logisch liegt es daher nahe, die Verifikation eines neuronalen Netzes auf die Verifikation der Formeln zurückzuführen, mit denen sich ihre Rand- und Nebenbedingungen formalisieren lassen. Bei der SAT-Verifikation werden solche neuronalen Netze untersucht, deren Rand- und Nebenbedingungen sich durch Formeln der Booleschen Aussagenlogik charakterisieren lassen. Verifikation besteht dann in dem Nachweis der logischen Erfüllbarkeit dieses Formelsystems. Damit sollen seine Widerspruchsfreiheit und der sichere Ablauf im entsprechenden neuronalen Netz garantiert werden.

Hier zeigt sich sehr klar, wie aktuelle Fragen der Sicherheit moderner Software und KI in Grundlagenfragen der Logik und Philosophie verwurzelt sind. Derzeit beschäftige ich mich mit der Frage, wie das moderne maschinelle Lernen durch solche Beweisassistenten kontrolliert werden kann [MSS18]. Am Ende geht es um die Herausforderung, ob und wie man KI-Programme zertifizieren kann, bevor man sie auf die Menschheit loslässt. Statistisches Lernen, wie es heute praktiziert wird, funktioniert zwar häufig in der Praxis, aber die kausalen Abläufe bleiben oft unverstanden und eine Black Box. Statistisches Testen und Probieren reicht für sicherheitskritische Systeme nicht aus. Daher plädiere ich in der

Zukunft für eine Kombination von kausalem Lernen mit zertifizierten KI-Programmen durch Beweisassistenten, auch wenn das für Praktiker aufwendig und ambitioniert erscheinen mag.

4 Verifikation und Technikgestaltung

KI-Programme treten mittlerweile aber nicht nur in einzelnen Robotern und Computern auf. So steuern bereits lernfähige Algorithmen die Prozesse einer vernetzten Welt mit exponentiell wachsender Rechenkapazität. Ohne sie wäre die Datenflut im Internet nicht zu bewältigen, die durch Milliarden von Sensoren und vernetzten Geräten erzeugt wird. Aufgrund der Sensoren kommunizieren nun also auch Dinge miteinander und nicht nur Menschen. Daher sprechen wir vom Internet der Dinge (Internet of Things: IoT). Im industriellen Internet („Industrie 4.0“) wird das Internet der Dinge auf die Industrie- und Arbeitswelt angewendet. Künstliche Intelligenz und Machine Learning werden dazu in den Arbeitsprozess integriert. Werkstücke kommunizieren untereinander, mit Transporteinrichtungen und beteiligten Menschen, um den Arbeitsprozess flexibel zu organisieren. Produkte können so individuell zur gewünschten Zeit nach Kundenwünschen erstellt werden. Technik, Produktion und Markt verschmelzen zu einem soziotechnischen System, das sich selbst flexibel organisiert und sich verändernden Bedingungen automatisch anpassen soll.

Die sicherheitskritischen Herausforderungen, die wir eben erörtert haben, werden sich in solchen Infrastrukturen noch einmal potenzieren. Darüber hinaus stellt sich aber die Frage nach der Rolle des Menschen in einer mehr oder weniger automatisierten Welt. Ich plädiere daher für Technikgestaltung, die über Technologiefolgenabschätzung hinausgeht. Die traditionelle Sicht, die Entwickler einfach werkeln zu lassen und am Ende die Folgen ihrer Ergebnisse zu bewerten, reicht aus Erfahrung nicht aus. Am Ende kann das Kind in den Brunnen gefallen sein und es ist zu spät. Nun lässt sich zwar Innovation nicht planen. Wir können aber Anreize für gewünschte Ergebnisse setzen. Ethik wäre dann nicht Innovationsbremse, sondern Anreiz zu gewünschter Innovation. Eine solche ethische, rechtliche, soziale und ökologische Roadmap der Technikgestaltung für KI-Systeme würde der Grundidee der sozialen Marktwirtschaft entsprechen, nach der ein Gestaltungsspielraum für Wettbewerb und Innovation gesetzt wird. Maßstab bleibt aber die Würde des einzelnen Menschen, wie sie im Grundgesetz der Verfassung als oberstes Axiom der parlamentarischen Demokratie festgelegt ist.

Literaturverzeichnis

[Ac78] Aczel, P.: The type theoretic interpretation of constructive set theory., in: A. Macintyre, L. Pacholski, J. Paris (eds.), Logic Colloquium ’77. North-Holland : Amsterdam-New York, 55–66, 1978.

[AZ01] Aigner, M. und Ziegler, G.M.: Proofs from The Book, Berlin 2. Aufl., 2001.

[AW09] Awodey, S. Warren, M.A.: Homotopy theoretic models of identity type, in: Mathematical Proceedings of the Cambridge Philosophical Society 146 (1), 45-55, 2009.

[BC04] Bertot, Y. und Castéran, P.: Interactive Theorem Proving and Program Development: Coq'Art: CiC, Springer, 2004.

[BG18] Bringsjord, Selmer; N. S. Govindarajulu, Naveen Sundar. Artificial Intelligence, The Stanford Encyclopedia of Philosophy (Fall 2018 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/fall2018/entries/artificial-intelligence.

[CM81] Clocksin, William F.; Mellish, Christopher S.: Programming in Prolog. Berlin: Springer, 1981.

[CJ96] S. Coupet-Grimal; L. Jakubiec: Coq and Hardware Verification: a Case Study (TPHOLs , 1996, LCNS 1125, 125-139).

[CH88] Coquand, T; Huet, G.: The calculus of constructions (Coc), in: Information and Computation 76(2–3), 95–120, 1988.

[DP60] Davis, M.; Putnam, H.: A computing procedure for quantification theory, in: J.ACM 7(3): 210-215, 1960.

[Da+62] Davis, M.; Logemann, G.; Loveland, D.: A machine program for theorem-proving, in: Commun. ACM 5(7): 394–397, 1962.

[DP16] Dybjer, P. und Palmgren, E.: Intuitionistic Type Theory, in: The Stanford Encyclopedia of Philosophy, The Metaphysics Research Lab (CSLI), Stanford University: Stanford (open access), 2016.

[Fe96] Feferman, S.: Kreisel's „unwinding" Program, in: P. Odifreddi (Ed.) Kreisleriana. About and Around Georg Kreisel, Review of Modern Logic, 1996, 247-273.

[FH07] Fränzle, Martin und Christian Herde: HySAT: An efficient proof engine for bounded model checking of hybrid systems, in: Formal Methods in System Design 30:179-198, 2007.

[Gi60] Gilmore, P. C.: A proof method for quantification theory: its justification and realization, in: IBM J. Res. Dev. 4(1): 28–35, 1960.

[Gi96] Giménez, E.: Un calcul de constructions infinies et son application à la vérification de systèmes communicants (PhD thesis Lyon), 1996.

[He34] Heyting, A.: Mathematische Grundlagenforschung. Intuitionismus, Beweistheorie, Springer, Berlin, 1934, repr. 1974.

[Ho69] Howard, W.A.: The formulae-as-types notion of construction, in: J. P. Seldin, J.R. Hindley (Hrsg.), To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press: Boston, MA, 479–490, 1969.

[Jo02] Joyal, A.: Quasi-categories and Kan complexes, in: Journal of Pure and Applied Algebra 175, 207-222, 2002.

[Kn16] Knuth, D. E.: Satisfiability. The Art of Computer Programming, Vol 4, Fasc. 6. Boston: Addison Wesley, 2016.

[Ko08] Kohlenbach, U.: Applied Proof Theory: Proof Interpretations and Their Use in Mathematics, Berlin, 2008.

[Ko32] Kolmogorov, A.N.: Zur Deutung der intuitionistischen Logik, in: Math. Z. 35, 58-65, 1932.

[KS00] Küchlin, Wolfgang; Sinz, C.: Proving Consistency Assertions for Automotive Product Data Management, in: J. Automated Reasoning 24:145-163, 2000.

[KM20] Küchlin, W.; Mainzer, K.: Logische Grundlagen der klassischen KI, in: K. Mainzer (Hrsg.), Philosophisches Handbuch der Künstlichen Intelligenz, Springer, 2020.

[Ma20] Mainzer, K.: Leben als Maschine: Wie entschlüsseln wir den Corona-Kode? Von der Systembiologie und Bioinformatik zu Robotik und Künstlicher Intelligenz, Brill Mentis: Paderborn 2. erweiterte Auflage, 2020.

[Ma19] Mainzer, K.: Künstliche Intelligenz. Wann übernehmen die Maschinen? Springer: Berlin 2. erweiterte Auflage (engl. Übersetzung), 2019.

[Ma18] Mainzer, K.: The Digital and the Real World. Computational Foundations of Mathematics, Science, Technology, and Philosophy, World Scientific Singapore, 2018.

[MSS18] Mainzer, K; Schuster, P.; Schwichtenberg, H. (Eds.): Proof and Computation. Digitization in Mathematics, Computer Science, and Philosophy. World Scientific Singapore, 2018.

[Ma95] Marques-Silva, J. P.: Search Algorithms for Satisfiability Problems in Combinatorial Switching Circuits. PhD Thesis, U. Michigan, 1995.

[MS96] Marques-Silva, J. P.; Sakallah, Karem A.: GRASP-A new search algorithm for satisfiability, in: Proceedings of International Conference on Computer Aided Design, 220-227, 1996.

[MS99] Marques-Silva, J. P.; Sakallah, Karem A.: GRASP: A search algorithm for propositional satisfiability, in: IEEE Transactions on Computers 48 (5): 506-521, 1999.

[Ma98] Martin-Löf, P.: An intuitionistic theory of types. Twenty-five years of constructive type theory (Venice, 1995), in: Oxford Logic Guides 36, Oxford University Press, New York, 127-172, 1998.

[MJS13] Mooij, J.M. ; Janzing, D.; B. Schölkopf, B.: From ordinary differential equations to structural causal models: The deterministic case, in: Proceedings of the 29th Annual Conference on Uncertainty in ArtificiaL Intelligence (UAI), 440-448, 2013.

[Ni71] Nilsson, Nils J.: Problem Solving Methods in Artificial Intelligence. New York: Mc Graw Hill, 1971.

[Ni80] Nilsson, N. J.: Principles of Artificial Intelligence. Palo Alto: Tioga Publishing Co., 1980.

[Pa93] Paulin-Mohring, C.: Inductive Definition in the System Coq: Rules and Properties (Research Report 92-49, LIP-ENS Lyon), 1993.

[Qu55] Quine, W. V. O.: A Proof Procedure for Quantification Theory, in: J. Symbolic Logic 20: 141-149, 1955.

[Sc06] Schwichtenberg, H.: Minlog, in: F. Wiedijk (ed.), The Seventeen Provers of the World. Lecture Notes in Artificial Intelligence vol. 3600, Berlin, 151-157, 2006.

[SW12] Schwichtenberg, H. und Wainer, S. S.: Proofs and Computations, Cambridge, 2012.

[UFP13] The Univalent Foundations Program: Homotopy Type Theory: Univalent Foundations of Mathematics. Princeton, NJ: Institute for Advanced Study, 2012.

[TB03] J. Tretmans; E. Brinksma: TorX: Automated model-based testing, in: A. Hartman and K. Dussa-Zieger (Eds.), Proceedings of the First European Conference on Model-Driven Software Engineering, 2003.

[Vo12] Voevodsky, W.: A universe polymorphic type system. http://uf-ias-2012.wikispaces.com/file/view/Universe+polymorphic+type+sytem.pdf, 2012.