

Formale Methoden der Künstlichen Intelligenz

Christoph Kreitz

Aufgrund der zunehmenden Komplexität von Hard- und Softwaresystemen werden computergestützte formale Methoden in der Zukunft immer weiter an Bedeutung zunehmen. Aufgabe dieser Methoden ist es, menschlichen Benutzern bei der Verifikation und Synthese von Computersystemen auf intelligente Weise zu assistieren. In diesem Artikel werden Ziele, Herausforderungen und Entwicklungen im Bereich formaler Methoden der Künstlichen Intelligenz angesprochen, die aller Voraussicht nach lohnenswerte Aufgabenstellungen für die Forschung der kommenden Jahre darstellen.

1 Einleitung

In der Informatik bezeichnet der Begriff der Formalen Methoden eine Vielzahl von Techniken zum Modellieren und rigorosen Überprüfen von Computersystemen. Dabei ist das Verständnis dessen, was als formal anzusehen ist, in den einzelnen Disziplinen durchaus sehr verschieden. Im Software-Engineering sind formale Methoden im wesentlichen ingenieurwissenschaftliche Techniken, die den Softwareentwicklungsprozess systematisieren und sich hierzu auf formale Notationen wie zum Beispiel UML stützen. In der Künstlichen Intelligenz dagegen versteht man unter formalen Methoden computergestützte Verfahren zur Erzeugung und Überprüfung von Hard- und Softwaresystemen. Diese Verfahren basieren in der Regel auf formaler mathematischer Logik und setzen Techniken des automatischen Beweisens als zentrale Mechanismen ein.

Diese Unterschiede spiegeln sich auch in den Forschungsschwerpunkten der entsprechenden Disziplinen wider. Während im Software-Engineering der Schwerpunkt auf der Entwicklung von Leitlinien und Methoden liegt, mit denen ein Team menschlicher Softwarespezialisten schneller und kostengünstiger zu einem besseren Produkt kommen kann, steht in der Künstlichen Intelligenz die Entwicklung eines intelligenten Assistenten im Vordergrund, der unter Anleitung eines menschlichen Benutzers einen Großteil der Arbeit von Hard- und Softwarespezialisten übernehmen kann. Dabei steht als zusätzliche Anforderung im Raum, daß der intelligente Assistent im Gegensatz zu seinem menschlichen Gegenüber keine Fehler machen darf. Eine formale *Verifikation*, die Überprüfung eines Computersystems durch den intelligenten Assistenten, soll eine Garantie dafür liefern, daß das System die überprüften Eigenschaften wirklich besitzt. Bei einer formalen *Synthese* soll der intelligente Assistent aus einer gegebenen Spezifikation ein funktionsfähiges Computersystem entwickeln, das diese Spezifikation garantiert erfüllt.

Im Laufe der Jahrzehnte hat sich das Bild formaler Methoden in der KI gewandelt. Während man in der Euphorie der sechziger Jahre noch davon ausging, in naher Zukunft intelligente Assistenten bauen zu können, welche Hard- und Softwaresysteme vollautomatisch überprüfen und synthetisieren, führte in den folgenden Jahrzehnten der langsame Fortschritt der Forschung zu einer gewissen Ernüchterung, zumal die Notwendigkeit von Synthese und Verifikation für die Praxis immer noch in Frage gestellt wurde. Erst mit dem Erfolg von Model Checkern in der Hardwareverifikation begannen logische Techniken für die Industrie ak-

zeptabel zu werden. Dies beschränkte sich zunächst jedoch auf den Hardwarebereich, für den vollautomatische und hocheffiziente Beweistechniken zur Verfügung standen. Im Softwarebereich wurden dagegen die größten Anwendungserfolge der letzten zehn Jahre von strategisch gestützten interaktiven Systemen erzielt, die bisher nur von wenigen Spezialisten erfolgreich bedient werden können. Die aktuellen Arbeiten in der Programmsynthese und -verifikation konzentrieren sich auf den Ausbau der praktischen Anwendbarkeit derartiger Systeme.

Die ursprüngliche KI-Vision von einem brauchbaren intelligenten Assistenten für menschliche Benutzer ist damit jedoch nicht verlorengegangen. Verschieben hat sich nur der Ausgangspunkt, von dem aus auf eine künstliche Realisierung von Intelligenz im Bereich der Softwareentwicklung hingearbeitet wird. Basis ist nun nicht mehr das vollautomatische System, das durch Effizienzsteigerungen einen immer größeren Anwendungsbereich bewältigen soll, sondern ein vielseitiges interaktives Grundsystem, das bei entsprechender Benutzersteuerung im Prinzip jedes Problem lösen kann und durch Einbau intelligenter Strategien immer größere Aufgaben des Benutzers übernehmen kann.

Welche Entwicklungen im Bereich formaler Methoden der Künstlichen Intelligenz können wir in den nächsten Jahren erwarten? Welche Ziele sollten verfolgt werden? Auch wenn es natürlich nicht möglich ist, die Zukunft vorauszusagen, lassen sich aus dem derzeitigen Stand der Technik in der Künstlichen Intelligenz und in der Informatik Trends und Herausforderungen ablesen, die in der nahen Zukunft vor uns liegen werden. In diesem Artikel sollen, nach einem kurzen Abriss der historischen Entwicklung, Ziele und zukünftige Entwicklungen im Bereich formaler Methoden der Künstlichen Intelligenz angesprochen werden, die aus Sicht des Autors lohnenswerte Aufgabenstellungen für die Forschung darstellen.

2 Historische Entwicklung

Eine der zentralen Aufgaben von formaler Synthese und Verifikation ist die Erstellung von Hard- und Softwaresystemen mit garantierten Eigenschaften. Dabei soll eine *Verifikation* nachweisen, daß ein gegebenes System *korrekt* ist, also eine bestimmte formale Spezifikation erfüllt, während eine *Synthese* ein korrektes System aus einer gegebenen formalen Spezifikation erzeugen soll. Damit konzentrieren sich beide Prozesse auf die Aspekte der Hard- und Softwareentwicklung, die sich zumindest im Prin-

zip vollständig formalisieren und auch automatisieren lassen, da die Frage der *Validierung*, d.h. ob die formale Spezifikation die vom System zu lösende Aufgabe wirklich genau beschreibt, von ihnen nicht berührt wird.

Formale Verifikation und Synthese sind komplementäre Aufgabenstellungen, die mit ähnlichen deduktiven Mechanismen bearbeitet werden können. Da das Ziel der Verifikation ist, existierende Systeme auf ihre Eigenschaften zu überprüfen, können sich Verifikationsverfahren schneller an praktisch relevanten Problemen orientieren, während die automatische Synthese eher der ursprünglichen KI-Vision eines intelligenten Assistenten entspricht, dafür aber lange als akademisches Spielzeug angesehen wurde.

Die ersten Ansätze¹ zur Verifikation und Synthese [16, 50, 30] entstanden aus der Vision, daß ein allgemeiner Problemlösungsmechanismus für logisches Schließen auch für die Behandlung von Programmierproblemen einsetzbar sein sollte. Die Verwendung automatischer Theorembeweiser und Termersetzungssysteme lieferte zu Beginn vielversprechende Ergebnisse.

Bei der Verifikation von Hardware (oder auch Softwaresystemen mit endlichen Zustandsmengen) brachten dramatische Effizienzsteigerungen bei aussagenlogischen Beweisverfahren in den 90er Jahren einen Durchbruch. Gleichzeitig führten die schweren wirtschaftlichen Auswirkungen von Fehlern in kommerziellen Prozessoren zu einer großen Nachfrage nach Verifikationswerkzeugen aus der Industrie. Seit dem Pentium-FDIV-Bug ist *Model checking* [11, 12, 29] ein fester Bestandteil des Entwurfs von Hardware geworden.

Bei der erheblich schwierigeren Verifikation und Synthese von Software scheiterten derart universelle Ansätze im Endeffekt jedoch lange am Skalierungsproblem: die für die Beschreibung von Programmierproblemen notwendigen Formeln waren für automatische Beweisverfahren erheblich zu komplex. Auch wenn *Software model checking* [21, 22] in den letzten Jahren erfolgreich in Fallstudien zur Verifikation kommerzieller Software eingesetzt werden konnte, erscheinen aus Sicht der Künstlichen Intelligenz Ansätze, die stärker auf ein intelligentes Verhalten des Verifikations- und Syntheseverfahrens abzielen und semantisches *Wissen* zur Steuerung syntaktischer Methoden einsetzen, langfristig erfolgversprechender zu sein. Schon in den achtziger Jahren wurde im LOPS Ansatz [8] eine kleine Menge allgemeiner Programmwurfstrategien durch Wissen über Eigenschaften verschiedener Anwendungsbereiche gesteuert. Noch konsequenter war die Vorgehensweise des KIDS Systems [44, 46, 45], in dem auch Wissen über algorithmische Strukturen und algorithmische Optimierung explizit gemacht wurde. Mit diesem Ansatz gelang es erstmalig, Programmsyntheseverfahren bei der Erzeugung kommerzieller Algorithmen einzusetzen und Scheduling Algorithmen zu generieren [47, 15], die nicht nur garantiert korrekt waren sondern auch erheblich effizienter als alle Algorithmen, die bis zu dieser Zeit von Hand erzeugt waren.

Der Erfolg von KIDS beruhte allerdings nicht nur auf der konsequenten Verwendung semantischen Wissens sondern auch auf einer *Kooperation* mit menschlichen Experten bei der Bearbeitung einer Problemstellung. Dies entsprach einem wachsenden Trend, formale Methoden als intelligente Verfahren auszugestalten, die Wissen über Programmierung dazu verwenden, um menschliche Benutzer bei der Lösung von Programmier- oder Verifikationsproblemen zu unterstützen. Dieses Ziel kommt

auch der KI-Vision eines intelligenten Assistenten erheblich näher als der ursprüngliche Versuch, vollautomatische Programmierer oder Verifikationssysteme zu erstellen. Systeme, die heutzutage erfolgreich für die Erzeugung und Verifikation von Software eingesetzt werden, stützen sich daher meist auf interaktive Beweisassistenten, die eine Reihe von Strategien zur Verfügung stellen, welche von erfahrenen Benutzern gesteuert werden. So konnte z.B. mit taktischen Systemen wie KIV [19, 20, 4], VSE [25], und PVS [37, 41, 42] Anwendungssoftware mit mehreren tausend Zeilen Code und mit dem Higher-Order Beweissystem Isabelle [38, 39, 40, 33] abstrakte kryptographische Protokolle verifiziert werden. Der konstruktive Beweisassistent Nuprl [13, 1] wurde für den formalen Entwurf sowie für die Verifikation und Optimierung verteilter Systeme mit mehr als 50000 Zeilen Programmcode eingesetzt [27, 9, 26]. Dies zeigt, daß man bei entsprechender Führung durch menschliche Benutzer bereits heute in der Lage ist, auch größere Softwaresysteme formal zu verifizieren oder zu generieren.

Diese Erfolge sind jedoch noch Einzelfälle, die von Expertenbenutzern der jeweiligen Systeme mit verhältnismäßig hohem Aufwand erzielt wurden. Im Gegensatz zur Hardwareverifikation sind im Softwarebereich die Verifikations- und Synthesysteme noch weit davon entfernt, für normale Benutzer eine brauchbare Hilfe zu sein. Bisher gibt es nur sehr wenige Benutzer, welche die entsprechenden Systeme überhaupt bedienen können, und der erfolgreiche Einsatz der Systeme verlangt gute Kenntnisse der zugrundeliegenden Logiken und Techniken. Bei komplexen Problemstellungen ist zudem die Anzahl der nötigen Benutzereingriffe noch zu groß, um eine Verifikation oder Synthese in akzeptabler Zeit durchführen zu können.

Kernziel der Forschungen im Bereich formaler Methoden der KI sollte daher sein, neben der Hardwareverifikation auch die Verifikation und Synthese von Softwaresystemen für einen industriellen Einsatz verwendbar zu machen. Dies verlangt die Bereitstellung leicht zu nutzender formaler Werkzeuge mit klaren und einfachen Benutzungsvorschriften, die mit sehr wenigen Benutzereingriffen in akzeptabler Zeit eigenständig eine Lösung liefern. Bei Model-checkern haben derartige Eigenschaften signifikant zur Akzeptanz von formaler Methoden in der industriellen Hardwareverifikation beigetragen. Dieselben Eigenschaften für formale Methoden im Softwarebereich zu erzielen ist eine der größten Herausforderungen an die KI-Forschung der nächsten Jahrzehnte, denn sie verlangt die Erstellung intelligenter Systeme, deren Fähigkeiten und Komplexität weit über das bisher Erreichte hinausgeht.

In den folgenden Abschnitten sollen einige Forschungsrichtungen angesprochen werden, die auf dem Weg, dieses Ziel zu erreichen, verfolgt werden sollten. Dabei gehen wir davon aus, daß die Entwicklungen der kommenden Jahre im wesentlichen von zwei Faktoren beeinflusst werden: von dem Bestreben der Künstlichen Intelligenz, intelligente logische Verfahren zu entwickeln, die ihrem menschlichen Vorbild in mancherlei Hinsicht ebenbürtig oder sogar überlegen sind, und von den zu erwartenden Entwicklungen in der Informatik, die neue Herausforderungen an die Fähigkeiten dieser Verfahren stellen werden.

¹Einen Überblick und systematischen Vergleich verschiedener Ansätze zur Programmsynthese findet man in [6].

3 Intelligente Beweisassistenten

Computergestützte Verfahren zur Konstruktion logisch gültiger Schlußfolgerungen sind Kernbestandteil jedes intelligenten Systems. Theorembeweiser und Beweisassistenten sind ein eigenständiges Forschungsgebiet mit vielfältigen Anwendungen. Im folgenden werden wir uns auf Aspekte konzentrieren, welche für den praktischen Erfolg formaler Methoden in der Hard- und Softwareentwicklung besonders wichtig sind. Einige der hier angesprochenen Themen sind schon seit einiger Zeit in Bearbeitung, werden die Forscher aber noch für einige Zeit beschäftigen. Andere werden erst seit kurzem verfolgt oder können erst in den kommenden Jahren in Angriff genommen werden. Sie lassen sich unterteilen in allgemeine Inferenztechniken, Methoden zur Verwaltung und Verarbeitung semantischen Wissens sowie die Interaktion mit den vorgesehenen Benutzern.

Allgemeine Inferenztechniken

Ein großer Teil der Forschungen im Bereich der Inferenzsysteme konzentriert sich derzeit immer noch auf die Erzeugung hocheffizienter SAT- und QBF-Solver (Lösungsverfahren für das aussagenlogische Erfüllbarkeitsproblem bzw. das Problem der Gültigkeit quantifizierter Boolescher Formeln) sowie auf automatische Theorembeweiser für Logiken erster Stufe. In den letzten Jahren wurden signifikante Leistungssteigerungen erzielt. Auswertungen mit umfangreichen Problemsammlungen für klassische und konstruktive Beweisverfahren [48, 36] zeigen jedoch, daß diese Performanzsteigerungen immer noch nicht ausreichen, um einen signifikanten Anteil praktisch relevanter Problemstellungen automatisch lösen zu können. Bisher konnte noch kein nichttriviales Verifikations- oder Syntheseproblem alleine von einem automatischen Beweisverfahren gelöst werden. Beim Lösen komplexer Probleme zeigt sich, daß Intelligenz mehr ist als nur ein schnelles Verarbeiten einfachster logischer Schlußfolgerungen. Es kommt auch darauf an, verschiedene Lösungsmethoden an passender Stelle anwenden und ggf. kombinieren zu können. Daher ist zu erwarten, daß allgemeine Beweissuchprozeduren in der Forschung zwar nach wie vor einen großen Stellenwert haben werden, aber daß andere Beweistechniken und ihre effiziente Integration immer größere Bedeutung erhalten werden.

Zu diesen Techniken gehören insbesondere die schon seit vielen Jahren erforschten Induktionsbeweisverfahren, Termersetzungssysteme und Entscheidungsprozeduren, die erforderlich sind um Schleifen und rekursive Aufrufe in Programmen zu analysieren, Programmausdrücke symbolisch auszuwerten sowie einfache Aussagen über lineare Arithmetik, Listen und Kongruenzen zu beweisen. Auch wenn es in diesen Bereichen in den letzten Jahren große Fortschritte gegeben hat, ist insbesondere die Frage nach effizienten Kontrollstrategien für diese Techniken immer noch nicht zufriedenstellend beantwortet.

Untersucht werden sollten aber auch Hilfstechniken für die Erzeugung komplexer Beweise, die sich mit den derzeitigen Methoden kaum erzeugen lassen. Dazu gehört zum Beispiel die automatische Erzeugung von Fallunterscheidungen, wenn ein geradliniger Beweis nicht möglich ist. Bisher geschieht dies nur, wenn sich ein Teil A der gegebenen Behauptung nicht beweisen läßt. In diesem Fall wird die Fallunterscheidung $A \vee \neg A$ eingeführt und die beiden Fälle dann einzeln analysiert. In mathematischen Beweisen findet man jedoch noch eine Reihe anders-

artiger Fallunterscheidungen, die sich oft aus der Struktur der in der Aussage vorkommenden Konzepte ergeben. Noch allgemeiner und deutlich schwieriger ist die Erzeugung von Lemmata, welche eine Teilaussage generalisieren und erst dadurch einen Beweis dieser Aussage ermöglichen. Diese Technik hat in der Mathematik einen hohen Stellenwert und stellt eine große Herausforderung an die Intelligenz von Inferenzsystemen dar, da sie sich, von wenigen Spezialfällen abgesehen, bisher nicht automatisieren läßt.

Nicht zufriedenstellend erforscht sind auch Metatechniken wie die automatische Wiederverwendung von (Teil-)Beweisen, der Einsatz von Analogieargumenten, sowie eine Analyse von (automatisch erzeugten) Gegenbeispielen, Trugschlüssen, und fehlschlagenden Beweisversuchen. In der Mathematik werden all diese Methoden dazu verwendet, um Erkenntnisse für die Erzeugung neuer Beweise zu gewinnen. In der Künstlichen Intelligenz lassen sie sich voraussichtlich am besten als eine Methode der *Beweisplanung* [10, 7] darstellen, da sie eigentlich nur einen groben Beweisplan liefern, der dann noch im Detail ausgeführt und überprüft werden muß.

Kooperierende verteilte Inferenzsysteme

Neben der Erforschung einzelner Techniken sollten vor allem aber auch Methoden zur effektiven Integration verschiedener Techniken und Beweiswerkzeuge untersucht werden, da davon auszugehen ist, daß größere Problemstellungen nur durch das Zusammenspiel verschiedener Methoden gelöst werden können. So könnte zum Beispiel eine Verifikation zunächst eine logische Dekomposition benötigen, bevor Induktion angewandt werden kann, und bei der nachfolgenden Analyse mit einem Theorembeweiser könnte anstelle einer einfachen Unifikation der Aufruf einer Entscheidungsprozedur, eines Computeralgebrasystems, eines Constraint Solvers oder einiger Termersetzungsschritte erforderlich werden. Umgekehrt könnte ein SAT Solver durch Bestimmung des "unsatisfiability-cores" aus einer großen Anzahl von Formeln mit Aussagen zu den in der zu verifizierenden Software vorkommenden Begriffen eine kleine Menge von Formeln identifizieren, die für die Beweisführung wirklich relevant sind, wodurch das Problem dann für einen Theorembeweiser handhabbar wird.

Es gibt zwei grundsätzliche Arten, wie verschiedene Beweistechniken zu einem Gesamtsystem kombiniert werden können. Der vielleicht naheliegenste Ansatz ist, ein bestehendes Beweissystem durch Einbettung anderer Techniken leistungsfähiger zu machen. So kann man zum Beispiel den Unifikationsmechanismus einer automatischen Beweissuchprozedur um Rewrite Techniken, Entscheidungsprozeduren und Computeralgebrasystemen erweitern und die allgemeine Suchstrategie so ergänzen, daß sie bei der Behandlung von Induktionsschritten zunächst nach einem Zusammenhang zwischen Induktionsannahme und -schluß sucht. Der Vorteil dieser Vorgehensweise ist eine enge Integration der Komponenten und eine größere Kontrolle über das Verhalten des Gesamtsystems. Der große Nachteil dieses "Master-Slave" Ansatzes ist jedoch der massive Eingriff in die Struktur des ursprünglichen Beweissystems, der im Extremfall eine Veränderung aller wichtigen Datenstrukturen erforderlich macht, und der hohe Codierungsaufwand für die Integration der zusätzlichen Komponenten, die entweder vollständig neu codiert oder um Code für die Anpassung an das Master-System ergänzt werden müssen.

Außerdem macht es dieser Ansatz unnötig schwer, Komponenten gegen verbesserte Versionen auszutauschen, da hierbei erneut Arbeit für die Anpassung erforderlich ist.

Eine sinnvollere Architektur für eine Integration verschiedener Inferenzsysteme ist die der kooperierenden verteilten Prozesse, die über einen gemeinsamen formalen Bus miteinander kommunizieren [43, 14, 23]. Sie eignet sich besonders dann, wenn die einzelnen Beweiswerkzeuge wie eigenständige Agenten agieren, die nur gelegentlich miteinander kommunizieren müssen. In diesem Ansatz werden ungelöste Probleme in einer für alle Komponenten verständlichen Einheitssyntax wie z.B. OPENMATH [35] oder OMDOC [24, 34] auf den Bus gestellt und dann nur von den Komponenten bearbeitet, die im Prinzip für die jeweilige Problemstellung geeignet sind, wobei gegebenenfalls neue Teilprobleme auf den Bus gestellt werden. Diese Architektur wurde erstmalig dazu eingesetzt um das Beweisplanungssystem Ω MEGA [7] mit Theorembeweisern, Computeralgebrasystemen und Constraint-Solvern zu verbinden, und läßt sich im Prinzip dazu verwenden, beliebige formale Systeme bei der Lösung eines Problems kooperieren zu lassen. Sie ermöglicht die parallele Bearbeitung mehrerer Teilprobleme, die Verwendung vielfacher Kopien derselben Komponente sowie den Austausch einzelner Komponenten im laufenden Betrieb. Außerdem bleibt das Gesamtsystem auch beim Ausfall einzelner Komponenten funktionsfähig, solange noch genügend viele andere Komponenten zur Verfügung stehen und für die Kommunikation ein fehlertolerantes Gruppenkommunikationssystem verwendet wird.

Neben dem Aspekt der Architektur muß bei der Integration verschiedener deduktiver Werkzeuge vor allem auch die Frage der Korrektheit des Gesamtergebnisses geklärt werden. Die einzelnen Komponenten basieren oft auf verschiedenen Logiken, die nicht immer vollständig miteinander kompatibel sind. Wünschenswert wäre es, von den einzelnen Komponenten *Beweisobjekte* zu erhalten, die unabhängig geprüft werden können. Da dies nicht immer möglich sein wird, sollten die einzelnen Inferenzschritte zumindest mit *Zertifikaten* annotiert werden, aus denen erkenntlich ist, mit welchen Beweismethoden die jeweiligen Teilergebnisse erzielt wurden. Diese Technik, die erstmalig in der formalen Wissensbank des Nuprl Systems [2, 1] eingesetzt wurde, macht Beweise, die mit Hilfe verschiedener Werkzeuge erzeugt wurden, für die Benutzer transparent und ermöglicht es ihnen, Beweise zu verwerfen oder neu ausführen zu lassen, wenn bestimmte Werkzeuge ihren Anforderungen nicht genügen.

Die Entwicklung kooperierender verteilter Beweisprozeduren steckt bisher noch in ihren Anfängen. Ziel der Forschungen der kommenden Jahre wird sein, ein Integrationsmodell zu entwerfen, das flexibel, effizient, robust, aus logischer Sicht vertrauenswürdig und aus Benutzersicht möglichst unkompliziert ist, und dieses Modell dann in die Praxis umzusetzen.

Verwaltung und Verarbeitung von Wissen

Wissenschaftliche Argumentation besteht neben der Ausführung logischer Schlüsse vor allem auch aus der Anwendung semantischen Wissens. In der Mathematik bauen die Beweise wichtiger Sätze nicht nur auf den Axiomen der jeweiligen Theorien sondern auf allen bis dahin gesammelten Erkenntnissen auf. In der Softwareentwicklung setzen gute Programmierer nicht nur ihr Wissen über die Konstrukte der jeweilige Programmiersprache ein, sondern Erkenntnisse über Datenstrukturen und Program-

miermethoden, die sie während ihrer Ausbildung gelernt haben.

Für die formalen Methoden der Künstlichen Intelligenz bedeutet das, daß die Verarbeitung formalen semantischen Wissens eine wesentliche Rolle spielen wird. Formales Wissen bildet das Fundament für die Anwendung von Inferenzsystemen in der Praxis. Dieses Wissen muß gesammelt, systematisiert, verwaltet und für Inferenzen in geeigneter Form bereitgestellt werden. Diese Thematik hat in den letzten Jahren zunehmende Bedeutung erhalten und wird unter anderem auf den MKM Workshops (Mathematical Knowledge Management) intensiv diskutiert.

Die meisten formalen Beweisassistenten besitzen eine Wissensbank, die wegen der Vielfalt der formalisierten Begriffe oft jedoch nur eine fragmentarische Sammlung von Basislemmata enthält und von einzelnen Benutzern bei Bedarf ergänzt werden muß. Für die Anwendung dieser Systeme in der Praxis ist dies sehr unbefriedigend, da Formulierung und Beweis fehlender Lemmata eine zeitraubende Tätigkeit ist, welche die Nutzer von der Bearbeitung ihres eigentlichen Ziels abhält und wenig intellektuelle Erkenntnisse liefert. Es ist daher notwendig, den systematischen Aufbau großer formaler Theorien mit tausenden von Lemmata maschinell zu unterstützen. Da viele Lemmata einer Theorie sich mit Aussagen zur Kombination von zwei oder drei Konzepten befassen, kann die Formulierung eines Teils dieser Lemmata schematisch generiert und dann bewiesen oder mit Gegenbeispielen widerlegt werden. Menschliche Benutzer könnten sich dann auf den Beweis nichttrivialer Aussagen zu mathematischen Grundlagentheorien, Datenstrukturen, und Operationen sowie auf die Formulierung anspruchsvolleren Wissens zu algorithmischen Strukturen und Entwurfsverfahren konzentrieren.

Neben dem Inhalt einer Wissensbank ist auch ihre Verwaltung von großer Bedeutung. Wissen muß gut genug strukturiert sein, um Benutzern und automatischen Beweisstrategien die Suche nach geeigneten Lemmata zu erleichtern. Ein Zertifizierungsmechanismus ist nötig, um transparent zu machen, aus welchem Grunde bestimmtes Wissen gespeichert wird, also zum Beispiel mit welchen Beweistechniken ein Lemma bewiesen wurde. Um Konsistenz zu sichern, muß die Wissensbank feststellen können, von welchen Annahmen, Lemmata und Werkzeugen die Gültigkeit eines Theorems in der Wissensbank abhängt. Auf diese Art kann sichergestellt werden, daß die Wissensbank keine zirkulären Referenzen enthält und daß Änderungen in der Formulierung eines Lemmas zur Überprüfung von Beweisen führen, in denen dieses Lemma benutzt wurde.

Da formales Wissen an vielen Stellen mit verschiedenen Systemen erstellt wird, sollten Mechanismen entwickelt werden, die ähnlich wie verteilte Datenbanken eine verteilte Lagerung formalen Wissens unterstützen. Um formale Mathematik an verschiedenen Stellen auf dem Web lagern und zugänglich machen zu können [31], benötigt man einheitliche Standards zur Kommunikation formaler Mathematik [35, 24, 34], Zertifizierungsmechanismen zur Kennzeichnung der Herkunft von Informationen, Synchronisierungsmechanismen und Techniken, die verschiedene Formulierungen desselben Konzepts ineinander übersetzen um die entsprechenden Lemmata anwendbar zu machen.

Wegen der zu erwartenden Größe formaler Wissensbanken ist es auch erforderlich, Methoden zu entwickeln, welche das für eine bestimmte Beweissituation relevante Wissen identifizieren können. Ähnlich wie Datenbanken müssen Wissensbanken Suchmechanismen bereitstellen, um Lemmata, die bestimmte Kriterien wie zum Beispiel das Vorkommen bestimmter Operationen

erfüllen, effizient aus der Wissensbank extrahieren zu können. Mit diesen Mechanismen kann die Effizienz wissensabhängiger Inferenztechniken signifikant gesteigert werden.

Interaktion mit Benutzern

Ein lange vernachlässigter Aspekt bei der Entwicklung formaler Methoden der KI ist die Entwicklung von Benutzerinterfaces, die es "Laienbenutzern" ermöglichen, den Umgang mit dem formalen System in kurzer Zeit zu erlernen. Zukünftige Anwender müssen mit den Systemen arbeiten können, ohne die logischen Hintergründe oder Beweismechanismen im Detail zu verstehen. Das bedeutet, daß das Interface vertraute Notationen bereitstellen und Interaktionskonzepte anbieten sollte, die den üblichen Standards entsprechen und damit nahezu selbsterklärend sind. Es sollte Benutzern in jeder Situation eine kleine Auswahl anwendbarer Strategien bereitstellen, anstatt von ihnen zu erwarten, die Namen aller Strategien und die Bedeutung ihrer Parameter zu erlernen. Es soll ermöglichen, Beweise zu inspizieren und das verwendete Wissen zurückzuverfolgen. Gleichzeitig sollte es eine Reihe von Features anbieten, die Expertennutzer des Systems verwenden können, um in komplexen Anwendungen effizienter zum Ziel zu kommen.

Die Entwicklung geeigneter Interfaces für formale Methoden ist seit einigen Jahren das Thema der UITP Workshops (User Interfaces for Theorem Provers). Für Systeme wie KIV [18] und Ω MEGA [49] gibt es bereits prototypische Interfaces, die einen wichtigen Schritt in Richtung praktisch nutzbarer Interfaces darstellen. Bisher orientieren sich Interfaces jedoch noch zu sehr an Benutzern, die gewisse Erfahrungen mit logischen Deduktionsverfahren besitzen. Es ist jedoch nicht zu erwarten, daß industrielle Anwender in absehbarer Zukunft diese Kenntnisse besitzen werden. Daher sollte sich die Entwicklung an den Standards der Human-Computer Interfaces orientieren, um eine größere Akzeptanz formal-logischer Werkzeuge in der Praxis zu erreichen.

4 Anforderungen aus der Informatik

Eine der Schlüsselfragen beim Entwurf formaler Methoden ist, in welchem Maße sie geeignet sind, Beiträge zur Lösung aktueller Probleme der Informatik zu leisten. Wegen der immer stärker zunehmenden Komplexität von Software ist davon auszugehen, daß der Bedarf nach formalen Methoden weiter zunehmen wird. Es ist daher nicht verwunderlich, daß die Forschungslabors großer Firmen vor einiger Zeit damit begonnen haben, mit verschiedenen Theorembeweisern, Model Checkern, Computeralgebrasystemen und anderen formalen Systemen zu experimentieren, um auf dieser Basis eigene automatische Verifikations- und Synthesysteme für die Anforderungen der kommenden Jahre zu entwickeln [5, 3]. Genauso muß die akademische Forschung im Bereich formaler Methoden der Entwicklung in der Informatik folgen und sich darauf vorbereiten, Antworten auf die Herausforderungen der Zukunft liefern zu können. Formale Methoden müssen in den kommenden Jahren mit veränderten Architekturen und Algorithmenstrukturen umgehen können und neben der Korrektheit (im Sinne von "bug free") auch Eigenschaften wie Fehlertoleranz, Sicherheit, Laufzeitverhalten oder Ressourcenbedarf überprüfen.

Neue Strukturen und Architekturen

In Ihrem Bericht *Towards 2020 Science* [17] hat eine Gruppe international anerkannter Wissenschaftler versucht, Trends und Herausforderungen zu identifizieren, welche Entwicklung der Informatik in den kommenden 15 Jahren beeinflussen werden. Demnach ist davon auszugehen, daß physikalische Schranken für die Geschwindigkeit von Prozessoren dazu führen werden, daß Software zunehmend aus einer komplexen Mischung von parallelen und verteilten Algorithmen bestehen wird, welche weitestgehend die hohe Kommunikationsgeschwindigkeit und Zuverlässigkeit von multi-core Prozessoren ausnutzen, aber auf langsamere und weniger stabile Netzwerke zurückgreifen, wenn die eine Anwendung zu komplex für eine einzelne Maschine wird. Dabei werden fehler- und verzögerungstolerante asynchron kommunizierende Algorithmen zunehmend Bedeutung erhalten. Diese Entwicklungen zu unterstützen wird eine der größten Herausforderungen an formale Methoden sein. Es müssen präzise Modelle sowie Verifikations- und Synthesetechniken formuliert werden, welche den robusten Entwurf verteilt kommunizierender Komponenten und die flexible Komposition solcher Komponenten zu einem qualitativ hochwertigen und effizienten Gesamtsystem ermöglichen. Der Ansatz in [27] ist ein erster Schritt in diese Richtung, der erheblich vertieft, erweitert und an neue Softwarearchitekturen angepaßt werden muß.

Aufgrund der explodierenden Menge von Daten in vielen Wissenschaftsdisziplinen werden verteilte Datenbanken ebenfalls eine sehr wichtige Rolle spielen. Die schnelle Verarbeitung großer und vielseitiger Datenmengen, eine gute Verteilung von Daten, intelligentes Data Mining, die "semantische" Analyse von Daten anhand von Metadaten und die Interpretation ihrer Bedeutung mithilfe formaler Werkzeuge stellen große Herausforderungen an die Forschung dar. Es müssen Algorithmen und Methoden entwickelt werden, die sich ständig an neue Anwendungsszenarien und Plattformen anpassen können und dabei jederzeit fehlerfrei arbeiten.

Katastrophale Folgen von Einheitenfehlern bei der Interpretation wissenschaftlicher Meßdaten Ende der neunziger Jahre haben zu einem wachsenden Interesse an Programmiersprachen mit leistungsfähigen Typsystemen geführt, welche neben einfachen Datentypen auch Einheiten, Datenpräzision, Unsicherheit und Fortpflanzung von Meßdatenfehlern statisch überprüfen können. Der Entwurf entsprechender Typsysteme, Typchecker und Typinferenzverfahren wird in den kommenden Jahren ein wichtiges Forschungsthema für die formalen Methoden bleiben.

Prüfung zusätzlicher Eigenschaften

Formale Methoden werden in der Zukunft nicht nur den Entwurf neuer Arten von Softwaresystemen unterstützen, sondern auch ein breiteres Spektrum an Systemeigenschaften garantieren müssen. Zu diesen Eigenschaften gehören neben dem klassischen Korrektheitsbegriff vor allem Sicherheit und Vertraulichkeit. Aber auch andere Fragestellungen wie Synchronisierbarkeit und Fehlertoleranz verteilter Systeme, Laufzeit und Platzbedarf von Algorithmen und der Verbrauch beschränkter Ressourcen wie Strom oder Zugriff auf mobile Netze werden zunehmend mit formalen Methoden bearbeitet werden müssen.

Seit dem ersten erfolgreichen Angriff [28] auf das bis dahin als absolut sicher geltende Needham-Schroeder Authentifikationsprotokoll [32] ist die formale Analyse von Sicherheits-

protokollen ein wichtiges Thema geworden. Angriffe auf Protokolle nutzen oft implizite Annahmen aus, die bei der Formulierung und dem informalen Korrektheitsbeweis des Protokolls gemacht wurden. Erst die Rigorosität formaler Beweise macht derartige Annahmen sichtbar und fehlschlagende Teilbeweise offenbaren oft unvorhergesehene Szenarien, in denen das Protokoll nicht mehr sicher ist. Die Vielzahl (siehe z.B. die SPORE library <http://www.lsv.ens-cachan.fr/spore>) und die zunehmende Komplexität moderner Sicherheitsprotokolle für direkte und Gruppenkommunikation macht die Entwicklung flexibler automatischer Verifikationswerkzeuge erforderlich, die es ermöglichen, Protokolle, Sicherheitsanforderungen und Angreiferszenarien in einem einheitlichen Formalismus zu präzisieren und zu überprüfen. Dabei sind zwei Teilprobleme zu lösen: die Korrektheit des abstrakten Protokolls relativ zu abstrakten Sicherheitsbedingungen und Angriffsmöglichkeiten sowie die Korrektheit der Implementierung relativ zur abstrakten Spezifikation des Protokolls. Die besondere Herausforderung liegt hierbei in der Vielfalt der Sicherheitsanforderungen (Vertraulichkeit, Integrität, Verfügbarkeit, Authentizität, Zugriffskontrolle, Datenschutz, etc.) und Angreiferszenarien (Hacker, Viren, trojanische Pferde, Man-in-the-Middle Attacken, Denial-of-Service Attacken, Überwachung externer Hardwareeigenschaften, etc.), die es nahezu unmöglich machen, "absolute" Sicherheit zu garantieren. Die Aufgabe formaler Methoden ist damit, herauszuarbeiten, welche externen Anforderungen an Zugriffe auf Systeme gestellt werden müssen, um ein ausreichendes Maß an Sicherheit garantieren zu können.

5 Zusammenfassung

Die ursprüngliche Vision der Künstlichen Intelligenz, Computersysteme zu bauen, welche Alltagswissen berücksichtigen um zu brauchbaren Assistenten für menschliche Benutzer zu werden, wird nach wie vor auch das Leitziel der formalen Methoden bleiben. Vershoben hat sich allerdings der Ausgangspunkt, von dem aus dieses Ziel angegangen wird. An die Stelle eines universellen deduktiven Systems, das versucht, allgemeine Intelligenz zu modellieren, und zur Lösung spezieller Probleme auf eine große Bibliothek von Wissen zurückgreift, werden Synthese- und Verifikationsverfahren treten, die neben einer umfangreichen Wissensbank vor allem auch spezielle Strategien zur Lösung eines Problems einsetzen. Der Trend geht hierbei zum künstlichen IT-Spezialisten oder IT-Assistent, der dem realen Softwarespezialisten bei der Lösung komplexer Probleme zur Seite steht.

Diese Vorgehensweise steht keineswegs im Widerspruch zur Vision der Modellierung allgemeinen intelligenten Verhaltens, sondern entspricht der Tatsache, daß die Bearbeitung anspruchsvoller Aufgaben neben einer hohen Intelligenz auch eine Aufgabe in dem entsprechenden Spezialgebiet erfordert. So wie menschliche Programmierer erst nach jahrelanger Ausbildung gute Softwareprodukte entwickeln können, können formale Methoden erst dann erfolgreich eingesetzt werden, wenn sie eine große Menge von Wissen über verschiedene Anwendungsbereiche und vielfältige Spezialstrategien zur Analyse und Synthese von Software "gelernt" haben und in der Lage sind, mit menschlichen Benutzern zusammenzuarbeiten.

Literatur

- [1] Stuart Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, Evan Moran. Innovations in computational type theory using Nuprl. *Journal of Applied Logic*, 2006.
- [2] Stuart F. Allen. Abstract identifiers, intertextual reference and a computational basis for recordkeeping. *First Monday*, 9(2), 2004.
- [3] Thomas Ball, Shuvendu K. Lahiri, Madanlal Musuvathi. Zap: Automated theorem proving for software analysis. Technical Report TR-2005-137, Microsoft Research, 2005.
- [4] M. Balseer, W. Reif, G. Schellhorn, K. Stenzel. KIV 3.0 for Provably Correct Systems. In *Current Trends in Applied Formal Methods*, LNCS 1641. Springer, 1999.
- [5] M. Barnett, K. Rustan M. Leino, Wolfram Schulte. The Spec# programming system: An overview. In *CASSIS 04: Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*, LNCS 3362, pp. 49–69. Springer, 2005.
- [6] D. Basin, Y. Deville, P. Flener, A. Hamfelt, J. F. Nilsson. Synthesis of Programs in Computational Logic. In *Program Development in Computational Logic*, LNCS 3049, pp. 30–65, Springer, 2004.
- [7] Christoph Benzmüller, et. al. Ω mega: Towards a mathematical assistant. In W. McCune, ed., *14th Conference on Automated Deduction*, LNAI 1249, pp. 252–256. Springer, 1997.
- [8] Wolfgang Bibel. Syntax-directed, semantics-supported program synthesis. *Artificial Intelligence*, 14(3):243–261, October 1980.
- [9] Mark Bickford, Christoph Kreitz, Robbert van Renesse, Robert Constable. An experiment in formal design using meta-properties. In J. Lala, D. Maughan, C. McCollum, B. Witten, eds., *DARPA Information Survivability Conference and Exposition II*, volume II, pp. 100–107. IEEE Computer Society Press, 2001.
- [10] A. Bundy, F. van Harmelen, J. Hesketh, A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.
- [11] E. M. Clarke, O. Grumberg, D. Peled. *Model Checking*. MIT Press, 1999.
- [12] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, Helmut Veith. Progress on the state explosion problem in model checking. In R. Wilhelm, ed., *Informatics: 10 Years Back, 10 Years Ahead*, LNCS 2000, pp. 176–194. Springer, 2000.
- [13] Robert L. Constable, Stuart F. Allen, H. Mark Bromley, W. Rance Cleaveland, J. F. Cremer, Robert W. Harper, Douglas J. Howe, Todd B. Knoblock, Nax Paul Mendler, Prakash Panangaden, Jim T. Sasaki, Scott F. Smith. *Implementing Mathematics with the Nuprl proof development system*. Prentice Hall, 1986.
- [14] A. Franke, M. Kohlhase. MATHWEB, an agent-based communication layer for distributed automated theorem proving. In H. Ganzinger, ed., *16th Conference on Automated Deduction*, LNAI 1632, pp. 217–221, 1999.
- [15] Carla P. Gomes, Douglas R. Smith, Stephen J. Westfold. Synthesis of schedulers for planned shutdowns of power plants. In *Eleventh Knowledge-Based Software Engineering Conference*, pp. 12–20. IEEE Computer Society Press, 1996.
- [16] Cordell C. Green. An application of theorem proving to problem solving. In *IJCAI-69 – 1st International Joint Conference on Artificial Intelligence*, pp. 219–239, 1969.
- [17] The 2020 Schience Group. Towards 2020 science. Technical report, Microsoft Research, 2006.

- [18] Dominik Haneberg, Simon Bäumler, Michael Balsler, Holger Grandy, Frank Ortmeier, Wolfgang Reif, Gerhard Schellhorn, Jonathan Schmitt, Kurt Stenzel. The user interface of the KIV verification system - a system description. In *User Interfaces for Theorem Provers Workshop (UITP 2005)*.
- [19] M. Heisel, W. Reif, W. Stephan. Implementing verification strategies in the KIV system. In E. Lusk, R. Overbeek, eds., *9th Conference on Automated Deduction, LNCS 310*, pp. 131–140. Springer, 1988.
- [20] M. Heisel, W. Reif, W. Stephan. Tactical theorem proving in program verification. In M. E. Stickel, ed., *10th Conference on Automated Deduction, LNCS 449*, pp. 117–131. Springer, 1990.
- [21] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
- [22] G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison Wesley, 2003.
- [23] M. Kerber, M. Kohlhase, V. Sorge. Integrating computer algebra into proof planning. *Journal of Automated Reasoning*, 1998.
- [24] M. Kohlhase. OMDoc: Towards an internet standard for the administration, distribution, and teaching of mathematical knowledge. In *6th International Conference on Artificial Intelligence and Symbolic Computation, LNAI 1930*, 2000.
- [25] F. Koob, M. Ullmann, S. Wittmann, (BSI); D. Hutter, B. Langenstein, C. Sengler, W. Stephan, A. Wolpers, W. Reif. The VSE development method - a way to engineer high-assurance software systems. In *Proceedings 5. GI/ITG-Fachgespräch Formale Beschreibungstechniken für verteilte Systeme*, 1995.
- [26] Christoph Kreitz. Building reliable, high-performance networks with the Nuprl proof development system. *Journal of Functional Programming*, 14(1):21–68, 2004.
- [27] Xiaoming Liu, Christoph Kreitz, Robbert van Renesse, Jason Hickey, Mark Hayden, Kenneth Birman, Robert Constable. Building reliable, high-performance communication systems from components. In *17th ACM Symposium on Operating Systems Principles, Operating Systems Review*, 33:80–92, 1999.
- [28] G. Lowe. An attack on the Needham-Schroeder public-key protocol. *Information Processing Letters*, 56(3):131 – 133, 1995.
- [29] Z. Manna, A. Pnueli. *Temporal Verification of Reactive Systems*. Springer, 1995.
- [30] Zohar Manna, Richard J. Waldinger. Knowledge and reasoning in program synthesis. *Artificial Intelligence*, 6(2):175–208, 1975.
- [31] MoWGLI: Mathematic on the web: Get it by logics and interfaces. <http://mowgli.cs.unibo.it>.
- [32] R. M. Needham, M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, DIGITAL Systems Research Center 1978.
- [33] Tobias Nipkow, Lawrence C. Paulson, Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic, LNCS 2283*, Springer, 2002.
- [34] OMDoc home page. <http://www.mathweb.org/omdoc>.
- [35] OpenMath home page. <http://www.openmath.org>.
- [36] Jens Otten, Thomas Raths, Christoph Kreitz. The ILTP Library: Benchmarking automated theorem provers for intuitionistic logic. In Bernhard Beckert, ed., *International Conference TABLEAUX-2005, LNAI 3702*, pp. 333–337. Springer, 2005.
- [37] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, M. K. Srivas. PVS: Combining specification, proof checking and model checking. In Rajeev Alur, Thomas A. Henzinger, eds., *Computer-Aided Verification, LNCS 1102*, pp. 411–414. Springer, 1996.
- [38] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.
- [39] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
- [40] Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, 1999.
- [41] John Rushby. Automated deduction and formal methods. In Rajeev Alur, Thomas A. Henzinger, eds., *Computer-Aided Verification (CAV '96), LNCS 1102*, pp. 169–183. Springer, 1996.
- [42] John Rushby. Systematic formal verification for fault-tolerant time-triggered algorithms. *IEEE Transactions on Software Engineering*, 25(5):651–660, 1999.
- [43] John Rushby. Harnessing disruptive innovation in formal verification. In *4th IEEE International Conference on Software Engineering and Formal Methods*, 2006.
- [44] Douglas R. Smith. Top-down synthesis of divide-and-conquer algorithms. *Artificial Intelligence*, 27(1):43–96, 1985.
- [45] Douglas R. Smith. KIDS — a knowledge-based software development system. In Michael R. Lowry, Robert D. McCartney, eds., *Automating Software Design*, pp. 483–514. AAAI Press / The MIT Press, 1991.
- [46] Douglas R. Smith, Michael R. Lowry. Algorithm theories and design tactics. *Science of Computer Programming*, 14(2-3):305–321, 1990.
- [47] Douglas R. Smith, Eduardo A. Parra. Transformational approach to transportation scheduling. In *8th Knowledge-Based Software Engineering Conference, 1993*, pp. 60–68, 1993.
- [48] G. Sutcliffe, C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.
- [49] Marc Wagner, Serge Autexier, Christoph Benzmüller. PLATO: A mediator between text-editors and proof assistant systems. In *User Interfaces for Theorem Provers Workshop (UITP 2006)*.
- [50] Richard J. Waldinger, R. T. C. Lee. PROW: A step toward automatic program writing. In *IJCAI-69 – 1st International Joint Conference on Artificial Intelligence*, pp. 241–252. 1969.

Kontakt

Prof. Dr. Christoph Kreitz
 Institut für Informatik
 Universität Potsdam
 August-Bebel Str. 89, 14482 Potsdam
 Tel.: +49 331 977 3060
 Fax : +49 331 977 3042
 Email: kreitz@cs.uni-potsdam.de
 Web: <http://www.cs.uni-potsdam.de/ti/kreitz>



Christoph Kreitz ist Professor für Theoretische Informatik an der Universität Potsdam und Senior Research Associate am Department of Computer Science der Cornell University. Seine Forschungsschwerpunkte sind die Entwicklung von Strategien und Beweis-suchprozeduren zur Steuerung taktischer Beweisassistenten sowie ihre Anwendung für die Verifikation, Synthese und Optimierung von Softwaresystemen.