Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Deduktion: Von der Theorie zur Anwendung

Franz Baader[1], **Bernhard Beckert**[2], **Tobias Nipkow**[3]

[1] Institut für Theoretische Informatik, TU Dresden, baader@tcs.inf.tu-dresden.de
[2] Institut für Theoretische Informatik, Karlsruher Institut für Technologie, beckert@kit.edu
[3] Institut für Informatik, TU München, www.in.tum.de/~nipkow

Received: date / Revised version: date

1 Einleitung

Unter Deduktion versteht man die Herleitung logischer Konsequenzen aus einer gegebenen Menge logischer Axiome. In den Anfängen der formalen Logik bestand die Hoffnung, dass Deduktion auch für ausdrucksstarke Logiken, die alles mathematische Wissen axiomatisieren können, voll automatisierbar ist und damit mathematische Theoreme nicht mehr mühsam vom Menschen bewiesen werden müssen, sondern ein Deduktionsverfahren bei Eingabe des Theorems einfach *entscheidet*, ob dieses aus der Axiomenmenge folgt oder nicht. Erste Unentscheidbarkeitsresultate machten dann aber die Hoffnung auf eine vollständige Automatisierung der Mathematik zunichte. Entsprechendes gilt für die Hoffnung, nicht-triviale semantische Eigenschaften von Programmen (wie Terminierung, Korrektheit bzgl. einer formalen Spezifikation) zu entscheiden oder Künstliche Intelligenz dadurch zu realisieren, dass man alles wichtige Wissen über die Welt axiomatisiert und dann einen „allgemeinen Problemlöser“ einsetzt, der daraus Schlüsse zieht. Für *partiell-entscheidbare Logiken* (wie die wichtige Prädikatenlogik erster Stufe) kann man aber mit einem partiellen Entscheidungsverfahren für *gültige Aussagen* (d. h. Aussagen, die tatsächlich aus der Axiomenmenge folgen) stets in endlicher Zeit feststellen, dass dies der Fall ist. Automatische Theorembeweiser sind im Prinzip solche partiellen Entscheidungsverfahren. Es stellte sich aber zunächst heraus, dass mit diesen nur sehr triviale mathematische Theoreme vollautomatisch bewiesen werden können. Zur Lösung dieses Problems wurden hier drei verschiedene Strategien verfolgt, die sich alle in geeigneten Anwendungsbereichen als erfolgreich herausstellten. Zum einen wurde die Effizienz von automatischen Beweisern durch Verbesserung der zugrunde liegenden Theorie, aber auch durch ausgefeilte Implementierungstechniken, erheblich verbessert. Zum anderen ging man bei der Interaktiven Deduktion dazu über, den Menschen in den Beweisprozess mit einzubeziehen. Schließlich suchte man, für auf spezielle Anwendungsprobleme angepasste ausdrucksschwächere Logiken, nach effizienten Entscheidungsverfahren. Im folgenden gehen wir auf Erfolge dieser drei Ansätze exemplarisch ein.

2 Interaktive Deduktion

Interaktive Deduktion ist die Einbeziehung des Menschen in den Beweisprozess. Der Benutzer steuert den globalen Beweisprozess, die automatischen Beweiser aller Art erledigen die lokalen Einzelschritte. Dies hat sich zu einem eigenen Forschungszweig entwickelt, der sehr von den dort entwickelten und benutzten Systemen, interaktiven Theorembeweisern (*ITPs*), geprägt ist. Deren Hauptvertreter sind ACL2 [22], Coq [7], HOL4 [37], HOL Light [15], Isabelle [30] und PVS [31].

Die Entwicklung der ITPs hat seit den Anfängen in den 1970er Jahren erstaunliche Fortschritte gemacht. Statt 10-zeiliger Programme werden inzwischen 10.000-zeilige verifiziert, statt mathematischer Trivialitäten werden inzwischen tiefe mathematische Theorien formalisiert. Geschichtlich betrachtet handelt es sich um die erfolgreiche Realisierung des Programms von Hilbert und Russel, nämlich der Beschreibung aller Mathematik (und Informatik!) in Logik. Etwas pathetisch könnte man sagen, dass es darum geht die DNA der Mathematik und Informatik zu formalisieren. Allerdings funktioniert dies bisher, im Gegensatz zur biologischen DNA, nur interaktiv.

Der Mensch ist kognitiv nicht in der Lage, mit großen Formeln effektiv umzugehen. Daher benutzen ITPs ausdrucksstarke Logiken, mit denen sich komplexe mathematische Zusammenhänge abstrakt beschreiben und beweisen lassen. Herausragende Beispiele sind HOL (Higher Order Logic) und Typentheorie. In beiden Fällen

kann man aus einer Informatik-Perspektive die Logiken als Erweiterungen einer funktionalen Programmiersprache (in der Tradition von ML und Haskell) mit Quantoren betrachten. Es ist diese Kombination von (funktionaler!) Programmierung und Logik in einer Sprache, die die Attraktivität und Ausdrucksstärke des Ansatzes ausmacht. Das heißt, die Theorie der Programmiersprachen ist neben der Logik eine der wichtigsten Grundlagen für ITPs.

Eine Herausforderung für die Logik in der Informatik ist das Bereitstellen (halb)automatischer Verfahren, d. h. von (partiellen) Entscheidungsprozeduren, um für den Menschen bzw. Experten offensichtliche logische Schlüsse zu generieren oder zu überprüfen. Hier konnte man in den Anfängen der Interaktiven Deduktion auf existierende theoretische Grundlagen zurückgreifen, insbesondere aus den Bereichen Termersetzung und Unifikation [4] und automatisches Beweisen in Aussagen- und Prädikatenlogik (siehe Abschnitt 3). Mit der Anwendungsnähe stieg aber auch der Anspruch an die Automatisierung. Inzwischen wird oft auf in der Mathematik bereits bekannte Verfahren zurückgegriffen, etwa im Bereich reelle Analysis. Ein beeindruckendes Beispiel ist eine beweisgenerierende Entscheidungsprozedur für Arithmetik über reellen Zahlen [25]. Dies ist „Angewandte Logik“ im besten Sinne. Und die logische Komponente ist der innovative Mehrwert. Denn die meisten ITPs akzeptieren nicht einfach ein Ja einer Entscheidungsprozedur sondern insistieren auf einem Beweis oder auf einer Verifikation der Entscheidungsprozedur (im ITP selbst). Aus gutem Grund, denn Implementierungen von Entscheidungsprozeduren sind genauso fehlerhaft wie alle unverifizierte Software. Da ITPs aber zur Verifikation eingesetzt werden, sollten sie selbst so zuverlässig wie möglich sein. Diese Zuverlässigkeit ist eine der herausragenden Merkmale vieler ITPs.

Die Mächtigkeit von ITPs kann man am besten durch erfolgreiche Anwendungen verdeutlichen. Daher beschreiben wir im folgenden einige *Meilensteine der Anwendung.*

Ein verifizierter C Compiler. Schon McCarthy und Painter hatten 1967 von einem verifizierten Compiler geträumt und auch erste bescheidene Beweisschritte auf dem Papier gemacht. 40 Jahre später hat Leroy im Alleingang einen C Compiler mit Coq verifiziert. Der Compiler ist in Coqs funktionaler Programmiersprache geschrieben und umfasst 5.000 Zeilen, der Beweis weitere 30.000 Zeilen. Die Qualität des generierten Codes ist mit gcc auf Optimierungsstufen 1 und 2 vergleichbar. Der Arbeitsaufwand war etwa 2 Personenjahre. Der Erfolg des Projekts ist zu einem guten Teil das Resultat von Leroys genauer Kenntnis der Theorie von Programmiersprachen und Compilern, die es ihm erlaubte, den Compiler in 7 Phasen zu unterteilen, und so den Beweis in 7 überschaubare Teile zu zerlegen. Der Erfolg des Projekts erfreut den Autor dieser Zeilen auch deshalb, weil ihm noch vor 15 Jahren von einem deutschen Kollegen erklärt wurde, dass man schon alleine die Semantik einer realen Programmiersprache nie in einem ITP definieren könne.

Ein verifizierter Betriebssystemkern. Im *L4.verified* Projekt unter der Leitung von Klein beim australischen Forschungsinstitut NICTA wurde erstmalig die funktionale Korrektheit eines kommerziell anwendbaren Betriebssystemkerns verifiziert [23], mit Hilfe des Isabelle Systems. Der Beweis zeigt, dass der C-Code des seL4 Mikrokerns (secure embedded L4) seiner formalen Spezifikation entspricht. Der seL4-Kern umfasst 8.700 Zeilen hardware-nahen C-Code [41] und 600 Zeilen ARM-Assembler-Code. Bei der Verifikation wurde besonders darauf geachtet, die in Mikrokernen so wichtige Effizienz des Message-Passing im Kern nicht zu beeinträchtigen: Nachrichten in seL4 sind schneller als in vielen verfügbaren L4-Kernen auf ARM11 und mit nur 224 Prozessorzyklen gleichauf mit den schnellsten veröffentlichten Implementierungen auf dieser Platform.

Der Beweis verlief über drei Ebenen: eine abstrakte Spezifikation direkt in Isabelle, eine ausführbare Spezifikation in Isabelle, die aus einem Haskell-Prototypen des Kerns generiert wurde, und schliesslich der C-Code. Die Verifikation der C-Ebene benutzte eine in Isabelle formalisierte C-nahe Programmiersprache, in die C automatisch übersetzt wird. Der Beweis umfasst 200.000 Zeilen.

Neben funktionaler Korrektheit impliziert der Beweis auch eine ganze Reihe von Sicherheitseigenschaften: der C-Code des seL4-Kerns beinhaltet weder Buffer-Overflows, noch Null-Pointer-Dereferenzierungen, noch Speicherfehler. Alle Systemaufrufe terminieren, es gibt keine Speicher-Lecks und im Systemmodus wird nur Systemcode ausgeführt.

Auch im BMBF-Projekt Verisoft [1] sind umfangreiche Arbeiten zur Betriebssystemverifikation entstanden, ebenfalls in Isabelle. Hierbei lag der Schwerpunkt auf einer durchgängigen Verifikation eines Prozessors, eines Betriebssystems und eines Compilers.

Der Vier-Farben-Satz. Der erste Beweis des Vier-Farben-Satzes („Jede Landkarte kann mit vier Farben so eingefärbt werde, dass keine zwei benachbarten Länder die selbe Farbe haben“) von Appel und Haken [2] wurde kritisiert, weil er ein Assembler-Programm benutzte, um die enorme Zahl der Fälle zu bewältigen, die kein Mensch je von Hand überprüfen könnte. War dies noch ein Beweis? Insbesondere, da die Korrektheit des Programms nicht verifiziert worden war. Dreißig Jahre später formalisierte Gonthier [12] einen 60.000-Zeilen-Beweis in Coq, der sich auf neuere Arbeiten abstützte, aber weiterhin eine riesige Fallunterscheidung enthielt. Allerdings mit dem feinen Unterschied, dass nun die Programme, die die Fälle überprüften, ebenfalls Teil der Verifikation waren. Der Vier-Farben-Satz war damit endgültig bewie-

sen, auch wenn der Beweis weiterhin nicht den ästhetischen Kriterien der traditionellen Mathematik entspricht.

Die Keplersche Vermutung. Als Hales die 400 Jahre alte Keplersche Vermutung bewies [13], dass die dichteste Packung von Kugeln gleichen Durchmessers die vom Obsthändler her bekannte Pyramide ist, erging es ihm wie Appel und Haken, denn auch sein Beweis stützte sich auf umfangreiche unverifizierte Programme. Daher hat Hales eine informelle Initiative mit dem Namen *Flyspeck* ins Leben gerufen, um den Beweis formal zu verifizieren [14]. Dies ist für einige der verwendeten Programme gelungen (z. B. [29]), aber an der vollständigen Verifikation des Beweises wird derzeit noch gearbeitet.

3 Automatische Deduktion

Die jahrzehntelange intensive Forschung im Bereich der automatischen Deduktion hat einige der elegantesten und durchdachtesten Technologien der Informatik geliefert. Dadurch wurden automatische Deduktionssysteme immer leistungsfähiger, sowohl was die Größe der handhabbaren Einzelprobleme angeht als auch die Anzahl der Probleme, die in vorgegebener Zeit gelöst werden können. Anders als in der Frühzeit des automatischen Beweisens angenommen, ist dieses heute vor allem in Anwendungsbereichen erfolgreich, in denen viele kleine Einzelbeweise anfallen, die ein Mensch in ihrer Vielzahl und ihrem Zusammenspiel nicht mehr überschauen kann. Der vollautomatische Beweis einzelner Theoreme, für die zuvor trotz intensiver Bemühungen von Mathematikern kein Beweis gefunden werden konnte, gelingt dagegen selten. Eine Ausnahme war der im Jahr 1996 automatisch gefundene Beweis der lange unbewiesenen Vermutung, dass alle Robbins-Algebren auch Boolesche Algebren sind [9]. Dieser bemerkenswerte Erfolg, den William McCune mit dem automatischen Beweissystem EQP erzielte, blieb ein Einzelfall. Erfolgreich und routinemäßig angewendet werden automatische Deduktionssysteme dagegen in Bereichen, in denen Beweise eine große Zahl von Lemmata benötigen, die in jedem konkreten Fall verschieden sind und die zwar jedes für sich einfach zu beweisen sind, aber wegen ihrer Anzahl den Menschen überfordern. Loveland [24] nennt solche Beweise „shallow and messy“. Solche Beweisprobleme treten in der Software- und Hardwareverifikation auf, bei der Programmsynthese, im Bereich Wissensrepräsentation (s. Abschnitt 4) sowie auch immer dann, wenn automatische Beweissysteme eingesetzt werden, um kleinere Teilprobleme zu lösen, die bei der interaktiven Deduktion anfallen (s. Abschnitt 2).

In den letzten Jahren gab es beachtliche theoretische Fortschritte im automatischen Beweisen, wobei hier auch das in den 90er Jahren von der DFG geförderte Schwerpunktprogramm *Deduktion* bis heute nachwirkt. Entscheidend waren zudem aber auch die Fortschritte in der ingenieurmäßigen Umsetzung der Theorie, in der effizienten Implementierung der Kalküle. Und nicht zuletzt spielt in der Praxis die Leistungszunahme der Rechner, auf denen Beweissysteme laufen, eine große Rolle.

Eine wichtige Triebfeder für die Entwicklung neuer Implementierungstechniken ist die CADE ATP System Competition (CASC) [38]. Diese „Weltmeisterschaft“ der automatischen Deduktion fand 2009 zum vierzehnten Mal statt. Automatische Deduktionssysteme messen sich dabei in verschiedenen Kategorien, wobei sich die Problemklassen zum Teil nur syntaktisch unterscheiden, zum Teil in der Ausdrucksstärke der verwendeten Formeln (z. B. mit/ohne Gleichheit) und zum Teil in den semantischen Eigenschaften der Probleme (so gibt es beispielsweise eine Kategorie, in der für *nicht* beweisbarer Aussagen ein Gegenbeispiel zu generieren ist). Der direkte Vergleich von Beweissystemen anhand der selben Beweisaufgaben (für CASC wird die TPTP-Problembibliothek [39] verwendet) macht es einfach, zu beurteilen, welche theoretischen Ansätze und welche Implementierungstechniken für die unterschiedlichen Problemklassen erfolgreich sind. Im Jahr 2009 nahmen mehr als 20 Systeme an CASC teil. Aktueller Sieger in der wichtigsten Kategorie der prädikatenlogischen Probleme ist Vampire 11.0 [34]; dieses System verwendet eine Variante des Resolutionskalküls (mit Superposition zur Gleichheitsbehandlung). Den zweiten Platz belegte der Beweiser „E“ in der Version 1.1pre [36]. Stärkstes System in der Kategorie, in der Gegenbeispiele zu generieren sind, ist Paradox 3.0 [8]. In der Kategorie des Gleichheitsschließens ist das System Waldmeister [16] seit vielen Jahren ungeschlagen. Seit 2008 trägt CASC einer Anforderung Rechnung, die sich in vielen Anwendungen des automatischen Beweisens ergibt, indem eine Kategorie eingeführt wurde, in der Probleme zu lösen sind, die aus einer sehr großen Axiomenmenge bestehen, aus der zudem mehrere Theoreme abzuleiten sind. Bei der Programmverifikation treten solche Probleme beispielsweise auf, wenn eine Axiomenmenge gegeben ist, die die Semantik der Programmiersprache formalisiert, und viele Aussagen über das Verhalten kleiner Teilprogramme daraus abzuleiten sind. Auch in dieser neuen CASC-Kategorie hatte 2009 das System Vampire 11.0 die Nase vorn.

Eine weitere Anforderung, die sich aus den aktuellen Anwendungsbereichen ergibt, ist die Fähigkeit mit speziellen Theorien effizient umzugehen. Dazu gehören neben der Gleichheitstheorie beispielsweise ganzzahlige und rationale Arithmetik und auch Axiomatisierungen von Datenstrukturen wie Listen und Arrays. Dies hat in den letzten Jahren zur Entwicklung sehr leistungsfähiger Systeme geführt, die sogenannte SMT-Probleme entscheiden können (SMT steht für „Satisfiability Modulo Theories“). Beispiele dafür sind Z3 [10], CVC3 [6] und Yices [11]. Die verwendeten Techniken haben ihre Wurzeln in Kalkülen für aussagenlogische Probleme (d.h. SAT Probleme). SMT-Systeme vereinigen diese Kalküle mit Entscheidungsverfahren für verschiedene Theorien

und Konzepten aus der prädikatenlogischen Deduktion. Auch für SMT-Probleme wird inzwischen regelmäßig ein Wettbewerb veranstaltet. Dieser verwendet die Problemsammlung SMT-LIB [33].

Der Wunsch, die theoretischen Ergebnisse im Gebiet des automatischen Beweisens ingenieurmäßig in praktische Erfolge umzusetzen, ist auch Motor einer weiteren aktuellen Entwicklung. Automatische Beweissysteme integrieren heutzutage viele verschiedene Techniken und Kalküle, um alle Anforderungen einer Anwendung abdecken zu können: aussagenlogische und prädikatenlogische Kalküle, Entscheidungsverfahren für Theorien, Termersetzungsverfahren zur Gleichheitsbehandlung, Induktionsverfahren, Gegenbeispielgenerierung, um nur die wichtigsten zu nennen.

Die automatische Deduktion ist zu einer Erfolgsgeschichte der Theoretischen Informatik geworden. Über Jahrzehnte entwickelte und verfeinerte Techniken und Kalküle werden ingenieurmäßig umgesetzt und halten Einzug in praktische Anwendungsgebiete.

4 Logikbasierte Wissensrepräsentation

Intelligentes Verhalten hängt wesentlich ab von dem vorhandenen Wissen über die Umwelt, in der dieses Verhalten stattfindet, sowie der Fähigkeit, Schlußfolgerungen aus diesem Wissen zu ziehen. Daher sind Wissensrepräsentation und Inferenz ("knowledge representation and reasoning") seit den Anfängen der Künstlichen Intelligenz (KI) wichtige Teilbereiche dieses Gebietes. Stärker umstritten war aber die Frage, welche Formalismen für die Repräsentation von Wissen geeignet sind und welche Schlußfolgerungsverfahren zum Einsatz kommen sollen. Während frühe Verfechter einer logikbasierten Wissensrepräsentation (wie Newell und McCarthy) zunächst auf einen einheitlichen, sehr ausdrucksstarken Formalismus (wie Prädikatenlogik erster Stufe) setzten, für den Inferenz mittels eines allgemeinen Problemlösers ("general problem solver") realisiert werden sollte, bezweifelten andere (wie Minsky) überhaupt, dass formale Logik für die Repräsentation von Wissen geeignet sei und dass Deduktion (d. h. logisches Schließen) angemessene Inferenzverfahren liefern könne. Wie so oft im Leben hatten beide Seiten sowohl Recht als auch Unrecht.

Die Gegner der logikbasierten Wissensrepräsentation schlugen alternative Repräsentationsansätze wie Semantische Netze [32] und Frames [27] vor, sowie eher syntaxbasierte Inferenzverfahren wie "Spreading Activation" und "Matching". Das Hauptproblem dieser Ansätze war aber das Fehlen einer formalen Semantik. Bei Semantischen Netzen wurde argumentiert, dass deren Bedeutung wegen ihrer graphischen Darstellung intuitiv klar sei. Es stellte sich aber schnell heraus, dass der Wissensingenieur, der ein konkretes Netwerk entwirft, eine ganz andere Intuition haben kann als der Implementierer von Inferenzalgorithmen, und dass deren Verständnis nicht mit dem des Benutzers übereinstimmen muss, der deshalb möglicherweise weder die Wissensbasis selbst noch die Ausgaben des Inferenzsystems richtig interpretieren kann. (Dieser "clash of intuition" tritt bei vielen angeblich intuitiven graphischen Formalismen ohne formale Semantik auf, wie z. B. auch bei den im Bereich des Semantic Web derzeit auf praktischer Seite so beliebten "leight-weight ontologies".)

Bei der logikbasierten Wissensrepräsentation hatte man aber zunächst das Problem, dass Prädikatenlogik erster Stufe einerseits zu ausdrucksstark und andererseits zu ausdrucksschwach war. Die hohe Ausdrucksstärke führt dazu, dass wichtige Inferenzprobleme (wie: Ist die Wissensbasis inkonsistent? Folgt Faktum α aus der Wissensbasis Γ?) unentscheidbar sind. Natürlich kann man trotzdem versuchen, automatische Theorembeweiser für die Prädikatenlogik erster Stufe als Inferenzmaschinen in der Wissensrepräsentation einzusetzen, aber diese erreichen trotz der in Abschnitt 3 erwähnten Fortschritte in diesem Bereich auch heute noch nicht immer die für Anwendungen in der Wissensrepräsentation nötigen kurzen Antwortzeiten. Andererseits ist Prädikatenlogik erster Stufe teilweise zu ausdrucksschwach: sie unterstützt die Darstellung vagen, unvollständigen, subjektiven, oder zeitabhängigen Wissens nicht, was aber für die Repräsentation von Alltagswissen ("commonsense knowledge") wichtig wäre. Aus diesen Gründen waren frühe logikbasierte Wissensrepräsentationssysteme in der Praxis häufig entweder zu langsam oder zu ausdrucksschwach.

Diese Situation hat sich aber inzwischen grundlegend verändert, zum einen natürlich durch die wesentlich gesteigerte Kapazität moderner Computer, zum anderen aber auch durch die intensive Forschung der letzten 20 Jahre in der logikbasierten Wissensrepräsentation [26], sowohl auf theoretischer als auch auf praktischer Seite. Anstatt nach dem allgemeinen Problemlöser zu suchen, hat man sich auf handhabbarere Teilprobleme konzentriert und versucht, auf diese Probleme zugeschnitte Logiken mit möglichst effizienten Inferenzverfahren zu entwickeln. Obwohl in der modernen logikbasierte Wissensrepräsentation automatische Theorembeweiser für Prädikatenlogik erster Stufe meist nicht mehr direkt Verwendung finden, werden dort aber eine Vielzahl der im Bereich der Deduktion entwickelten Methoden beim Entwurf spezialisierter Inferenzverfahren benutzt. Sehr erfolgreich wurde dieser Ansatz z. B. in den Bereichen Aktionsformalismen, nichtmonotone Logiken, Modal- und Temporallogiken, Beschreibungslogiken, und Planungsformalismen und -verfahren eingesetzt (siehe z. B. die entsprechenden Kapitel in [26] und in [40]). Im folgenden stellen wir dies exemplarisch für den Bereich der Beschreibungslogiken dar.

Beschreibungslogiken [3] sind eine sehr gut untersuchte Familie logikbasierter Wissensrepräsentationsformalismen zur Darstellung terminologischen Wissens, d. h. Formalismen, mit deren Hilfe die in einem Anwendungs-

gebiet wichtigen Begriffe definiert und darüber Schlussfolgerungen gezogen werden können. Sie haben in verschiedenen Anwendungsbereichen (wie Sprachverarbeitung, Konfiguration, Datenbanken) Verwendung gefunden. Der bisher größte Anwendungserfolg des Gebiets ist aber, dass der von der Ontologiearbeitsgruppe des WWW-Konsortiums entwickelte und inzwischen als Standard akzeptierte Vorschlag für eine Ontologiesprache für das Semantische Web, OWL,[1] im wesentlichen auf einer ausdrucksstarken Beschreibungslogik beruht. In Beschreibungslogiken werden Konzepte durch Konzeptbeschreibungen formalisiert, welche aus Konzeptnamen (einstelligen Prädikaten) und Rollennamen (zweistelligen Prädikaten) unter Verwendung von Konzeptkonstruktoren aufgebaut werden. Die Ausdrucksstärke einer Beschreibungslogik wird durch die verfügbaren Konstruktoren bestimmt. Aus semantischer Sicht werden Konzeptnamen und Konzeptbeschreibungen als Mengen von Individuen und Rollennamen als binäre Relationen auf Individuen interpretiert. So können wir z. B. unter Verwendung des Konzeptnamens *Frau* und des Rollennamens *hat_kind* das Konzept der *Frauen, die eine Tochter haben*, durch die Beschreibung $\mathit{Frau} \sqcap \exists \mathit{hat\_kind}.\mathit{Frau}$ ausdrücken[2] und das Konzept der *Frauen, die nur Töchter haben* durch $\mathit{Frau} \sqcap \forall \mathit{hat\_kind}.\mathit{Frau}$. Die einfachste Form einer Beschreibungslogikterminologie (TBox) ermöglicht es, Abkürzungen für komplexe Konzeptbeschreibungen einzuführen. Die TBox

$$\begin{aligned} \mathit{Frau} &\equiv \mathit{Mensch} \sqcap \mathit{Weiblich}, \\ \mathit{Mutter} &\equiv \mathit{Frau} \sqcap \exists \mathit{hat\_kind}.\top \end{aligned}$$

definiert das Konzept *Frau* als einen weiblichen Menschen und das Konzept *Mutter* als eine Frau, die ein Kind hat (wobei $\top$ für das universelle Konzept steht, das als Menge aller Individuen interpretiert wird). Sogenannte allgemeine Inklusionsaxiome (GCIs für "general concept inclusions") können dazu verwendet werde, die Interpretation von Konzepten zusätzlich einzuschränken. Insbesondere kann man damit Definitions- und Wertebereichsbeschränkungen für Rollen ausdrücken, wie z. B.

$$\begin{aligned} \exists \mathit{hat\_kind}.\mathit{Mensch} &\sqsubseteq \mathit{Mensch}, \\ \mathit{Mensch} &\sqsubseteq \forall \mathit{hat\_kind}.\mathit{Mensch}, \end{aligned}$$

welche aussagen, dass nur Menschen Menschen als Kinder haben können und dass Kinder von Menschen stets Menschen sind. Beschreibungslogiksysteme stellen ihren Benutzern Inferenzdienste zur Verfügung, die es erlauben, implizites Wissen aus explizit vorhandenem Wissen abzuleiten. Ein für Beschreibungslogiken besonders wichtiges Inferenzproblem ist das Subsumtionsproblem, bei dem es darum geht, implizite Unterkonzeptbeziehungen aufzudecken. Aus den Konzeptdefinitionen und GCIs unseres Beispiels folgt z. B. die Subsumtionsbeziehung $\mathit{Mutter} \sqsubseteq \exists \mathit{hat\_kind}.\mathit{Mensch}$, d. h. wir können folgern, dass jede Mutter einen Menschen als Kind hat, obwohl dies in der Definition von Mutter nicht explizit gefordert wird. Um eine vorhersehbares Verhalten des Systems mit kurzen Antwortzeiten zu erhalten, sollten das Subsumtions- und andere Inferenzprobleme entscheidbar und vorzugsweise von niedriger Komplexität sein. Dabei hat sich aber die Einschätzung, was hier eine akzeptable Komplexität darstellt, im Laufe der letzten 20 Jahre dramatisch verändert. Während in den achtziger Jahren zunächst polynomielle Komplexität gefordert wurde, zeigten erste Komplexitätsresultate zu Beginn der neunziger Jahre, dass dies für die damals verwendeten Beschreibungslogiken (die alle über Konjunktion $\sqcap$ und Werterestriktion $\forall r.C$ verfügten) nicht möglich war [28]. Zur gleichen Zeit wurde von Schmidt-Schauß and Smolka [35] ein neuer Ansatz zur Entscheidung von Inferenzproblemen in Beschreibungslogiken eingeführt, der auf den in der Deduktion entwickelten Tableau-Verfahren beruhte. Es stelle sich heraus, dass dieser Ansatz auf eine Vielzahl von Beschreibungslogiken anwendbar ist [5] und auch für sehr ausdrucksstarke Beschreibungslogiken noch Entscheidungsverfahren liefert [19,18]. Obwohl die Worst-case-Komplexität dieser Algorithmen sehr hoch ist, liefert der Tableau-Ansatz trotzdem oft praktikable Inferenzverfahren: hochoptimierte Implementierungen dieser Verfahren, z. B. in den Systemen FacT[3] und Racer,[4] haben in realistischen Anwendungen meist sehr gute Antwortzeiten. Viele der verwendeten Optimierungen [17] gehen dabei auf in der Deduktion entwickelte Optimierungsansätze für automatische Theorembeweiser zurück. Entscheidungsverfahren für Beschreibungslogiken können auch durch Anpassung von Resolutionsverfahren erhalten werden [21,20], die so erzeugten Inferenzverfahren für das Subsumtionsproblem sind aber derzeit nicht konkurrenzfähig zu den tableau-basierten Verfahren.

Neben der Tatsache, dass Beschreibungslogiken über eine wohlverstandene formale Semantik verfügen, war die Verfügbarkeit von ausgereiften Systemen, die praktikable Entscheidungsverfahren für sehr ausdrucksstarke Repräsentationsformalismen zur Verfügung stellen, ein wichtiges Argument für die Wahl von Beschreibungslogiken als formale Grundlage der Standards für die "Web Ontology Language" OWL. In der Deduktion entwickelte Methoden haben erheblich zu diesem Erfolg beigetragen.

[1] http://www.w3.org/TR/owl-features/

[2] In Prädikatenlogik erster Stufe könnte man diese Beschreibung durch die Formel $\mathit{Frau}(x) \wedge \exists y.\mathit{hat\_kind}(x,y) \wedge \mathit{Frau}(y)$ mit einer freien Variablen x ausdrücken.

Literatur

1. E. Alkassar, M. Hillebrand, D. Leinenbach, N. Schirmer, and A. Starostin. The Verisoft approach to systems verification. In N. Shankar and J. Woodcock, editors, *Verified Software:*

[3] http://owl.cs.manchester.ac.uk/fact++/

[4] http://www.racer-systems.com/

Theories, Tools, Experiments (VSTTE 2008), volume 5295 of *LNCS*, pages 209–224. Springer, 2008.
2. K. Appel and W. Haken. Every planar map is four colorable. *Bulletin of the AMS*, 82:711–712, 1976.
3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
4. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
5. F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
6. C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *Proceedings, 19th International Conference on Computer Aided Verification (CAV '07)*, LNCS 4590, pages 298–302. Springer, 2007.
7. Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development*. Springer, 2004.
8. K. Claessen and N. Sörensson. New techniques that improve MACE-style finite model finding. In P. Baumgartner and C. Fermüller, editors, *Proceedings of the CADE-19 Workshop: Model Computation – Principles, Algorithms, Applications (Miami, USA)*, 2003.
9. B. I. Dahn. Robbins algebras are boolean: A revision of mccune's computer-generated solution of Robbins's problem. *Journal of Algebra*, 208(2):526–532, 1998.
10. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of the 14th International Conference, Budapest, Hungary*, LNCS 4963, pages 337–340. Springer, 2008.
11. B. Dutertre. System description: Yices 1.0.10. In *SMT-COMP'07*, 2007.
12. G. Gonthier. Formal proof—the four-color theorem. *Notices of the AMS*, 55:1382–1393, 2008.
13. T. C. Hales. Cannonballs and honeycombs. *Notices of the AMS*, 47:440–449, 2000.
14. T. C. Hales, J. Harrison, S. McLaughlin, T. Nipkow, S. Obua, and R. Zumkeller. A revision of the proof of the Kepler conjecture. *Discrete and Computational Geometry*. Published online 2009.
15. J. Harrison. HOL Light: An overview. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2009)*, volume 5674 of *LNCS*, pages 60–66. Springer, 2009.
16. T. Hillenbrand. Citius altius fortius: Lessons learned from the theorem prover WALDMEISTER. *Electr. Notes Theor. Comput. Sci.*, 86(1), 2003.
17. I. Horrocks. Implementation and optimization techniques. In *[3]*, pages 306–346. 2003.
18. I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible $\mathcal{SROIQ}$. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67, Lake District, UK, 2006. AAAI Press/The MIT Press.
19. I. Horrocks and U. Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, Edinburgh (UK), 2005. Morgan Kaufmann.
20. U. Hustadt, B. Motik, and U. Sattler. Reducing SHIQ-description logic to disjunctive datalog programs. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *Proc. of the 9th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 152–162. Morgan Kaufmann, 2004.
21. U. Hustadt and R. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 67–71. Springer, 2000.
22. M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.
23. G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal verification of an OS kernel. In *Proc. 22nd ACM Symposium on Operating Systems Principles (SOSP)*, pages 207–220. ACM, 2009.
24. D. E. Loveland. Automated deduction: Achievements and future directions. *Communications of the ACM*, 43(11), 2000.
25. S. McLaughlin and J. Harrison. A proof-producing decision procedure for real arithmetic. In R. Nieuwenhuis, editor, *Automated Deduction (CADE-20)*, volume 3632 of *LNCS*, pages 295–314. Springer, 2005.
26. J. Minker, editor. *Logic-Based Artificial Intelligence*. Kluwer Academic Publisher, 2000.
27. M. Minsky. A framework for representing knowledge. In J. Haugeland, editor, *Mind Design*. The MIT Press, 1981. A longer version appeared in *The Psychology of Computer Vision* (1975).
28. B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.
29. T. Nipkow, G. Bauer, and P. Schultz. Flyspeck I: Tame graphs. In U. Furbach and N. Shankar, editors, *Automated Reasoning (IJCAR 2006)*, volume 4130 of *LNCS*, pages 21–35. Springer, 2006.
30. T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
31. S. Owre and N. Shankar. A brief overview of PVS. In A. Mohamed, Muñoz, and Tahar, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2008)*, volume 5170 of *LNCS*, pages 22–27. Springer, 2008.
32. M. R. Quillian. Semantic memory. In M. Minsky, editor, *Semantic Information Processing*, pages 216–270. The MIT Press, 1968.
33. S. Ranise and C. Tinelli. The SMT-LIB standard: Version 1.2. Technical report, Department of Computer Science, The University of Iowa, 2006.
34. A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2-3):91–110, 2002.
35. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
36. S. Schulz. System description: E 0.81. In D. A. Basin and M. Rusinowitch, editors, *Automated Reasoning – Second International Joint Conference, IJCAR 2004, Cork, Ireland, July 4-8, 2004, Proceedings*, LNCS 3097, pages 223–228. Springer, 2004.
37. K. Slind and M. Norrish. A brief overview of HOL4. In A. Mohamed, Muñoz, and Tahar, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2008)*, volume 5170 of *LNCS*, pages 28–32. Springer, 2008.
38. G. Sutcliffe. The 4th IJCAR automated theorem proving system competition - CASC-J4. *AI Commun.*, 22(1):59–72, 2009.
39. G. Sutcliffe and C. Suttner. The TPTP problem library for automated theorem proving. At `http://www.cs.miami.edu/~tptp/`.
40. F. van Harmelen, V. Lifschitz, and B. Porter, editors. *Handbook of Knowledge Representation (Foundations of Artificial Intelligence)*. Elsevier Science, 2007.
41. S. Winwood, G. Klein, T. Sewell, J. Andronick, D. Cock, and M. Norrish. Mind the gap: A verification framework for low-level C. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5674 of *LNCS*, pages 500–515. Springer, 2009.