Die Beweisentwicklungsumgebung Ω–MKRP*

Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis
Dan Nesmith, Jörn Richts, Jörg Siekmann
Fachbereich Informatik, Universität des Saarlandes
D-66041 Saarbrücken
{huang|kerber|kohlhase|melis|nesmith|richts}@cs.uni-sb.de
siekmann@dfki.uni-sb.de

Zusammenfassung

Die Beweisentwicklungsumgebung Ω–MKRP soll Mathematiker bei einer ihrer Haupttätigkeiten, nämlich dem Beweisen mathematischer Theoreme unterstützen. Diese Unterstützung muß so komfortabel sein, daß die Beweise mit vertretbarem Aufwand formal durchgeführt werden können und daß die Korrektheit der so erzeugten Beweise durch das System sichergestellt wird. Ein solches System wird sich nur dann wirklich durchsetzen, wenn die rechnergestützte Suche nach formalen Beweisen weniger aufwendig und leichter ist, als ohne das System.

Um dies zu erreichen, ergeben sich verschiedene Anforderungen an eine solche Entwicklungsumgebung, die wir im einzelnen beschreiben. Diese betreffen insbesondere die Ausdruckskraft der verwendeten Objektsprache, die Möglichkeit, abstrakt über Beweispläne zu reden, die am Menschen orientierte Präsentation der gefundenen Beweise, aber auch die effiziente Unterstützung beim Füllen von Beweislücken. Das im folgenden vorgestellte Ω–MKRP-System ist eine Synthese der Ansätze des vollautomatischen, des interaktiven und des planbasierten Beweisens und versucht erstmalig die Ergebnisse dieser drei Forschungsrichtungen in einem System zu vereinigen. Dieser Artikel soll eine Übersicht über unsere Arbeit an diesem System geben.

1 Einführung

Die klassische Aufgabe eines rechnergestützten Beweissystems ist der Nachweis, ob eine bestimmte Formel, das Theorem, aus einer gegebenen Formelmenge, der Axio-

*Das Ω–MKRP-System ist im Teilprojekt D2 des Sonderforschungsbereich 314 „Künstliche Intelligenz – Wissensbasierte Systeme“ entstanden. Informationen über das Projekt kann man auch über das World Wide Web erhalten, URL: `http://js-sfbsun.cs.uni-sb.de/pub/www/`

menmenge (der Wissensbasis), logisch folgt. Diese Frage ist im allgemeinen unentscheidbar, jedoch wenn das Theorem gilt, dann kann man es in endlich vielen Schritten beweisen. In den letzten dreißig Jahren wurden mächtige Kalküle wie Resolution und Paramodulation entwickelt und verfeinert, mit denen in vielen für die Praxis relevanten Spezialfällen positive Ergebnisse erzielt werden können.

Das Hauptinteresse im Gebiet des automatischen Beweisens war lange Zeit – und ist heute auch weitgehend noch – von der Motivation geprägt, effiziente *vollautomatische* Systeme zu entwickeln. In diesem Paradigma sind verschiedene automatische Beweiser entstanden, wie das äußerst effiziente OTTER-System vom Argonne National Lab, USA, und unser eigenes MKRP-System, ein leistungsfähiger Resolutionsbeweiser für eine sortierte Logik erster Stufe mit eingebauter Gleichheit. Über einen Zeitraum von fast fünfzehn Jahren haben wir Stärken und Schwächen des MKRP-Systems unter anderem dadurch getestet, daß wir Theoreme eines mathematischen Lehrbuches [Deu71] mit dem System bewiesen haben. Obwohl viele Theoreme formal in MKRP bewiesen werden konnten, ist die abschließende Bewertung doch eher negativ, denn es zeigten sich prinzipielle Schwächen von automatischen Beweisern, die allen automatischen Beweisern anhaften und die eine Benutzung eines solchen klassischen Systems als Werkzeug im Sinne eines rechnergestützten mathematischen Assistenten utopisch erscheinen lassen. Diese Schwächen haben dazu geführt, daß unser derzeitiges Forschungsinteresse im wesentlichen nicht mehr im Rahmen des klassischen Beweiserbaus liegt, sondern daß sich die Arbeiten auf ein neues Paradigma und System, nämlich das Ω–MKRP-System, konzentrieren. Die negativen Erfahrungen lassen sich im wesentlichen so zusammenfassen:

1. Die Darstellung mathematischer Probleme in einer Logik erster Stufe ist oft nicht einfach, selbst wenn es sich dabei um eine Logik mit Gleichheit und Sorten handelt. Es zeigte sich bei der tatsächlichen Kodierung des Lehrbuches, daß eine ausdrucksstärkere Sprache notwendig ist, da sonst der Aufwand für die Kodierung sehr groß wird und der Zusammenhang zwischen den Fakten im Lehrbuch und deren Kodierung oft nicht mehr offensichtlich ist. Diese Sprache muß auf jeden Fall Konstrukte höherer Stufe enthalten. Damit sie möglichst nahe an die von Mathematikern verwendete Sprache herankommt, sind Sorten als Ausdrucksmittel wichtig. Dies hat bei uns zur Entwicklung der Objektsprache $\mathcal{POST}$ geführt, die weiter unten ausführlicher dargestellt wird.

2. Ein Beweiser wie MKRP zeichnet sich unter anderem dadurch aus, daß er universell einsetzbar ist, weil er zunächst kein spezielles mathematisches Wissen hat. Daher ist es nötig, die für den Beweis notwendigen Informationen für jeden Satz immer wieder neu einzugeben. Dieser Prozeß ist nicht nur sehr umständlich, sondern aufgrund der Redundanz in verschiedenen Versionen auch in hohem Maße fehleranfällig. Ein System, das die enorme Informationsmenge von einigen Megabytes, die bei der Kodierung eines einzigen Lehrbuches anfällt, verwalten muß, benötigt eine zentrale Datenbank des mathematischen

Faktenwissens. Dazu haben wir die Ω-Datenbank entwickelt und aufgebaut, die es gestattet mathematisches Wissen strukturiert abzulegen.

3. Oft sind Sätze zu schwer, als daß sie vollautomatisch bewiesen werden können. Will man ein solches System als Werkzeug benutzen, so muß der Benutzer die Möglichkeit haben, in den Beweisprozeß eingreifen und diesen gegebenenfalls interaktiv steuern zu können. Leider ist das bei klassischen Beweissystemen nahezu unmöglich, da der Suchraum riesig ist, die Beweissuche in einem nur Eingeweihten verständlichen Format abläuft und zudem wenig geeignet ist, die menschliche Intuition zu unterstützen. Die derzeitigen interaktiven Systeme für die Beweisentwicklung sind zwar sehr viel besser in ihren Repräsentationsmöglichkeiten mathematischer Sachverhalte, aber sie sind oft relativ schwach in ihrer deduktiven Unterstützung.

4. Für Beweiser, die wie das MKRP-System auf einer Normalform arbeiten, stellt sich die Frage, wie gefundene Beweise so präsentiert werden können, daß sie dem Benutzer verständlich sind. Für ein System, das als interaktives Werkzeug eingesetzt werden soll, erhält diese Frage natürlich noch viel stärkeres Gewicht. Der Benutzer will keine Klauselbeweise, die oft einige hundert Schritte lang sind, lesen müssen, sondern möglichst gut gewählte und strukturierte Darstellungen, die die wesentlichen Schritte betonen. In unserem Fall wird in den Beweisen die Sprache $\mathcal{POST}$ zur Darstellung verwendet und durch geeignete Transformationen in eine mathematische Umgangssprache überführt.

Um Mathematikern eine größtmögliche automatische Unterstützung bei der Beweisentwicklung bieten zu können, haben wir das im folgenden beschriebene System Ω–MKRP entworfen und bisher teilweise implementiert. Es ist eine interaktive Beweisentwicklungsumgebung, die Vorteile des maschinenorientierten Beweisens, wie der Resolution, und des eher auf menschlichem Vorgehen aufbauenden planungsbasierten Beweisens, wie es von Bundy [Bun88] und anderen vorgeschlagen wurde, zu verbinden versucht. Dazu sollen in einer Umgebung, in der ein Benutzer seine Beweise interaktiv entwickeln kann, klassische automatische Beweiser erster Stufe wie das MKRP-System integriert werden, aber auch Beweiser für Logiken höherer Stufe wie TPS [AINP90] oder unser eigenes System LEO, das weiter unten vorgestellt wird. Dadurch werden dem Benutzer die Möglichkeiten an die Hand gegeben, die eigenen Ideen auszuführen, ohne daß er wie in einer der heutigen Beweisentwicklungsumgebungen gezwungen ist, den Beweis durch die Angabe von Kalkülregeln oder zusammengesetzten Kalkülregeln (sogenannten Taktiken) detailliert selbst führen zu müssen.
Das Problem wird am Anfang als trivialer Beweisbaum eingeführt, und dieser wird als zentrale Datenstruktur solange modifiziert, bis ein endgültiger Beweis erzeugt ist. Der Benutzer hat dabei mehrere Möglichkeiten einzugreifen, er kann insbesondere

- mathematisches Faktenwissen aus der Ω-Datenbank in den Beweisbaum einbauen (im folgenden gehen wir darauf nicht ein, siehe [Ker91b, SK93]),

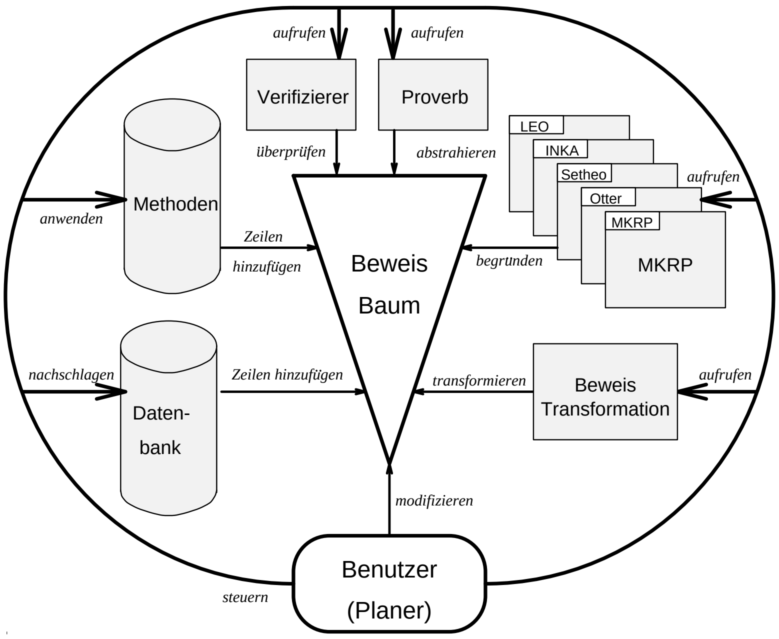

Abbildung 1: Architektur von Ω–Mkrp

- verschiedene automatische Beweiser aufrufen, unter anderem Mkrp oder Leo (vergleiche Abschnitt 2),
- auf vorhandene Methoden aus der Methodenbank zurückgreifen, die dann zum Beispiel einzelne Beweisschritte ausführen (vergleiche Abschnitt 3),
- neue Methoden erzeugen lassen durch die Kombination bestehender Methoden oder durch die Anwendung von Metamethoden, oder
- einen Analogieplaner verwenden, der Beweispläne durch Analogie zu bereits vorhandenen erstellt (vergleiche Abschnitt 3),
- den erzeugten Beweis von einem Verifizierer überprüfen lassen und
- automatisch erzeugte Beweise in das Format des natürlichen Schließens transformieren oder den Gesamtbeweis in natürliche Sprache übersetzen lassen (vergleiche Abschnitt 4).

Man kann die Aufgabe des Benutzers als einen abstrakten *Planungs*prozeß auffassen, in dem ein Beweis interaktiv mit dem System entwickelt wird. Der Benutzer plant sein Vorgehen auf einer wesentlich abstrakteren Ebene als auf der eines konkreten Beweises, und er kann dabei allgemeine Lemmata oder Beweismethoden verschiedener Granularität benutzen. Diese menschliche Planungsaufgabe soll zunehmend von einer automatischen Planungskomponente unterstützt werden, so daß der Automatisierungsgrad von Ω–MKRP auf die Dauer steigen wird. Der Planer kann den Beweisbaum so modifizieren, daß als Begründungen für Beweiszeilen, die Methoden oder Beweise der angeschlossenen vollautomatischen Beweiser eingetragen werden. Um das Gesamtproblem zu beweisen, müssen diese Schritte abschließend vom Verifizierer überprüft werden, da wir – wie auch Bundy – nicht fordern, daß eine Methode immer erfolgreich, ja nicht einmal korrekt sein muß. Einen Überblick über die Architektur von Ω–MKRP gibt Abbildung 1.

2 Mechanisierung von Logiken höherer Stufe

Der Beweiserbau für Logiken höherer Stufe ist durch die Beobachtung motiviert, daß die mathematische Fachsprache (und damit auch die in fast allen mathematischen Lehrbüchern verwendete Sprache) inhärent Konstrukte höherer Stufe verwendet, also Konstrukte, in denen nicht nur über Eigenschaften von *Objekten* gesprochen wird, wie in $\mathsf{Gerade}(4)$, sondern auch über solche von *Funktionen* wie $\mathsf{Stetig}(f)$ oder von Prädikaten, wie $\mathsf{Symmetrisch}(=)$. Um eine Sprache zu entwickeln, in der sich solche mathematischen Konzepte adäquat darstellen lassen, hat man die Alternative, entweder diese Konstrukte in einer Mengentheorie zu kodieren oder direkt in einer Prädikatenlogik höherer Stufe. In der mathematischen Praxis verwendet man meist eine Mischung aus beidem: die Mengenlehre stellt zwar ein mächtiges Hilfsmittel zur Beschreibung mathematischer Sachverhalte dar, aber auch Konstrukte höherer Stufe sind üblich und werden zudem, vor allem durch die Wahl der Notation (n, m, k für natürliche Zahlen; f, g, h für Funktionen usw.) implizit sortiert beziehungsweise getypt. Die theoretische Möglichkeit wieder alles in die Mengenlehre zu übersetzen, wird praktisch natürlich nie durchgeführt, sondern die Schlüsse werden vielmehr direkt in der jeweiligen (im allgemeinen sehr mächtigen und mit Spezialkonstrukten angereicherten) Fachsprache durchgeführt.

Entsprechend gibt es für die Automatisierung nun zwei Möglichkeiten: man kann die mathematischen Konstrukte in eine Mengenlehre wie die von Zermelo-Fraenkel oder in die für die anschließende Mechanisierung wesentlich besser geeignete von von Neumann-Gödel-Bernays kodieren (wie zum Beispiel in [BLM+86, Qua92] vorgeschlagen) und versuchen, einen automatischen Beweiser erster Stufe zu verwenden, der um Verfahren zur Behandlung der anfallenden Mengenkonstrukte angereichert wird. Eine solche Behandlung hat sich zumindest in unserem Fall als nicht sehr erfolgversprechend herausgestellt: allein die Darstellung einer Funktion als linkstotale, rechtseindeutige Relation ist für dieses ja noch relativ simple Konstrukt so aufwendig, daß an eine erfolgreiche Bearbeitung schwieriger Probleme mit einem allgemeinen Beweiser erster Stufe nicht zu denken ist.

Daher haben wir uns entschlossen, den zweiten Weg einzuschlagen und als Ausgangspunkt eine sortierte Logik höherer Stufe zu verwenden. Eine Übersetzung von Konstrukten der Mengenlehre in eine solche Logik ist relativ natürlich, so daß man

mit einer solchen Sprache eine gute Annäherung an die mathematische Fachsprache erreichen kann. Aus diesen Überlegungen entstand die sortierte Sprache höherer Stufe $\mathcal{POST}$, die wir kurz vorstellen wollen.
Für ihre Operationalisierung haben wir beide Hauptansätze zur Automatisierung von Logiken höherer Stufe untersucht, nämlich zum einen die Übersetzung von $\mathcal{POST}$ in eine (sortierte) Logik erster Stufe und zum anderen die Entwicklung von Beweisern für Logiken höherer Stufe. Beiden Ansätzen ist gemeinsam, daß sie für die Bewertung der Vollständigkeit der entwickelten Methoden nicht die Standardsemantik einsetzen, da es bezüglich dieser Semantik nach dem Gödelschen Unvollständigkeitsresultat keine vollständigen Kalküle geben kann. Vielmehr bauen sie wie üblich auf verallgemeinerten Semantiken auf, die auf Henkin [Hen50] zurückgehen.

2.1 $\mathcal{POST}$

Die Sprache $\mathcal{POST}$ ist die universelle Repräsentationssprache für das Ω–MKRP-Projekt. Um diesen Anspruch erfüllen zu können, muß $\mathcal{POST}$ die teilweise widersprüchlichen Anforderungen von möglichst maximaler Expressivität einerseits und der Mechanisierbarkeit andererseits berücksichtigen. Bei dem Entwurf waren deswegen einerseits die Anforderungen der mathematischen Praxis und andererseits die Suchraumeinschränkungen, die zur Entwicklung von effizienten automatischen Beweisern nötig sind, maßgebend. $\mathcal{POST}$ basiert auf einer Logik höherer Stufe in Form eines getypten λ-Kalküls [Chu40]. In diesem sind im Gegensatz zur Prädikatenlogik erster Stufe auch sogenannte λ-Abstraktionen der Form $\lambda x\textbf{.}\ A$ erlaubt, wobei dieser Ausdruck eine Funktion repräsentiert, die einen Term B auf den Term abbildet, den man erhält, wenn man in A alle freien Vorkommen von x durch B ersetzt (diese Ersetzung nennt man λ-Konversion). Zum Beispiel läßt sich die Quadratfunktion durch den λ-Ausdruck $\lambda x\textbf{.}\ x \cdot x$ darstellen. Die Theorie der λ-Konversion wird in λ-Kalkülen als universell gültig angenommen.
Wir haben in $\mathcal{POST}$ diese Logik höherer Stufe mit Sorten angereichert, so daß das Typsystem zu einer ordnungssortierten Struktur verallgemeinert wird. Die Erfahrungen mit dem Beweisen erster Stufe haben gezeigt, daß sortierte Logiken sowohl die Expressivität als auch die Mechanisierbarkeit erhöhen, denn Sorten wirken bei geschickter Nutzung als Randbedingungen, die den Suchraum einschränken. Darüberhinaus hat der Benutzer die Möglichkeit, einen Teil seines taxonomischen Wissens prägnant und adäquat in der Signatur auszudrücken und so den Inferenzprozeduren für eine entsprechende Spezialbehandlung zugänglich zu machen.
Durch die Sorten und die Möglichkeit Termdeklarationen anzugeben, unterstützt $\mathcal{POST}$ inbesondere die Formalisierung von Funktionsklassen mit Werte- und Definitionsbereich. So lassen sich zum Beispiel in der folgenden Signatur

$$[+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}], [+ : G \times G \to G], [+ : U \times U \to G], [+ : G \times U \to U],$$
$$[+ : U \times G \to U], [(x_{\mathbb{N}} + x_{\mathbb{N}}) : G]$$

die Eigenschaften der Addition auf natürlichen Zahlen ($\mathbb{N}$), geraden (G) und unge-

raden (U) Zahlen sehr natürlich spezifizieren. Für die letzte Termdeklaration wurde zum Beispiel ausgenutzt, daß die Verdoppelung einer natürlichen Zahl immer zu einer geraden Zahl führt.
Im Zusammenhang mit Funktionen höherer Stufe entfalten sortierte λ-Kalküle mit Termdeklarationen eine besondere Ausdruckskraft: so läßt sich zum Beispiel die Klasse der Polynome (P) durch die folgende Signatur als Subklasse der Funktionen auf den reellen Zahlen (IR) vollständig charakterisieren

$$[(\lambda x_{\mathsf{IR}}.\, a_{\mathsf{IR}}) : P], [(\lambda x_{\mathsf{IR}}.\, x) : P], [(\lambda x_{\mathsf{IR}}.\, f_P(x) + g_P(x)) : P], [(\lambda x_{\mathsf{IR}}.\, f_P(x) \cdot g_P(x)) : P].$$

Hier wird ausgenutzt, daß konstante Funktionen ($p(x) = a$) und die Identitätsfunktion ($p(x) = x$) Polynome sind, und die Menge der Polynome der Abschluß dieser beiden Funktionen unter der Summe und dem Produkt von Funktionen ist.

2.2 Übersetzung von höherer Stufe in erste Stufe

Zur Operationalisierung von $\mathcal{POST}$ gibt es wie erwähnt die Möglichkeit, eine sortierte Logik erster Stufe zu verwenden. Dazu werden Ausdrücke der Logik höherer Stufe durch eine sogenannte Reifikation in die Logik erster Stufe übersetzt [Ker91a]. Dies geschieht durch die Einführung neuer Funktions- und Prädikatssymbole α, so daß beispielsweise Ausdrücke der Form $\mathsf{Stetig}(f)$ in $\alpha(\mathsf{Stetig}, f)$ übersetzt werden. Sowohl Stetig als auch f werden in diesem Ausdruck als Objekte erster Stufe aufgefaßt. Durch diese Übersetzung ist es auch möglich über Objekte zu quantifizieren, die ursprünglich Funktionssymbole oder Prädikatssymbole waren, wie in $\forall P.\, \mathsf{Symmetrisch}(P) \Leftrightarrow (\forall x, y.\, P(x, y) \Rightarrow P(y, x))$.
Um allerdings bezüglich einer im Henkinschen Sinne verallgemeinerten Semantik vollständig zu sein, reicht es nicht aus, nur alle Formeln auf diese Weise zu übersetzen. Es sind weitere Axiome erforderlich, nämlich die Extensionalitätsaxiome und Komprehensionsaxiome.
Die Extensionalitätsaxiome besagen, daß zwei Funktionen, die in all ihren Werten übereinstimmen, gleich sind. Es handelt sich also um Formeln der Form
$\forall f, g.\, \forall x.\, \alpha(f, x) = \alpha(g, x) \Rightarrow f = g$.
Die Komprehensionsaxiome erlauben das Zusammenfassen von Ausdrücken. Schauen wir uns dazu ein Beispiel an, nämlich die Aussage, daß alle Menschen alle Eigenschaften vom Vater oder von der Mutter ererbt haben, das heißt, daß sich jede ihrer Eigenschaften bei Vater oder Mutter wiederfinden läßt. Dies kann man in einer Logik zweiter Stufe durch folgende Formel ausdrücken: $\forall P.\, \forall x.\, \mathsf{mensch}(x) \wedge P(x) \Rightarrow P(\mathsf{vater}(x)) \vee P(\mathsf{mutter}(x))$. Die Prädikatsvariable P kann zum Beispiel mit der Prädikatskonstanten $\mathsf{kurzsichtig}$ instantiiert werden. Soll $P(t)$ durch eine Formel wie $(\mathsf{trinkt\_kaffee}(t) \Rightarrow \mathsf{schläft\_schlecht}(t))$ ersetzt werden, braucht man bei einer Übersetzung das folgende Komprehensionsaxiom: $\exists Q.\, \forall x.\, Q(x) \Leftrightarrow (\mathsf{trinkt\_kaffee}(x) \Rightarrow \mathsf{schläft\_schlecht}(x))$. Da es von diesen Axiomen a priori unendlich viele gibt, bereiten sie beim Übersetzungsansatz besondere Probleme. Allerding sind für den Beweis

vieler Theoreme keine solchen Axiome notwendig, dann nämlich wenn die Theoreme zwar eine Syntax höherer Stufe verwenden, aber direkt in erster Stufe beweisbar sind. Es gibt zwar unendlich viele Komprehensionsaxiome, aber zu einer gegebenen Signatur lassen sich diese endlich repräsentieren [Ker94], so daß man prinzipiell aus einem endlichen Problem der Logik höherer Stufe ein endliches Problem der Logik erster Stufe erhält. Allerdings kann diese prinzipielle Endlichkeit alle vernünftigen Schranken heutiger Rechner sprengen.

2.3 Direkte Mechanisierung von $\mathcal{POST}$

Der andere Ansatz zur Operationalisierung von $\mathcal{POST}$ besteht in der Entwicklung von maschinenorientierten Kalkülen direkt für diese Sprache. Dieser kommt im Gegensatz zum Übersetzungsansatz ohne die Komprehensionsaxiome aus, da diese in der Theorie der λ-Konversion bereits fest in die Unifikation eingebaut sind. Eine Mechanisierung der Extensionalitätsaxiome konnte in [Koh95] erstmals erreicht werden. Unsere Arbeiten zum Beweisen in $\mathcal{POST}$ bauen auf Huets Unifikationsalgorithmus für den einfach getypten λ-Kalkül [Hue72] auf, der zusammen mit den Arbeiten von Andrews [And71] die Grundlage für einen Resolutionskalkül der Logik höherer Stufe bildet. Dieser Algorithmus hat allerdings auf theoretischer Ebene eine ganze Reihe von technischen Schwierigkeiten wie die Unentscheidbarkeit der Unifikation, die Nichtexistenz von allgemeinsten Unifikatoren und die enorme Proliferation möglicher Unifikatoren zu meistern.
Im Tps-Projekt [AINP90] von Andrews, in dem seit über fünfzehn Jahren System zum Beweisen von Sätzen höherer Stufe gebaut wurde, konnte zwar die generelle Machbarkeit der Mechanisierung von Logiken höherer Stufe gezeigt werden, allerdings wurde auch deutlich, daß trotz der praktischen Verbesserungen über diesen langen Zeitraum die angewandten Methoden noch zu schwach für den praktischen Einsatz sind. Insbesondere sind Systeme wie Tps auf dem Fragment erster Stufe weit schwächer als heutige Beweiser erster Stufe. Unserer Meinung nach lassen sich die Schwächen hauptsächlich darauf zurückführen, daß in Systemen höherer Stufe all die Methoden und Techniken (wie die Spezialbehandlung für Sorten und Gleichheit, mächtige Suchstrategien und ausgefeilte Implementierungstechniken wie das Termindexing), die das automatische Beweisen erster Stufe erfolgreich gemacht haben, noch nicht eingesetzt beziehungsweise entwickelt worden sind. Jüngste Entwicklungen in diesem Gebiet werden auch stark durch logische Programmiersprachen höherer Stufe beeinflußt. Wichtige Systeme, die auf getypten λ-Kalkülen höherer Stufe basieren sind zum Beispiel Isabelle [Pau90], λ-Prolog [NM88] und ELF [Pfe91].
In $\mathcal{POST}$ stellte sich die Entwicklung von Unifikationsalgorithmen als ein sehr schwieriges Unterfangen heraus, da die Mechanismen der λ-Konversion, der Sorten und der Termdeklarationen auf nichttriviale Weise interagieren. Insbesondere ist im Gegensatz zu einfach getypten λ-Kalkülen eine Eindeutigkeit von approximierenden Bindungen einer gegebenen Sorte nicht mehr garantiert, da jede Termdeklaration mit entsprechendem Kopfsymbol einen Beitrag leisten kann. Dieses Phänomen ist aller-

dings keine Besonderheit des λ-Kalküls, da es bereits bei Termen erster Stufe, wie sie von Schmidt-Schauß [SS89] betrachtet wurden, auftritt. Allerdings lassen sich mit dem Hilfsmittel der approximierenden Bindungen auf herkömmliche Weise Unifikationsalgorithmen höherer Stufe [JK94, KP93, Koh94] bauen, deren Verzweigungsrate im Vergleich zum getypten Fall durch gewisse Termdeklarationen stark beschränkt ist.
Durch die sortierte Unifikation lassen sich die bekannten Widerlegungskalküle für Logiken höherer Stufe [Hue72, And89] zu sortierten Kalkülen [Koh94] für $\mathcal{POST}$ verallgemeinern. Im System Leo (**L**ogic **E**ngine for Ω–Mkrp) werden im Moment verschiedene sortierte Widerlegungskalküle höherer Stufe prototypisch implementiert und auf ihre praktische Verwendbarkeit getestet.
In der Logik erster Stufe haben sortierte Unifikationsalgorithmen mit Termdeklarationen direkt zur Entwicklung spezieller Resolutionskalküle mit dynamischer Sorteninformation [Wei91] geführt. Diese wurden dann in [KK94, KK95] ihrerseits eine Grundlage für die Mechanisierung partieller Funktionen, basierend auf den bekannten Kleene-Logiken [Kle52]. Diese Ansätze sollen in $\mathcal{POST}$ verallgemeinert werden, um damit Kalküle zu haben, die mit der Behandlung von Sorten und partiellen Funktionen einen signifikanten Teil der mathematischen Umgangssprache adäquat mechanisieren.

3 Beweisplanung

Eine wichtige Komponente zur Unterstützung des Benutzers ist ein rechnergestütztes Verfahren zur *Beweisplanung*. Der wesentliche Unterschied zum traditionellen Vorgehen der *Beweissuche* ist, daß nicht alle möglichen Inferenzschritte aufgelistet werden, um in diesem (riesigen) Suchbaum dann eine Sequenz von Schritten zu finden, die den Satz beweist. Stattdessen wird der Beweis auf einer viel allgemeineren Ebene, unter der Verwendung von bereichsspezifischen Methoden *geplant*. Natürlich gibt es auch hier wieder einen Suchraum, aber durch die abstraktere Sichtweise, die dem menschlichen Vorgehen sehr viel näher kommt, werden diese Suchräume auch für schwierige Probleme handhabbar. Wir verfolgen dabei eine Erweiterung von Bundys Ansatz [Bun88], den wir im folgenden beschreiben wollen.
Der Beweisplanungsansatz von Bundy baut auf den taktikbasierten Deduktionssystemen wie Nuprl [Con86] auf. Während die Taktiken in Nuprl Beweisprozeduren sind, werden in Bundys Ansatz die Taktiken zu sogenannten *Methoden* erweitert, indem sie um Spezifikationen ergänzt werden. Methoden stellen damit Planungsoperatoren im klassischen Sinne der Planverfahren in der KI dar, mit denen eine rechnergestützte Planung des Beweises möglich ist. Wenn ein vollständiger Plan gefunden ist, wird dieser ausgeführt, das heißt die entsprechenden Taktiken werden auf den partiellen Beweis angewendet und auf diese Weise wird dann ein fertiger Beweis auf Kalkülebene generiert. Da die Spezifikation der Methoden in der Regel eine Abstraktion der Eigenschaften der Taktik ist, muß der gefundene Beweis abschließend

auf seine Korrektheit überprüft werden. Falls der Beweis nicht vollständig ist, wird der Planungsvorgang mit den verbliebenen, offenen Teilproblemen rekursiv erneut gestartet. Der gesamte Prozeß von der Analyse des Problems bis zum vollständigen Beweis ist also ein verschränkter Prozeß von Beweisplanung, Anwendung von Taktiken und anschließender Verifikation (und eventueller menschlicher Interaktionen).
Unsere Erweiterung des Ansatzes von Bundy besteht darin, zusätzlich Metamethoden einzuführen, die die bestehenden Methoden verändern können. Dies wird dadurch motiviert, daß Mathematiker die von ihnen erlernten Beweisverfahren durchaus abändern, um sie an neue Situationen anzupassen. Um dies auch mit den im Rechner benutzten Methoden tun zu können, haben wir vorgeschlagen, Methoden und Taktiken teilweise deklarativ zu beschreiben [HKKR94], da die Änderung einer deklarativen Sprache durch die Metamethoden natürlich einfacher ist als die einer prozeduralen.

3.1 Methoden

Methoden sollen sowohl Beweisprozeduren (die Taktiken) repräsentieren als auch entsprechende Planungsoperatoren (durch deren Spezifikation). Da außerdem Methoden weitgehend deklarativ beschrieben werden sollen, benötigen wir Metavariablen, die für Symbole, Formeln, Beweiszeilen usw. stehen. Zu diesem Zweck bestehen Methoden in unserem Ansatz aus sechs Komponenten, von denen die ersten fünf deklarativ sind, während die letzte prozedural ist:

- Die *Deklaration* ist eine Signatur, die die verwendeten Metavariablen definiert.
- Die *Voraussetzungen* bestehen aus einer Liste von Beweiszeilen, die von der Methode benutzt werden, um die sogenannten Folgerungen zu beweisen.
- Die *Randbedingungen* sind zusätzliche Bedingungen, die gelten müssen, bevor die Methode angewendet werden kann.
- Die *Folgerungen* bestehen aus einer Liste von Beweiszeilen, die von der Methode bewiesen werden.
- Der *Deklarative Inhalt* wird von dem prozeduralen Inhalt interpretiert.
- Der *Prozedurale Inhalt* ist entweder eine Standardprozedur, die den deklarativen Inhalt interpretiert, oder eine spezielle Inferenzprozedur, die für einen ganz bestimmten Problemtyp geschrieben wurde.

Die *Voraussetzungen*, *Randbedingungen* und *Folgerungen* geben an, ob eine Methode in einer bestimmten Beweissituation anwendbar ist. Während diese Spezifikation also alle Informationen enthält, die für den Planer notwendig sind, repräsentieren der *deklarative Inhalt* und der *prozedurale Inhalt* zusammen eine Taktik, die in Systemen wie NUPRL oder in Bundys Ansatz als Prozedur realisiert würden. Die Taktik generiert später den eigentlichen Beweis. Die Aufteilung von deklarativem und prozeduralem Wissen einer Taktik bewegt sich innerhalb einer breiten Spannweite. Die

einzige Bedingung, die wir an Methoden stellen, ist, daß der prozedurale Inhalt Beweiszeilen konstruiert, die in den aktuellen Beweiszustand eingefügt werden können. Im einen Extrem ist der prozedurale Inhalt ein einfacher Interpreter, der das Beweisschema im deklarativen Inhalt in den aktuellen Beweiszustand einfügt (indem er die Metavariablen entsprechend bindet); im anderen Extrem (das genau Bundys Ansatz entspricht) ignoriert der prozedurale Inhalt den leeren deklarativen Inhalt und konstruiert neue Beweiszeilen rein prozedural.
Würden wir nur rein prozedurale Taktiken erlauben, so müßten bei einer Modifikation von Methoden die entsprechenden Programme verändert werden. Wird eine Methode durch eine Metamethode modifiziert, so bleibt das prozedurale Wissen unverändert, nur das deklarative wird geändert. Erst durch die Trennung von prozeduralem und deklarativem Wissen in den Methoden werden Metamethoden praktisch realisierbar.

3.2 Ein Beispiel: Die Homomorphie-Methode

Wir wollen nun die `hom1-1`-Methode beispielhaft vorführen (siehe Abbildung 2). Das Beweisverfahren, das durch diese Methode repräsentiert wird, kann informell so beschrieben werden: *Wenn F eine* in Zeile L_1 *gegebene Funktion und P ein* in L_2 *definiertes Prädikat ist und wir haben das Ziel,* in L_6 $P(F(C))$ *zu zeigen, dann zeige* in L_3 $P(C)$ *und benutze dies, um $P(F(C))$ zu zeigen.* Die Idee dieses Verfahrens ist, daß F ein Homomorphismus bezüglich der Eigenschaft P ist.

Methode : `hom1-1`	
Deklarationen	$L_1, L_2, L_3, L_4, L_5, L_6$: beweiszeile J_1, J_2, J_3: begründung $\quad X, Y$: variable $\Phi, \Psi, \Psi_1, \Psi_2$: formel $\quad P, F, C$: constante
Voraussetzungen	L_1, L_2, L_3
Rand-bedingungen	typ(C) $\doteq$ ergebnistyp(F) & $\Psi_1 := \Psi[X \leftarrow C]$ & $\Psi_2 := \Psi[X \leftarrow F(C)]$
Folgerungen	L_6
Deklarativer Inhalt	$(L_1)\ (L_1) \vdash \forall Y.\, \Phi_F \quad (J_1)$ $(L_2)\ (L_1, L_2) \vdash \forall X.\, P(X) \Leftrightarrow \Psi \quad (J_2)$ $(L_3)\ (L_1, L_2) \vdash P(C) \quad (J_3)$ $(L_4)\ (L_1, L_2) \vdash \Psi_1 \quad$ (`def-e` L_2, L_3) $(L_5)\ (L_1, L_2) \vdash \Psi_2 \quad$ (OFFENL_1, L_4) $(L_6)\ (L_1, L_2) \vdash P(F(C)) \quad$ (`def-i` L_2, L_5)
Prozeduraler Inhalt	`schema-interpreter`

Abbildung 2: Die Methode `hom1-1`.

Die Spezifikation von `hom1-1` fordert in L_1 und L_2 lediglich die Existenz einer Definition für F und P und keine weiteren Eigenschaften. Die Ausführung der Taktik hinterläßt daher eine Lücke im Beweis: Nachdem ein vollständiger Plan gefunden wurde, fügt die Taktik von `hom1-1` die Zeilen L_1 bis L_6 in den Beweis ein; für L_1, L_2 und L_3 sind durch das Planen Begründungen gefunden worden; L_4 und L_6 erhalten eine Begründung, indem die Untertaktiken `def-e` und `def-i` ausgeführt werden, die die Definition von P aus Zeile L_2 auf die Formeln aus L_3 bzw. L_6 anwenden; die Begründung von Zeile L_5 bleibt jedoch offen. Dies führt in der folgenden Verifikationsphase zu einem rekursiven Aufruf der Beweisplanung. Kann L_5 so bewiesen werden, ist der Beweis vollständig geführt, andernfalls ist die Anwendung von `hom1-1` gescheitert.

Die Methode `hom1-1` läßt sich nur für ein einstelliges Prädikat und eine einstellige Funktion benutzen (daher der Name `hom1-1`). Ein entsprechendes, auf Homomorpie beruhendes Beweisverfahren läßt sich auch auf mehrstellige Prädikate oder Funktionen anwenden. In unserem Ansatz kann aus der Methode `hom1-1` eine passende Methode durch eine Metamethode `add-argument` automatisch erzeugt werden [HKKR94]. Die Metamethode erhält als Argumente eine Methode und ein einstelliges Funktions- bzw. Prädikatssymbol G; sie ersetzt dann alle Vorkommen von $G(A)$ durch $G'(A, B)$ und fügt zu jeder Zeile, in der solch ein A ohne G vorkommt, eine zusätzliche Kopie ein, in der A durch B ersetzt ist.

Als Ergebnis der Anwendung von `add-argument` auf `hom1-1` und F erhält man eine Methode `hom1-2`, die unter anderem als Folgerung die Zeile $P(F'(C, D))$ und als Voraussetzung die Zeilen $P(C)$ und $P(D)$ enthält. Diese Methode kann man nun auf das neue Beweisziel $symmetrisch(\sigma \cup \rho)$ anwenden, und man erhält die Teilziele $symmetrisch(\rho)$ und $symmetrisch(\sigma)$ als Ergebnis. Dies ist ein wichtiger Schritt in dem Beweis, daß die Vereinigung zweier symmetrischer Relationen wieder symmetrisch ist.

3.3 Rechnergestützte Analogiebildung

Eine der wichtigsten Heuristiken, die Menschen bei der Suche nach Beweisen verwenden, ist die Analogie. Beweisen durch Analogie bedeutet, einen Beweis für ein Zieltheorem T_2 zu finden, indem man versucht, den Beweis eines ähnlichen Quelltheorems T_1 zu übertragen.

Um Analogie effizient im Ω–MKRP-System behandeln zu können, haben wir die folgenden Fragen näher untersuchen:

- Welche Operationen führen Mathematiker aus, um den Beweis des Theorems T_2 analog zum Beweis eines Theorems T_1 zu führen?
- Wie kann man diese Operationen formal repräsentieren und implementieren?
- Wie kann man Analogien auf einer abstrakteren Ebene als der eines Kalkülbeweises, zum Beispiel auf der von Beweisideen erfassen?

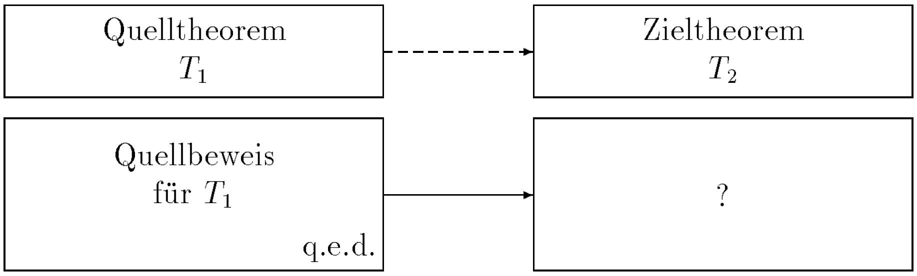

Abbildung 3: Analogie im Theorembeweisen

Um diese Fragen für real vorkommende nicht-triviale Beispiele zu lösen, haben wir zunächst empirische Untersuchungen der Beweise in [Deu71] durchgeführt, die vom Autor explizit als „analoger Beweis“ angegeben waren.
Diese Untersuchungen haben ergeben, daß bisherige Ansätze, Beweisen durch Analogie zu automatisieren (siehe zum Beispiel [Owe90, Slo92]), nur bedingt geeignet sind, tatsächlich vorkommende Beweise durch Analogie zu finden, da diese Verfahren vor allem auf der Abbildung von Symbolen des Quelltheorem auf Symbole des Zieltheorem und auf dem Transfer von logischen Kalkülschritten beruhen.
Es zeigt sich, daß auch Abstraktionen und andere *Reformulierungen* für die untersuchten Analogien erforderlich sind, die über Symbolabbildungen hinausgehen. Diese Reformulierungen führen Heuristiken aus, die den Zusammenhang zwischen den Änderungen einer zu beweisenden Formel und der Änderung ihres Beweisplanes beschreiben. Beispielsweise wird eine Reformulierung, die die Argumente einer Funktion vervielfacht, häufig benutzt, um Theoreme von n-dimensionalen Räumen auf $(n+m)$-dimensionale Räume zu übertragen. Eine solche Reformulierung kann nicht durch schlichte Symbolabbildung erreicht werden, weil das Vervielfachen von Argumenten die Beweisstruktur verändert. Soviel zur ersten Frage.
Reformulierungen werden durch die schon erwähnten Metamethoden, die Beweispläne verändern, ausgeführt. Einzelne Reformulierungsprozeduren sind zu Testzwecken implementiert worden. Dadurch wurde es möglich, einige (relativ schwierige) mathematische Theoreme erstmalig per Analogie zu beweisen. Ein interessantes Beispiel ist der Beweis eines zweidimensionalen Heine-Borel-Theorems, der durch Analogie zu einem eindimensionalen Heine-Borel-Theorem geführt werden konnte [Mel95b].
Neben der Einführung von Reformulierungen hat die Beantwortung der dritten Frage zu einem neuen theoretischen Ansatz für die Analogiebildung im Theorembeweisen geführt hat (siehe [Mel95a]): In sequentiell aufgeschriebenen Beweisen kann man die Strukturierung von Beweisen in Beweisidee und Beweisdetails nicht erkennen. Anders verhält es sich bei der Repräsentation von Beweisen durch Beweispläne, in denen hierarchische Methoden Beweisideen darstellen können.
Wenn Beweispläne übertragen werden, kann es vorkommen, daß nur die Beweisidee transferiert werden kann. Beispielsweise ist nur die Beweisidee vom Theorem T_1 „Es gibt unendlich viele Primzahlen der Form $(4k+3)$“ analog übertragbar auf das

Theorem T_2 „Es gibt unendlich viele Primzahlen der Form $(4k+1)$" [Ler83]. Für andere Theoreme, beispielsweise für die Heine-Borel-Theoreme, sind auch fast alle Beweisdetails übertragbar.
Außerdem umgeht man mit dem Transfer von Beweisplänen Probleme, die von Versuchen mit der detaillierten Übertragung logischer Kalkülschritte bekannt sind. Besonders deutlich wird das bei Methoden, die den Aufruf eines automatischen Beweisers beinhalten, der einen kleinen Beweis einer Teilbehauptung ausführt. Die analoge Übertragung erfordert nur, daß der Beweiser den Beweis einer reformulierten Teilbehauptung führen kann, jedoch nicht, daß jeder logische Kalkülschritt übertragen werden muß. Dadurch wird die analoge Übertragung robuster.
Unsere analogiegeleitete Beweisplankonstruktion ist eine Kontrollstrategie für die Beweisplanung, die anstelle der Suche nach anwendbaren Methoden die Übertragung der Methoden aus einem Quellbeweisplan bevorzugt. Für ein Teilziel im Zielbeweisplan wird ein entsprechendes Teilziel im Quellbeweisplan ausgesucht und versucht zu reformulieren, so daß die beiden Teilziele übereinstimmen. Diese Reformulierung kann, wie wir gesehen haben, den Quellbeweisplan verändern. Dann wird versucht, die relevante Methode aus dem reformulierten Quellplan auf den Zielbeweisplan zu übertragen. Lassen sich die Begründungen in den Beweiszeilen nicht übertragen, wird versucht, durch allgemeine Beweisplanung neue Begründungen zu erzeugen. Das heißt, analoge Übertragung und Planung können sich gegenseitig ablösen und ineinander verschränkt werden.

4 Beweispräsentation

Eine wichtige, aber bisher wenig untersuchte Anforderung an Deduktionssysteme ist die Präsentation gefundener Beweise in für Menschen verständlicher Form. Im Gebiet des automatischen Beweisens sind Übersetzungsverfahren entwickelt worden, die Beweise aus verschiedenen maschinenorientierten Formalismen in Beweise des natürlichen Schließens (ND-Beweise) transformieren [Lin90, PN90], aber leider führten diese Untersuchungen nicht zu wirklich verständlichen, natürlichen Beweisen. Im Gebiet der Generierung natürlicher Sprache wurde untersucht, wie ND-Beweise direkt verbalisiert werden können [Che76, McD83], wobei sich jedoch herausgestellt hat, daß eine direkte Verbalisierung von ND-Beweisen keinen adäquaten Text produziert, da die durch die jeweiligen ND-Inferenzregeln begründeten Ableitungen (im Vergleich zu Ableitungen, wie man sie in typischen mathematischen Lehrbüchern findet) auf einer viel zu niedrigen Abstraktionsebene liegen.
Wir haben deshalb einen rekonstruktiven Ansatz zu diesem Problem gewählt, in dem die Präsentation von einem ND-Beweis vom PROVERB-System in zwei Stufen durchgeführt wird [Hua94c]:

- Das System sucht zuerst nach einer neuen Repräsentation des Beweises, die abstraktere Ableitungsschritte enthält [Hua94d] und besser strukturiert ist.

- In einer zweiten Stufe verbalisiert dann die Präsentationskomponente den so abstrahierten Beweis [Hua94a].

4.1 Ein Modell für informelles mathematisches Beweisen

Dieses Verarbeitungsmodell definiert eine Zwischenrepräsentation, die einen geeigneteren Ausgangspunkt für die Präsentation der Beweise darstellt; insbesondere müssen dabei Ableitungsoperatoren definiert werden, die abstrakter sind als die üblichen logischen ND-Inferenzregeln.
Sieht man sich die Beweise in mathematischen Lehrbüchern an, so stellt man fest, daß die meisten dort präsentierten Ableitungen durch die Anwendung eines *Faktums* (das heißt einer Definition, eines Axioms oder eines bewiesenen Theorems) und nicht allein durch einen einzelnen logischen Inferenzschritt begründet sind. Zum Beispiel kann man aus den Prämissen $U_1 \subset F_1$ und $a_1 \in U_1$, die Schlußfolgerung $a_1 \in F_1$ mittels einer Anwendung der Teilmengendefinition ableiten. Logisch gesehen erklären wir die Anwendung eines Faktums durch die Existenz eines Teilbeweisbaums. Mit solchen Bäumen kann man zu jedem Faktum eine endliche Menge von *bereichspezifischen* Inferenzregeln erzeugen, die alle Fälle der Anwendung dieses Faktums abdeckt.
Als eine neue Zwischenrepräsentation spielen *Faktenbeweise* eine entscheidende Rolle in unserem Ansatz zur Beweispräsentation. Bevor die Präsentationstechniken der Sprachgenerierung in der zweiten Stufe angewendet werden können, müssen also zuerst aus maschinell erzeugten ND-Beweisen abstraktere Beweise erzeugt werden, deren Ableitungen zum großen Teil aus der Anwendung von Fakten bestehen. Beispielsweise wird der folgende ND-Beweisabschnitt

$$\text{(i)} \quad \cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\forall_{F,U} U \subset F \Leftrightarrow \forall_x x \in U \Rightarrow x \in F}{U_1 \subset F_1 \Leftrightarrow \forall_x x \in U_1 \Rightarrow x \in F_1}\forall D}{U_1 \subset F_1 \Rightarrow \forall_x x \in U_1 \Rightarrow x \in F_1}\Leftrightarrow D \quad U_1 \subset F_1}{\forall_x x \in U_1 \Rightarrow x \in F_1}\Rightarrow D}{a_1 \in U_1 \Rightarrow a_1 \in F_1}\forall D \quad a_1 \in U_1}{a_1 \in F_1}\Rightarrow D$$

durch den folgenden Schritt auf der Faktenebene ersetzt:

$$\text{(ii)} \quad \frac{a_1 \in U_1, U_1 \subset F_1}{a_1 \in F_1}\text{Def-Subset}$$

Die praktischen Erfahrungen mit unserer Implementierung zeigen, daß ein breites Spektrum von ND-Beweisen auf diese Weise wesentlich gekürzt werden kann, der Faktor liegt bei dreifach bis zehnfach kürzeren Beweisen. Darüberhinaus entsprechen die Ableitungen auf der Faktenebene sehr gut der Intuition von Mathematikern, die mechanisch erzeugten Beweise waren in den von uns untersuchten Beispielen fast immer mit den im Lehrbuch angegebenen vergleichbar.

4.2 Ein Modell für die Beweispräsentation

Zur Sprachgenerierung soll nun ein Modell vorgestellt werden, wie aus einem Faktenbeweis eine Folge von Beweiskommunikationsaktionen (proof communicative acts, PCAs) erzeugt werden kann, die am Ende (nach einer Reihe linguistischer Transformationen) den natürlichsprachlichen Beweis erzeugen. Jede PCA entspricht einem Ableitungsschritt, in dem die Information über die benutzten Prämissen, über das neue Zwischenergebnis und über die Begründung enthalten sind. Im folgenden sehen wir eine PCA, die aus der obigen Ableitung (ii) erzeugt wird:

```
(Derive Reasons: (a_1 ∈ U_1, U_1 ⊂ F_1)
        Derived-Formula: a_1 ∈ F_1
        Method: Def-Subset)
```

Daraus erzeugt PROVERB die folgende Verbalisierung:

> "Because a is an element of S_1 and S_1 is a subset of S_2, according to the definition of subset, a is an element of S_2."

Die Präsentation eines Beweises wird teilweise durch *hierarchische Planung* und teilweise durch einen *lokalen Fokus* gesteuert. Im Modus der hierarchischen Planung muß das System zu jedem Zeitpunkt entscheiden:

- wie der zu präsentierende Beweis in Teilbeweise aufgespalten werden kann,
- in welcher Reihenfolge die Teilbeweise präsentiert werden,
- welche PCAs den Übergang von einem Teilbeweis zum nächsten vermitteln sollen und
- welche PCAs die primitiven Teilbeweise präsentieren sollen.

Für viele bekannte Beweistrukturen sind Standardkommunikationsmuster vorhanden, durch die solche Entscheidungen gefällt werden können; denn die mathematische Umgangssprache ist sehr stark ritualisiert. Solche Kommunikationsmuster werden durch top-down Planungsoperatoren dargestellt. Mit dem lokalen Fokus wird das Phänomen simuliert, daß im allgemeinen anhand des zuletzt vermittelten Zwischenergebnisses das nächste Zwischenergebnis ausgewählt wird. Logisch gesehen ist in einem Beweisbaum jeder Knoten, der den lokalen Fokus als eine Prämisse benutzt, ein solcher Kandidat. Bei der Konfliktauflösung wird derjenige Knoten bevorzugt, der mit dem lokalen Fokus die meisten semantischen Objekte gemeinsam hat. Das Kommunikationswissen darüber, wie der ausgewählte Knoten präsentiert werden soll, wird durch bottom-up Präsentationsoperatoren dargestellt.

Da die top-down Planungsoperatoren und die bottom-up Präsentationsoperatoren im selben Format kodiert sind, lassen sich die zwei Textgenerierungstechniken leicht in einem Planungsmechanismus miteinander verbinden.
Ein weiterer Schwerpunkt des Verarbeitungsmodells für die Beweispräsentation ist die Festlegung der Referenzausdrücke in den PCAs [Hua94b]. Hier müssen Entscheidungen getroffen werden, ob und wie sich das System auf die als Prämissen benutzten Zwischenergebnisse und die als Begründung benutzte Inferenzmethode beziehen soll. Die Verbalisierung der PCAs erfolgt schließlich durch das unter Leitung von Wahlster entwickelte linguistische System TAG-GEN [KF95] zur syntaktischen Generierung.

5 Ein Beispiel

In diesem Abschnitt soll gezeigt werden, wie eine mathematische Aussage in Ω–MKRP bewiesen werden kann. Als Beispiel haben wir die Aussage gewählt, daß die transitive Hülle der Vereinigung zweier Relationen gleich der transitiven Hülle der Vereinigung ihrer transitiven Hüllen ist:

$$(\rho \cup \sigma)^* = (\rho^* \cup \sigma^*)^*$$

Obwohl wir aus Darstellungsgründen ein recht einfaches Beispiel gewählt haben, läßt es sich mit den heutigen vollautomatischen Beweisern allein nicht direkt lösen.
Als erstes muß sich der Benutzer in der Datenbank die notwendigen Axiome, Definitionen und Hilfssätze auswählen, die für den Beweis notwendig sind bzw. die er benutzen will. In diesem Fall wählen wir die Definitionen der *transitiven Hülle*, der *Vereinigung* und der *Mengengleichheit*. Natürlich könnte diese Auswahl auch semiautomatisch auf der Basis der verwendeten Symbole erfolgen, und in der Tat wird das System diese Auswahl in Zukunft unterstützen. Bei der Definition der transitiven Hülle haben wir die Wahl zwischen zwei äquivalenten Definitionen. Wir wählen die Definition, die die transitive Hülle einer Relation σ als die kleinste transitive Relation beschreibt, die σ enthält. Während des Beweises werden wir später feststellen, daß wir zusätzlich noch die Definition für die *Teilmengenbeziehung* benötigen.
Nun können wir mit dem Beweisen beginnen und wenden als erstes die Definition der Gleichheit von Mengen auf das Beweisziel an. Hierbei handelt es sich um die Anwendung eines Faktums (siehe Abschnitt 4), der am häufigsten verwendeten Schlußweise in Ω–MKRP, für die natürlich eine Taktik vorhanden ist. Als Ergebnis erhalten wir eine Konjunktion, die wir in die beiden Hauptteilziele $(\rho \cup \sigma)^* \subseteq (\rho^* \cup \sigma^*)^*$ und $(\rho^* \cup \sigma^*)^* \subseteq (\rho \cup \sigma)^*$ aufspalten können.
Nach weiteren Anwendungen von Definitionen auf die erste Formel und anschließender Simplifikation erhalten wir die Formel $\rho \cup \sigma \subseteq (\rho^* \cup \sigma^*)^*$. An dieser Stelle ist eine spezielle Taktik anwendbar, die die Homomorphieeigenschaft von $\cup$ für $\subseteq$ ausnutzt (siehe Abschnitt. 3.2). Als Ergebnis erhalten wir die beiden Teilziele $\rho \subseteq (\rho^* \cup \sigma^*)^*$ und $\sigma \subseteq (\rho^* \cup \sigma^*)^*$. Nun haben wir die Problemstellung genügend zerlegt, so daß sie von einem der angeschlossenen automatischen Beweiser in angemessener Zeit gelöst

werden kann. Das zweite Hauptteilziel läßt sich ähnlich lösen, so daß wir vier Teilprobleme bekommen, mit denen wir nun alternativ die Beweissysteme MKRP oder OTTER aufrufen.
Bei diesen Aufrufen wird als erstes unsere Repräsentation des Beispiels, das ja wegen der obigen Relationendarstellung eine Logik höherer Stufe voraussetzt, von $\mathcal{POST}$ in eine Repräsentation der Logik erster Stufe übersetzt [Ker91a]. Dann werden die entstandenen Formeln in Klauselnormalform transformiert und dem gewählten Beweiser in dessen Syntax übergeben. Nachdem ein Beweis gefunden ist, wird das spezielle Ausgabeprotokoll des Beweisers in eine einheitliche Sprache übersetzt, wobei eventuell im Protokoll fehlende Informationen (wie zum Beispiel die angewendeten Substitutionen) wiederberechnet werden müssen. Dieser einheitlich repräsentierte Resolutionsbeweis wird dann in einen Beweis transformiert, der in dem in Ω–MKRP benutzten Format, dem Kalkül des natürlichen Schließens, formuliert ist. Anschließend wird dieser Beweis wieder von der Logik erster Stufe in die ursprüngliche Logik höherer Stufe übersetzt. Dieser Beweis kann nun in den Beweisbaum in Ω–MKRP eingefügt werden.
Nachdem wir die vier Teilprobleme in unserem Beispiel mit einem der automatischen Beweiser gelöst haben, erhalten wir einen vollständigen ND-Beweis. Dieser kann nun mit der oben beschriebenen Beweistransformation abstrahiert und in natürliche Sprache transformiert werden (siehe Abschnitt 4).

6 Zusammenfassung

Die Entwicklung, die zu dem beschriebenen Ω–MKRP-System geführt hat, steht natürlich nicht für sich allein, sondern muß im Grenzbereich zwischen automatischen Beweisern, Beweisprüfern und planbasierten Systemen zur Beweiserstellung gesehen werden. Damit wird im Rahmen dieses Systems versucht, verschiedene Paradigmen der Beweiserstellung zu verbinden. Automatische Beweiser bauen meist auf einem maschinenorientierten Kalkül wie dem Resolutionsverfahren auf: man spezifiziert zu Beginn einer Beweissuche ein Problem, setzt gewisse Parameter, und dann versucht das System ohne weitere Benutzerinteraktion einen Beweis zu finden. Die meisten automatischen Beweiser bauen auf einer Variante der Logik erster Stufe auf, es gibt aber zum einen spezielle Systeme zur Behandlung von vollständiger Induktion wie den Boyer-Moore-Beweiser [BM79] oder das INKA-System [BHHW86]. Zum anderen Systeme gibt es Systeme, die auf einer Logik höherer Stufe aufbauen, wie das TPS-System [AINP90]. Das letzte System hat sowohl einen automatischen als auch einen interaktiven Modus.
Systeme zur Beweisprüfung wie AUTOMATH [Bru80] wurden auch benutzt um mathematische Lehrbücher streng formal zu beweisen. Sie nutzen in stärkerem Umfang bereichsspezifisches Wissen aus, aber die eigentliche Beweislast liegt fast ausschließlich beim Benutzer. Fortgeschrittene Systeme sind taktikbasierte Systeme wie NUPRL [Con86], Isabelle [Pau90] (Isabelle erlaubt zusätzlich die Spezifikation der

Objektlogik) und KIV [HRS90] oder wissensbasierte Systeme wie Muscadet [Pas89] und Ontic [McA89]. Die Taktiken wurden im OYSTER-CLAM-System [BvHHS90] zu Methoden erweitert, so daß eine deklarative Spezifikation von Taktiken angegeben werden kann.
Wir sehen die wachsende Aufmerksamkeit, die dem Bereich des taktikbasierten und planbasierten Beweisen gewidmet wird, nicht als Zufall, sondern als ein Indiz dafür an, daß vollautomatische Beweiser zwar starke Hilfmittel, aber als eigenständige Systeme nur von einem begrenzten Nutzen sind. Sie entfalten ihre Stärke vor allem als integrierte Systeme in einer interaktiven Umgebung.

Literatur

[AINP90] P. B. Andrews, S. Issar, D. Nesmith, F. Pfenning. The TPS theorem proving system. In [Sti90], S. 641–642.

[And71] P. B. Andrews. Resolution in type theory. *Journal of Symbolic Logic*, 3(36):414–432, 1971.

[And89] P. B. Andrews. On connections and higher order logic. *Journal of Automated Reasoning*, 5:257–291, 1989.

[BHHW86] S. Biundo, B. Hummel, D. Hutter, C. Walther. The Karlsruhe induction theorem proving system. In J. H. Siekmann, Hrsg., *Proc. of the 8th CADE*, S. 672–674. Springer, 1986.

[BHP95] P. Baumgartner, R. Hähnle, J. Posegga, Hrsg. *Theorem Proving with Analytic Tableaux and Related Methods*. Springer, LNAI 918, 1995.

[BLM+86] R. Boyer, E. Lusk, W. McCune, R. Overbeek, M. Stickel, L. Wos. Set theory in first-order logic: Clauses for Gödel's axioms. *Journal of Automated Reasoning*, 2:287–327, 1986.

[BM79] R. S. Boyer, J. S. Moore. *A Computational Logic*. Academic Press, 1979.

[Bru80] N. G. d. Bruijn. A survey of the project AUTOMATH. In J. Seldin, J. Hindley, Hrsg., *To H.B. Curry - Essays on Combinatory Logic, Lambda Calculus and Formalism*, S. 579–606. Academic Press, 1980.

[Bun88] A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk, R. Overbeek, Hrsg., *Proc. of the 9th CADE*, S. 111–120. Springer, LNCS 310, 1988.

[Bun94] A. Bundy, Hrsg. *Proc. of the 12th CADE*. Springer, LNAI 814, 1994.

[BvHHS90] A. Bundy, F. van Harmelen, C. Horn, A. Smaill. The OYSTER-CLAM system. In [Sti90], S. 647–648.

[Che76] D. Chester. The translation of formal proofs into English. *AI*, 7:178–216, 1976.

[Chu40] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Con86] R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall, 1986.

[Deu71] P. Deussen. *Halbgruppen und Automaten.* Springer, 1971.

[Hen50] L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15:81–91, 1950.

[HKKR94] X. Huang, M. Kerber, M. Kohlhase, J. Richts. Adapting methods to novel tasks in proof planning. In [ND94], S. 379–390.

[HRS90] M. Heisel, W. Reif, W. Stephan. Tactical theorem proving in program verification. In [Sti90], S. 117–131.

[Hua94a] X. Huang. Planning argumentative texts. In M. Nagao, Hrsg., *Proc. of 15th International Conference on Computational Linguistics*, S. 329–333, 1994.

[Hua94b] X. Huang. Planning reference choices for argumentative texts. In D. McDonald, Hrsg., *Proc. of 7th International Workshop on Natural Language Generation*, S. 145–152, 1994.

[Hua94c] X. Huang. PROVERB: A system explaining machine-found proofs. In A. Ram, K. Eiselt, Hrsg., *Proc. of 16th Annual Conference of the Cognitive Science Society*, S. 427–432. Lawrence Erlbaum Associates, 1994.

[Hua94d] X. Huang. Reconstructing proofs at the assertion level. In [Bun94], S. 738–752.

[Hue72] G. P. Huet. *Constrained Resolution: A Complete Method for Higher Order Logic.* Dissertation, Case Western Reserve University, 1972.

[JK94] P. Johann, M. Kohlhase. Unification in an extensional lambda calculus with ordered function sorts and constant overloading. In [Bun94], S. 620–634.

[Ker91a] M. Kerber. How to prove higher order theorems in first order logic. In J. Mylopoulos, R. Reiter, Hrsg., *Proc. of the 12th IJCAI*, S. 137–142. Morgan Kaufman, 1991.

[Ker91b] M. Kerber. On the representation of mathematical knowledge in frames and its consistency. In M. De Glas, D. Gabbay, Hrsg., *Proc. of the First World Conference on the Fundamentals of AI*, S. 293–301. Angkor, 1991.

[Ker94] M. Kerber. On the translation of higher-order problems into first-order logic. In T. Cohn, Hrsg., *Proc. of ECAI-94*, S. 145–149. John Wiley & Sons, 1994.

[KF95] A. Kilger, W. Finkler. TAG-based incremental generation. *Computational Linguistics*, 1995. Im Druck.

[KK94] M. Kerber, M. Kohlhase. A mechanization of strong Kleene logic for partial functions. In [Bun94], S. 371–385.

[KK95] M. Kerber, M. Kohlhase. A tableau calculus for partial functions. In *Annals of the Kurt-Gödel-Society.* Springer, 1995. Im Druck.

[Kle52] S. C. Kleene. *Introduction to Metamathematics.* Van Nostrand, 1952.

[Koh94] M. Kohlhase. *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle.* Dissertation, Universität des Saarlandes, 1994.

[Koh95] M. Kohlhase. Higher-Order Tableaux. In [BHP95], S. 294–309.

[KP93] M. Kohlhase, F. Pfenning. Unification in a λ-calculus with intersection types. In D. Miller, Hrsg., *Proc. of the International Logic Programming Sympsion. ILPS'93*, S. 488–505. MIT Press, 1993.

[Ler83] U. Leron. Structuring mathematical proofs. *The American Mathematical Monthly*, 90:174–185, 1983.

[Lin90] C. Lingenfelder. *Transformation and Structuring of Computer Generated Proofs*. Dissertation, Universität Kaiserslautern, 1990.

[McA89] D. A. McAllester. *ONTIC – A Knowledge Representation System for Mathematics*. MIT Press, 1989.

[McD83] D. D. McDonald. Natural language generation as a computational problem. In *Brady/Berwick: Computational Models of Discourse*. MIT Press, 1983.

[Mel95a] E. Melis. A model of analogy-driven proof-plan construction. In *Proc. of the 14th IJCAI*. Morgan Kaufman, 1995. Im Druck.

[Mel95b] E. Melis. Proving a Heine-Borel theorem by analogy. In *Proc. of the Symposium on Abstraction, Reformulation, and Approximation*, 1995. Im Druck.

[ND94] B. Nebel, L. Dreschler-Fischer, Hrsg. *KI-94: Advances in Artificial Intelligence – Proc. of KI-94*. Springer, LNAI 861, 1994.

[NM88] G. Nadathur, D. Miller. An overview of λ-prolog. In *Logic Programming 5*, S. 810–827, 1988.

[Owe90] S. Owen. *Analogy for Automated Reasoning*. Academic Press, 1990.

[Pas89] D. Pastre. Muscadet: An automated theorem proving system using knowledge and metaknowledge in mathematics. *Artificial Intelligence*, 38(3):257–318, 1989.

[Pau90] L. C. Paulson. Isabelle: The next 700 theorem provers. *Logic and Computer Science*, S. 361–386, 1990.

[Pfe91] F. Pfenning. Logic programming in the LF logical framework. In G. P. Huet, G. D. Plotkin, Hrsg., *Logical Frameworks*. Cambridge University Press, 1991.

[PN90] F. Pfenning, D. Nesmith. Presenting intuitive deductions via symmetric simplification. In [Sti90], S. 336–350.

[Qua92] A. Quaife. Automated deduction in von Neumann-Bernays-Gödel set theory. *Journal of Automated Reasoning*, 8(1):91–148, 1992.

[SK93] B. Schütt, M. Kerber. A mathematical knowledge base for proving theorems in semigroup and automata theory, part I. SEKI Working Paper SWP-93-02, Fachbereich Informatik, Universität des Saarlandes, 1993.

[Slo92] O. Slotosch. *Analogieschlüsse beim automatischen Beweisen*. Dr. Kovac Verlag, 1992.

[SS89] M. Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*. Springer, LNAI 395, 1989.

[Sti90] M. E. Stickel, Hrsg. *Proc. of the 10th CADE*. Springer, LNAI 449, 1990.

[Wei91] C. Weidenbach. A sorted logic using dynamic sorts. Technischer Bericht MPI-I-91-218, Max-Planck-Institut für Informatik, 1991.