Analogie im Beweisplanen

Diplomarbeit
von
Carsten Ullrich
`cullrich@ags.uni-sb.de`

Angefertigt nach einem Thema von Prof. Dr. Jörg Siekmann
am Fachbereich Informatik der Universität des Saarlandes

3. August 2000 (inzwischen um einige Tippfehler korrigiert)

Abstract

Analogical reasoning consists of transferring the solution of a previously solved problem (called source) onto a new problem (called target). In this thesis I describe TOPAL, an approach for the analogical transfer of mathematical proofs. The transfer is performed at the level of proof plans, therefore the application of abstract planning operators, called methods, is replayed in the new problem rather than the application of single calculus steps.

In order to cope with the complexity of proofs at plan level, I had to extent existing analogy frameworks in several ways:

Firstly, as in proof planning a standard higher order matcher can no longer compare source and target terms, I propose extensions to higher order matching that make these comparisons possible again.

Secondly, the underlying assumption of most analogy systems, namely that the differences in term structure expressed via the matching can be used to guide the transfer and adaption of the source plan, is not valid in proof planning. Therefore I show how planning mechanisms can be used to fulfil the same purpose of guiding transfer and adaptions.

Thirdly, I propose how theorem proving by analogy can be realised as one of several problem solving strategies. I identify situations in which the use of analogy makes sense and present the corresponding strategic control knowledge.

Finally, I have evaluated TOPAL. The evaluation shows that the use of planning mechanisms enhances the performance of the analogical transfer. By using these mechanisms a great number of complex problems can be solved.

Hiermit versichere ich an Eides statt, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Saarbrücken, 3. August 2000

Danksagungen

Mein Dank geht an Professor Jörg Siekmann, der es mir ermöglichte, diese interessante Diplomarbeit zu schreiben und an meine Betreuerin Erica Melis, die auch nach stundenlanger Diskussionen immer ein offenes Ohr für mich hatte.

Zu Dank verpflichtet bin ich auch Michael Kohlhase und Volker Sorge für die allerneusten LaTeX- und Emacstricks, Karsten Konrad für eine Einführung in Unifikation höherer Ordnung und natürlich den gesamten Mitgliedern der Ωmega-Gruppe für ihre tatkräftige Unterstützung.

Für das geduldige Korrekturlesen meiner Diplomarbeit bedanke ich mich zudem bei Erica Melis, Michael Kohlhase, Jürgen Zimmer und Daniel Dehnhard.

Ich möchte mich auch bei meinen Eltern bedanken, ohne die mein Studium nicht möglich gewesen wäre.

Nicht zuletzt bedanke ich mich bei meiner Freundin Kerstin Borau, die mich unermüdlich unterstützt und motiviert hat.

Inhaltsverzeichnis

Kapitel 1

Einführung

Pólya gibt uns in seinem Buch „How to solve it“ folgenden Ratschlag:

> *We know, of course, that it is hard to have a good idea if we have little knowledge of the subject, and impossible to have it if we have no knowledge. Good ideas are based on past experience and formerly acquired knowledge. Mere remembering is not enough for a good idea, but we cannot have any good idea without recollecting some pertinent facts; materials alone are not enough for constructing a house but we cannot construct a house without collecting the right materials. The materials necessary for solving a mathematical problem are certain relevant items of our formerly acquired knowledge, as formerly solved problems, or formerly proved theorems. Thus, it is often appropriate to start the work with the question: Do you know a related problem? [...] If we succeed in recalling a formerly solved problem which is closely related to our present problem, we are lucky. We should try to deserve such luck; we may deserve it by exploiting it.*

Unser Glück auf diese Weise auszunutzen, wird als *Problemlösen durch Analogie* bezeichnet: Ein gelöstes Problem (die *Quelle*) wird verwendet, um ein neues Problem (das *Ziel*) zu lösen. In der Mathematik wird häufig auf das Problemlösen durch Analogie zurückgegriffen, oftmals wird auf dieses Vorgehen auch explizit hingewiesen, wie in diesem Beweis aus dem Lehrbuch für Analysis von Bartle und Sherbert [BS92]:

> **Satz 3.2.6:** *Ist* $X = (x_n)$ *eine konvergente Folge und gilt* $a \leq x_n \leq b$ *für alle* $n \in \mathbb{N}$ *und fixe* $a, b \in \mathbb{R}$*, dann gilt auch* $a \leq \lim(x_n) \leq b$.
> **Beweis:** *Sei* Y *die konstante Folge* $(b, b, b, \ldots)$*. Nach Satz 3.2.5 folgt, daß* $\lim(X) \leq \lim(Y) = b$ *gilt.* Analog *kann man zeigen, daß* $a \leq \lim(X)$ *gilt.*

Ein anderes Beispiel findet man in dem Buch über Halbgruppen und Automaten von Deussen [Deu71]. Zuerst wird folgender Satz bewiesen:

> **Satz 4.8:** *Sind* ρ, σ *zwei Äquivalenzrelationen, so ist* $\rho \cap \sigma$ *eine Äquivalenzrelation.*

Einige Seiten später folgt:

> *In* Analogie *zu Satz 4.8 gilt:*
> **Satz 5.3:** *Sind* ρ, σ *Links- oder Rechtskongruenzen oder Kongruenzen, so ist* $\rho \cap \sigma$ *eine Links- bzw. Rechtskongruenz bzw. Kongruenz.*

Der Beweis des zweiten Satzes wird vom Autor nicht mehr angegeben, da er analog zum ersten Beweis geführt werden kann.

Diese beiden Beispiele stehen für zwei verschiedene Arten von Analogie: *interne* und *externe* Analogie. Wird eine Teillösung innerhalb eines Problems verwendet, wie im Beweis des Satzes 3.2.6, so spricht man von interner Analogie; wird die Lösung eines anderen Problems übertragen, wie für Satz 4.8, so spricht man von externer Analogie. Das Übertragen der Quelllösung nennt man auch *Transfer*. Abbildung 1.1 zeigt eine graphische Versinnbildlichung des Analogieprozesses.

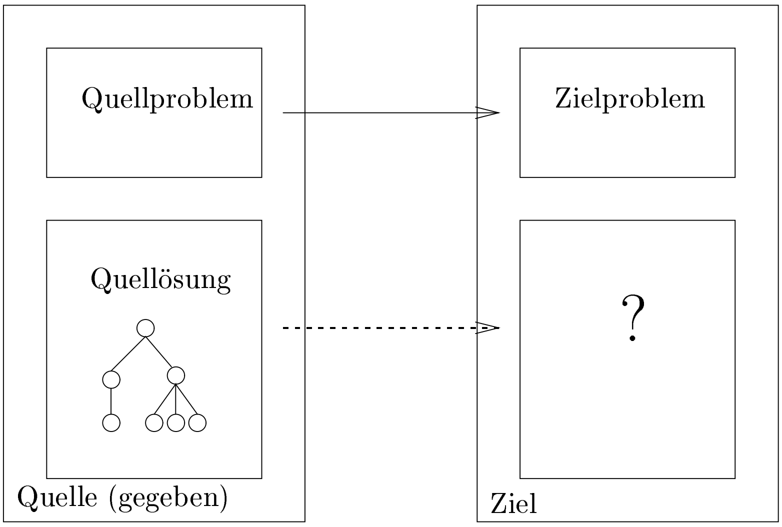


Abbildung 1.1: Problemlösen durch Analogie

Das Ziel dieser Diplomarbeit ist die Realisierung eines Analogiesystems, das es erlaubt, Probleme aus der Mathematik durch Analogie zu lösen. Mein System TOPAL (*T*ransfer *O*f *P*roofs at Pl*a*n *L*evel) ist eine Erweiterung des Beweisplaners des ΩMEGA-Systems, einem mathematischen Assistenzsystem [BCF$^+$97]. ΩMEGA unterstützt seinen Benutzer beim Entwickeln formaler Beweise mathematischer Sätze. Eine zentrale Stellung innerhalb von ΩMEGA nimmt die *wissensbasierte Beweisplanung* [MS99] ein. In der wissensbasierten Beweisplanung wird ein allgemeiner Planungsalgorithmus verwendet, der abhängig von dem zu beweisenden Problem theoriespezifisches Wissen nutzt: Methoden, die abstrakten mathematische Vorgehensweisen entsprechen; Kontrollwissen, das die sinnvolle automatische Anwendung der Methoden ermöglicht; und externe Systeme (wie zum Beispiel Constraintlöser), die Spezialfähigkeiten entsprechen (wie das Sammeln von Einschränkungen und die Konstruktion von Objekten, die diese Einschränkungen erfüllen).

Wissensbasierte Beweisplanung erlaubt es, Beweise auf einer abstrakten, dem menschlichen mathematischen Vorgehen entsprechenden Ebene zu führen. Auf dieser Ebene können auch menschliche *Problemlösestrategien* nachgebildet werden, wie eben die Analogie.

TOPAL erweitert das ΩMEGA-System um eine Analogiekomponente, die es ermöglicht, Beweise auf Planebene zu übertragen und, wenn nötig, anzupassen. Sowohl das Beweisen durch externe Analogie (TOPAL-X) als auch durch interne Analogie (TOPAL-I) ist möglich. Vorarbeiten zu dieser Diplomarbeit wurden in [MU99b] und [MU99a] veröffentlicht.

Gliederung dieser Arbeit In Kapitel 2 stelle ich die Grundlagen des Beweisplanens vor und gebe eine Übersicht über andere Analogiesysteme. Anschließend werde ich in Kapitel 3 auf eine zentrale Funktion der Analogie eingehen, dem Matchen, und zeigen, inwieweit es für die Analogie im Beweisplanen erweitert werden muß. Wie externe Analogie im Beweisplanen durch TOPAL-X realisiert wurde, zeigt Kapitel 4, interne Analogie (TOPAL-I) zeigt Kapitel 5. Der Einsatz der Analogie als Problemlösestrategie, ist Thema von Kapitel 6. Eine Zusammenfassung der Resultate, die wir mit mit dem Einsatz von TOPAL erzielt haben, folgt in Kapitel 7. Mit einer Zusammenfassung und einem Ausblick auf mögliche Erweiterungen in Kapitel 8 endet diese Diplomarbeit. Der Anhang enthält die von mir kommentierte Ausgabe von TOPAL während des Transfers eines Beispiels.

Kapitel 2

Grundlagen

> Calculemus – Rechnen wir es einfach aus.
>
> *Gottfried Wilhelm Leibniz*
> *Projet et Essais pour arriver à quelque certitude pour finir une bonne partie des disputes et pour avancer l'art d'inventer*

2.1 Logische Grundlagen

In diesem Abschnitt werde ich darauf eingehen, wie sich mathematische Aussagen formalisieren lassen, und wie über diesen Aussagen Beweise geführt werden können.

2.1.1 Formalisierung

Bereits Leibniz [Lei86] hatte das Ziel, natürlichsprachliche Beschreibungen von Sachverhalten in eine formale Sprache (*lingua characteristica*) zu übersetzen und durch einen dazugehörigen Kalkül (*calculus raticinator*) zu überprüfen. Streitigkeiten könnten dann einfach übersetzt und ausgerechnet werden.

Zweihundert Jahre später definierte Frege mit seiner *Begriffsschrift* eine logische Sprache, die wir heute als *Prädikatenlogik erster Stufe* bezeichnen. In Prädikatenlogik erster Stufe ist es bereits möglich, mathematische Objekte darzustellen, allerdings sind gewisse Konzepte nur sehr umständlich ausdrückbar. In der Zermelo-Fraenkelschen Axiomatisierung der Mengentheorie [Zer08] müßte beispielsweise das Konzept *Funktion* wenig intuitiv als linkstotale, rechtseindeutige Relation kodiert werden. Zudem lassen sich in Prädikatenlogik Paradoxien, wie die Menge aller Mengen die nicht sich selbst enthalten (Russels Paradoxon), formalisieren.

Eine Möglichkeit, Konstrukte für komplexe Objekte zur Verfügung zu stellen und Paradoxien auszuschließen, bieten *Typen* [Rus08, Chu40]:

Typ Der *Typ* eines Objektes orientiert sich an der Semantik dieses Objektes. Ausgehend von einer Menge von Basistypen $\mathcal{T}_0$ (zum Beispiel o für Wahrheitswerte, ι für Individuen, num für natürliche Zahlen usw.), ist die Menge von Typen $\mathcal{T}$ induktiv als die kleinste Menge definiert, für die gilt:

1. $\mathcal{T}_0 \subset \mathcal{T}$,
2. wenn $\alpha, \beta \in \mathcal{T}$, dann $(\alpha \rightarrow \beta) \in \mathcal{T}$.

Der Typ $(\alpha \rightarrow \beta)$ ist der einer Funktion von Objekten des Typs α auf Objekte des Typs β.

Signatur Eine *Signatur* ist eine Menge $\Sigma = \bigcup_\alpha \Sigma_\alpha^{const} \cup \bigcup \Sigma_\alpha^{var}$, wobei jede Menge $Sigma_\alpha^{const}$ eine (möglicherweise leere) Menge von Konstantensymbole vom Typ α und jede Menge Σ_α^{var} eine abzählbar unendliche Menge von Variablensymbolen vom Typ α ist. Die Mengen Σ_α seien disjunkt.

Nachdem eine Signatur festgelegt wurde, kann die Menge der wohlgeformten getypten *Terme* über die folgenden Syntaxregeln definiert werden:

Syntax Sei Σ eine Signatur. Die Syntaxregeln der einfachen Typtheorie lauten:

1. Jedes Variablen- und jedes Konstantensymbol vom Typ α aus Σ ist ein Term vom Typ α.

2. Ist $a_{\beta\rightarrow\alpha}$ ein Term vom Typ $(\beta \rightarrow \alpha)$ und b_β ein Term vom Typ β, so ist $(a_{\beta\rightarrow\alpha} b_\beta)$ ein Term vom Typ α (genannt Applikation).

3. Ist a_α ein Term vom Typ α und x_β eine Variable vom Typ β, so ist $(\lambda x_\beta . a_\alpha)$ ein Term vom Typ $(\beta \rightarrow \alpha)$ (genannt Abstraktion).

4. Sind a_o und b_o Terme vom Typ o und x_{v_α} eine Variable vom Typ α, so sind $\neg a_o, \forall x_{v_\alpha} a_o, \exists x_{v_\alpha} a_o, a_o \wedge b_o, a_o \vee b_o, a_o \rightarrow b_o$ und $a_o \equiv b_0$ Terme vom Typ o.[1] Terme von Typ o heißen Formeln; $\neg, \forall, \exists, \vee, \wedge, \rightarrow$ und $\equiv$ werden auch logische Symbole genannt.

Variablen lassen sich in *gebundene* und *freie* Vorkommen unterscheiden:

Gebundene/freie Vorkommen Ein Vorkommen einer Variablen x in einem Term t ist *gebunden*, wenn t einen Teilterm der Form $\lambda x.t'$ enthält. Ein Vorkommen einer Variablen x in einem Term t ist *frei*, wenn x ein Teilterm von t ist, aber nicht gebunden ist.

Welcher Art sind nun die Auswirkungen von Typen? Russells Paradoxon läßt sich zum Beispiel folgendermaßen formalisieren:

$$\forall Q.M(Q) \equiv Menge(Q) \wedge \neg Q(Q).$$

In dieser Formalisierung müßte Q sowohl den Typ $\alpha \rightarrow o$ als auch den Typ α haben, da Q sowohl als Prädikats- als auch als Individuenvariable verwendet wird. Dies ist aber nicht möglich. Es hat sich herausgestellt, daß durch Typen alle bekannten Arten von Paradoxien verhindert werden können.

Um die Darstellung klarer zu halten, werde ich im folgenden in der Regel die Typinformation nicht mit angeben und folgende Schreibweisen benutzen:

- F, G repräsentieren freie Variablen,

---

[1] Die Quantoren $(\forall, \exists)$ können mit Hilfe von λ definiert werden (siehe [And86]), ich habe sie aber der Übersicht halber mit angegeben.

- x, y, z stehen für gebundene Variablen,
- $f, g, +, a, b, c$ stehen für (Funktions)Konstanten,
- t für beliebige Terme,
- α, β für Typen.

Terme wie $\lambda x_1, \ldots, x_n.(a, x_1, \ldots, x_n)$ schreibe ich nach der Schreibweise der Logik erster Stufe als $\lambda x_1, \ldots, x_n.a(x_1, \ldots, x_n)$. Desweiteren kürze ich eine Folge von syntaktischen Objekten $s_1, \ldots, s_n$, mit $n \geq 0$, als $\overline{s_n}$ ab. Terme wie $\lambda \overline{x_k}.a(f_1(\overline{x_k}), \ldots, f_n(\overline{x_k}))$ werde ich als $\lambda \overline{x_k}.a(\overline{f_n(\overline{x_k})})$ schreiben.

Für das Matchen werden die Konzepte *Substitution* und *$\alpha\beta\eta$-Gleichheit* benötigt:

Substitution Eine Substitution ist eine endliche Abbildung von Variablen auf Terme und wird geschrieben als $\{\overline{x_n \mapsto t_n}\}$. Die Menge der Variablen x_i mit $1 \leq i \leq n$ wird Domäne D einer Substitution genannt. $\{x \mapsto t\}s$ steht für den Term, den man erhält, wenn jedes freie Vorkommen von x in s durch t ersetzt wird.

Gleichheit Die folgenden drei Axiome bestimmen Eigenschaften der Typtheorie bezüglich der Gleichheit:

> α-Axiom: $\lambda x.t = \lambda y.(\{x \mapsto y\}t)$, wenn y nicht in t vorkommt.
>
> β-Axiom: $(\lambda x.s)t = \{x \mapsto t\}s$
>
> η-Axiom: $\lambda x.(tx) = t$, wenn x nicht in t vorkommt.

Das α-Axiom besagt, daß der Name von gebundenen Variablen irrelevant ist. Das β-Axiom beschreibt das Verhalten von Funktionen, die durch Abstraktion gebildet werden. Das η-Axiom ist ein Spezialfall der Extensionalität, welche besagt, daß zwei Funktionen gleich sind, wenn sie auf allen möglichen Argumenten übereinstimmen.

Zwei Terme sind $\alpha\beta\eta$-gleich, wenn sie bezüglich der $\alpha\beta\eta$-Axiome gleich sind.

Das β- und η-Axiom können zu Regeln gerichtet werden, die den linken Term der Gleichung in den rechten überführen. Wir sprechen dann von β- bzw. η-Reduktion. Nach dem *Church-Rosser-Theorem* terminiert jede Kette von $\beta\eta$-Reduktionen eines Terms t und überführt t in eine eindeutige $\beta\eta$-Normalform. Das α-Axiom kann nicht gerichtet werden, da jede Umbenennung der gebundenen Variablen durch die inverse Umbenennung wieder rückgängig gemacht werden kann. Im folgenden betrachten wir Terme in $\beta\eta$-Normalform.

2.1.2 Unifikation und Matchen

1963 entwickelte Robinson den *Resolutionskalkül*, die erste effiziente Semientscheidungsprozedur für Prädikatenlogik erster Stufe [Rob65]. Eine zentrale Komponente des Resolutionsprinzip ist die *Unifikation*. Heute ist sie fester Bestandteil der Logik und Informatik und in fast allen Deduktionskalkülen zu finden. Für eine ausführliche Darstellung der Unifikation siehe zum Beispiel [Sny91].

Unifikation beschäftigt sich mit der Frage, ob es für zwei Terme t_1 und t_2 eine Substitution σ für die freien Variablen von Term t_1 und t_2 gibt, so daß $\sigma(t_1)$ und $\sigma(t_2)$ bezüglich der $\alpha\beta\eta$-Gleichheit gleich sind.

Im Beweisen durch Analogie wird eine eingeschränkte Version der Unifikation verwendet, das *Matchen*:

Matchen ist einseitige Unifikation, das heißt für zwei Terme t_1 und t_2 soll eine Substitution σ für die freien Variablen von Term t_1 gefunden werden, so daß $\sigma(t_1)$ und t_2 bezüglich der $\alpha\beta\eta$-Gleichheit gleich sind.

Beispiel 2.1: Seien F, x Variablen vom Typ $\alpha \to \beta$ bzw. α und g, c Konstanten vom Typ $\alpha \to \beta$ bzw. α. Die möglichen Substitutionen, die $F_{\alpha\to\beta}(x_\alpha)$ auf $g_{\alpha\to\beta}(c_\alpha)$ matchen, sind $\{F_{\alpha\to\beta} \mapsto \lambda z_\alpha.g_{\alpha\to\beta}(c_\alpha)\}$, $\{F_{\alpha\to\beta} \mapsto \lambda z_\alpha.g_{\alpha\to\beta}(z_\alpha), x_\alpha \mapsto c_\alpha\}$ und $\{F_{\alpha\to\beta} \mapsto \lambda z_\alpha.z_\alpha, x_\alpha \mapsto g_{\alpha\to\beta}(c_\alpha)\}$.

Im Gegensatz zur Logik erster Stufe kann es also mehrere allgemeinste Lösungen geben.

Ein *Matchproblem* ist folgendermaßen definiert:

Matchproblem Eine Termpaarmenge

$$\mathcal{T} := \{t_1 \overset{?}{\mapsto} t'_1, \ldots, t_n \overset{?}{\mapsto} t'_n\}$$

heißt *Matchproblem*, wenn die $t_i \overset{?}{\mapsto} t'_i$ jeweils den gleichen Typ haben. Ein Matchproblem heißt *gelöst*, wenn alle Termpaare von der Form $x_i \overset{?}{\mapsto} t'_i$ sind, so daß die Variable x_i nicht in t'_i vorkommt. Ist $\mathcal{T}$ ein Matchproblem in gelöster Form, dann ist $\sigma_{\mathcal{T}} := \{x_i \mapsto t'_i\}$ ein *allgemeinster Matcher* für $\mathcal{T}$. Ein Matchalgorithmus läßt sich durch *Transformationsregeln* angeben:

Transformationsregel Eine Transformationsregel hat die Form

$$\{t_1 \overset{?}{\mapsto} t'_1\} \cup S \Rightarrow S'$$

und überführt das Matchproblem $\{t_1 \overset{?}{\mapsto} t'_1\} \cup S$ in das Matchproblem S'.

Matchalgorithmus Aus einer Menge von Transformationsregeln erhält man einen Matchalgorithmus, wenn man den Suchraum, der durch diese Regelmenge aufgespannt wird, systematisch durchsucht. Ausgehend von einem initialen Matchproblem $\mathcal{T}$ werden dabei die Transformationsregeln so lange angewandt, bis keine Regel mehr anwendbar ist. Das Matchen ist daher ein Prozeß des Vereinfachens von Matchproblemen. Tabelle 2.1 zeigt die Transformationsregeln für das Matchen höherer Ordnung.

Die ersten drei Regeln findet man in analoger Form auch im Matchen erster Ordnung. Regel eins drückt aus, daß für identische Terme kein Matchen nötig ist. Die Dekompositionsregel besagt, daß Applikationen dann aufeinander matchbar sind, wenn ihrer Teilterme jeweils aufeinander matchbar sind. Die Bindungsregel bindet eine auftretende Variable an den Matchpartner und ersetzt die weiteren Vorkommen der Variable durch diesen Partner.

(1) Trivial

$$\{t \overset{?}{\mapsto} t\} \cup S \Rightarrow S$$

(2) Dekomposition

$$\{\lambda\overline{x_k}.f(\overline{t_n}) \overset{?}{\mapsto} \lambda\overline{x_k}.f(\overline{t'_n})\} \cup S \Rightarrow \bigcup_{1 \leq i \leq n} \{\lambda\overline{x_k}.t_i \overset{?}{\mapsto} \lambda\overline{x_k}.t'_i\} \cup S$$

(3) Bindung

$$\{\lambda\overline{x_k}.f_v(\overline{x_k}) \overset{?}{\mapsto} \lambda\overline{x_k}.t\} \cup S \Rightarrow \{f_v \overset{?}{\mapsto} \lambda\overline{x_k}.t\} \cup \{f_v \mapsto \lambda\overline{x_k}.t\}S$$

wenn f_v nicht frei in $\lambda\overline{x_k}.t$ vorkommt.

(4a) Imitation

$$\begin{aligned} & \{\lambda\overline{x_k}.f_v(\overline{t_n}) \overset{?}{\mapsto} \lambda\overline{x_k}.f(\overline{t'_m})\} \cup S \\ \Rightarrow\ & \{f_v \overset{?}{\mapsto} \lambda\overline{x_n}.f(\overline{H_m(\overline{x_n})})\} \cup \{\lambda\overline{x_k}.f_v(\overline{t_n}) \overset{?}{\mapsto} \lambda\overline{x_k}.f(\overline{t'_m})\} \cup S \end{aligned}$$

(4b) Projektion

$$\begin{aligned} & \{\lambda\overline{x_k}.f_v(\overline{t_n}) \overset{?}{\mapsto} \lambda\overline{x_k}.f(\overline{t'_m})\} \cup S \\ \Rightarrow\ & \{f_v \overset{?}{\mapsto} \lambda\overline{x_n}.x_i(\overline{H_m(\overline{x_n})})\} \cup \{\lambda\overline{x_k}.f_v(\overline{t_n}) \overset{?}{\mapsto} \lambda\overline{x_k}.f(\overline{t'_m})\} \cup S \end{aligned}$$

Tabelle 2.1: Transformationsregeln für Matchen höherer Ordnung

Ordnung	Unfikationsproblem	
	Unifikation	Matchen
1	entscheidbar	
2	unentscheidbar [Gol81]	entscheidbar [Hue73]
3	unentscheidbar [Hue73]	entscheidbar [Dow94]
4	unentscheidbar	entscheidbar [Pad96]

Tabelle 2.2: Entscheidbarkeit von Unfikationsproblemen

Wie bereits erwähnt, gibt es, im Gegensatz zum Matchen erster Ordnung, beim Matchen höherer Ordnung oft mehrere Lösungen. Der Grund dafür sind die in der Logik erster Stufe nicht auftretenden Funktionsvariablen. Für diese gibt es im allgemeinen mehrere mögliche Abbildungen. Die Transformationsregeln *Imitation* und *Projektion* sind im Fall von Funktionsvariablen anwendbar. Bei der Imitation wird die Funktionsvariable auf eine Konstante abgebildet, sie „imitiert“ sie. Bei der Projektion wird die Konstante auf eines der Argumente der Funktionsvariable abgebildet, gewissermaßen „projiziert“[2].

Beispiel 2.2: Seien F, x, g, c wie in Beispiel 2.1 definiert. Für das Matchproblem

$$\{F_{\alpha\to\beta}(x_\alpha)\overset{?}{\mapsto}g_{\alpha\to\beta}(c_\alpha)\}$$

ist eine Imitationslösung:

$$\{F_{\alpha\to\beta} \mapsto \lambda z_\alpha . g_{\alpha\to\beta}(z_\alpha), x_\alpha \mapsto c_\alpha\}$$

und eine Projektionslösung:

$$\{F_{\alpha\to\beta} \mapsto \lambda z_\alpha . g_{\alpha\to\beta}(c_\alpha)\}.$$

Theoretische Eigenschaften Wegen des Auftretens von Funktionsvariablen ist Unifikation bereits ab der zweiten Stufe unentscheidbar. Matchen ist

---

[2] Bei mehr als einem Argument wächst der Suchraum entsprechend, aufgrund der Wahl, auf welches Argument projiziert wird.

immerhin noch mindestens bis zur vierten Stufe entscheidbar. Glücklicherweise wird für die analogie-gesteuerte Beweisplankonstruktion nur Matching benötigt. Für eine Übersicht über die Entscheidbarkeit von Unifikationsproblemen verschiedener Ordnung siehe Tabelle 2.2.

2.1.3 Beweisführung

Nachdem im vorigen Abschnitt vorgestellt wurde, wie mathematische Aussagen formalisierbar sind, wird in diesem Abschnitt dargestellt, wie über diesen Aussagen Beweise geführt werden können.

Kalkül Ein Kalkül erweitert eine Logik um das syntaktische Konzept des Ableitens neuer Formeln aus gegebenen Formeln. Dazu definiert ein Kalkül eine Menge von *logischen Axiomen* und *Schlußregeln.*

Eine *Schlußregel* hat die Form:

$$\frac{F_1 \ldots F_n}{F}$$

Sie erlaubt, die neue Formel F aus den Formeln $F_1 \ldots F_n$ abzuleiten. Kann eine Formel F durch beliebig viele aufeinanderfolgende Anwendungen von Schlußregeln aus $F_1 \ldots F_n$ abgeleitet werden, schreibt man $F_1 \ldots F_n \vdash F$. $F_1 \ldots F_n \vdash F$ nennt man auch Urteil.

Die *logischen Axiome* stellen allgemeingültige Formeln des Kalküls dar. Zum Beispiel besagt das Axiom $F \vdash F$, daß eine Formel aus sich selbst abgeleitet werden kann.

Gerhard Gentzen entwickelte 1935 den *Kalkül des natürlichen Schließens*, der die in der Mathematik gebräuchlichen Schlußweisen nachbilden sollte [Gen35]. Beispielsweise stellen die Schlußregeln $\wedge E_L$ und $\wedge E_R$ die Aufspaltung einer Konjunktion in ihren linken bzw. rechten Konjunkt dar:

$$\wedge E_L : \ \frac{F_1 \wedge F_2}{F_1} \qquad \wedge E_R : \ \frac{F_1 \wedge F_2}{F_2}$$

Für eine komplette Übersicht über die Schlußregeln und Axiome siehe zum Beispiel [Pra65]. ΩMEGAs Kalkül $\mathcal{POST}$ [Nes94] basiert auf einer Variation dieses Kalküls.

Ableitung Eine *Ableitung* (oder auch *ND-Beweis*) ist eine endliche Folge von Urteilen, derart daß jedes Element der Folge entweder ein Axiom ist oder durch Anwendung einer Regel auf Elemente der Folge entsteht, die vor diesem in der Folge stehen. Eine Ableitung einer Formel F ist eine Ableitung, deren letztes Urteil die Form $\mathcal{F} \vdash F$ hat, wobei $\mathcal{F}$ entweder leer ist oder nur Axiome enthält.

Beweisdarstellung Durch die Ableitung einer Formel F erhält man einen Baum aus Urteilen, dessen Wurzel das Urteil $\mathcal{F} \vdash F$ ist und dessen Blätter aus Axiomen bestehen. Eine für längere Beweise übersichtliche linearisierte Darstellung dieser Ableitung läßt sich durch *Beweiszeilen* erhalten [And80]. Eine Beweiszeile hat die Form:

$$\mathit{Marke} \qquad Z_1 \dots Z_n \qquad \vdash \qquad F \qquad\qquad (\mathit{Regel\ Voraussetzungen})$$

Diese Darstellung besagt, daß sich die Formel F durch Anwendung der Schlußregel *Regel* auf die Zeilenliste *Voraussetzungen* aus den Annahmen $Z_1 \dots Z_n$ ableiten läßt. *Marke* wird im Rest des Beweises als Abkürzung der Formel F verwendet. Das Paar (*Regel Voraussetzung*) heißt *Begründung.* Zeilen, die noch nicht im Kalkül hergeleitet wurden, werden mit der Begründung *OPEN* gekennzeichnet. Eine Zeile L_1 ist *abhängig* von einer Zeile L_2, wenn im Laufe der Ableitung von L_1 aus den Annahmen L_2 abgeleitet wird.

Beweisproblem Als *Beweisproblem* wird eine zu beweisende mathematische Aussage bezeichnet. Ein Problem besteht aus der zu beweisenden Aussage, genannt *Konklusion*, und einer (möglicherweise leeren) Menge von *Annahmen* (zum Beispiel Definitionen oder Lemmas), die zur Herleitung der Konklusion verwendet werden können.

Ein einfaches Beispiel ist folgendes Problem:

ASS.	ASS	$\vdash (F_3 \land (F_2 \land F_1))$	(HYP)
CONC.	ASS	$\vdash F_1$	(*OPEN*)

In diesem Problem soll gezeigt werden, daß die Formel F_1 gilt, unter der Voraussetzung, daß die Annahme $(F_3 \wedge (F_2 \wedge F_1))$ gilt. Annahmen werden auch *Hypothesen* genannt, daher die Begründung *HYP*. Ein Beweis für dieses Problem im ND-Kalkül hat folgende Form:

ASS.	ASS	$\vdash (F_3 \wedge (F_2 \wedge F_1))$	(HYP)
L2.	ASS	$\vdash (F_2 \wedge F_1)$	($\wedge E_R$ ASS)
L3.	ASS	$\vdash F_2$	($\wedge E_L$ L2)
L1.	ASS	$\vdash F_3$	($\wedge E_L$ ASS)
CONC.	ASS	$\vdash F_1$	($\wedge E_R$ L2)

Wie man aus dieser Darstellung bereits erahnen kann, werden trotz seines Namens Beweise im Kalkül des natürlichen Schließens rasch umständlich und schwer verständlich. Eine Erleichterung bringt das Zusammenfassen mehrerer Schritte in einer *Taktik*:

Taktik Taktiken sind Prozeduren, die einen Teil des Beweises durch die Anwendung einer Folge von Schlußregeln erzeugen. Sie wurden zuerst im LCF-System eingesetzt [GMW79]. Beispielsweise spaltet die Taktik $\wedge E*$ eine Konjunktion direkt in alle Konjunkte auf. Eine Lösung des obigen Problems durch Anwendung der Taktik $\wedge E*$ hat folgende Form:

ASS.	ASS	$\vdash (F_3 \wedge (F_2 \wedge F_1))$	(HYP)
L2.	ASS	$\vdash F_2$	($\wedge E*$ ASS)
L1.	ASS	$\vdash F_3$	($\wedge E*$ ASS)
CONC.	ASS	$\vdash F_1$	($\wedge E*$ ASS)

Als Begründung wird der Name der Taktik eingesetzt. Bei der *Expansion* einer Taktik wird die Sequenz von Schlußregeln, die die Taktik darstellt, in den Beweis eingefügt. Beweisen mit Taktiken nennt man *taktisches Theorembeweisen.*

2.1.4 Automatisierung

Sind die logischen Grundlagen der Beweisformalisierung und Beweisführung gegeben, so ist es ein naheliegendes Ziel, die Suche nach einem Beweis zu automatisieren. Allerdings konnte Gödel 1931 zeigen, daß jedes hinreichend mächtige formale System, in dem sich mindestens die Arithmetik kodieren läßt, keinen vollständigen Kalkül besitzt. Es lassen sich immer Sätze konstruieren, die sich innerhalb dieses Systems weder beweisen noch widerlegen lassen. Außerdem ist, wie Church und Turing 1936 zeigten, die Erfüllbarkeit prädikatenlogischer Formeln bereits ab erster Stufe unentscheidbar.

Trotzdem rechtfertigt Herbrands Theorem von 1930 die Hoffnung, Beweise automatisch führen zu lassen. Er konnte zeigen, daß der Prädikatenkalkül semientscheidbar ist, d.h. ist ein Satz wahr, so kann dies in endlich vielen Schritten nachgewiesen werden. Ist der Satz nicht wahr, gelingt der Nachweis nur in Ausnahmefällen oder das Programm terminiert nicht.

Heute lassen sich zwei Richtungen des automatischen Beweisens unterscheiden, die den Computer in verschiedener Hinsicht nutzen: zum einen den Computer als außerordentlich leistungsfähigen Rechenschieber, zum anderen als Grundlage zur Nachahmung menschlicher Vorgehensweisen.

Maschinenorientieres Beweisen Das maschinenorientiere Beweisen verwendet einen maschinennahen Kalkül, wie zum Beispiel den Resolutionskalkül [Rob65]. Um einen Beweis zu finden, durchsucht ein Beweiser wie OTTER [McC90] blindlings in kürzester Zeit riesige Suchräume nach der richtigen Sequenz von Beweisschritten. Diese Systeme zeigen beeindruckende Leistungen, ihre maschinenorientierte Vorgehensweise bringt aber zwei gewichtige Mängel mit sich: Die Beweise sind für den Menschen unverständlich und menschliches mathematisches Wissen kann nicht verwendet werden.

Die andere Richtung des automatischen Beweisens nahm ihren Anfang im *Logic Theorist* [NSS57] und hat die Vorgehensweisen des Menschen zum Vorbild. Ihr Ziel ist die Nachbildung allgemeiner menschlicher heuristischer Verfahren und Vorgehensweisen. In dieser Tradition steht das *wissensbasierte Beweisplanen*.

2.2 Wissensbasiertes Beweisplanen

Im wissensbasierten Beweisplanen werden Techniken aus dem Teilgebiet der künstlichen Intelligenz, der sich mit dem automatischen *Planen* befaßt, auf das automatische Beweisen angewendet. Zuerst werde ich auf diese Techniken eingehen, um dann deren Einsatz im Beweisplanen zu erläutern.

2.2.1 Planen

Ein Planungsproblem besteht aus einem *Anfangszustand* und *Zielen*. In STRIPS [FN71], einem der ersten Planungssysteme, werden Planungszustände durch eine Menge prädikatenlogischer Formeln dargestellt. Während der Planerstellung versucht der Planer ausgehend vom Anfangszustand eine Folge von *Planungsoperatoren* (einen *Plan*) zu finden, die den Anfangszustand in einen Zustand umformt, in dem die Ziele gelten.

Planungsoperator Planungsoperatoren entsprechen verschiedenen Handlungen. In STRIPS besitzen sie *Vorbedingungen* und *Auswirkungen*: Vorbedingungen beschreiben die Formeln im Planungszustand, die erfüllt sein müssen, damit der Operator angewendet werden kann; Auswirkungen stellen die Veränderungen des Planungszustandes dar, die sich durch die Anwendung des Operators einstellen. Auswirkungen werden durch Listen von Literalen repräsentiert, die dem Planungszustand entnommen (**delete**-Liste) oder hinzugefügt (**add**-Liste) werden. Literale aus der **delete**- bzw. **add**-Liste werden mit $\ominus$ bzw. $\oplus$ markiert.

Planungsalgorithmus Welcher Operator und auf welche Literale dieser Operator angewendet wird, bestimmt der *Planungsalgorithmus*. Ein Planungsalgorithmus, der existierende Ziele durch neue Unterziele ersetzt, nennt man *rückwärtsschließend*; während ein *vorwärtschließender* Planungsalgorithmus ausgehend von geltenen Annahmen neue Annahmen ableitet.

Planungsschritt Einen instantiierten Planungsoperator nennt man *Planungsschritt*.

2.2.2 Beweisplanen

Im Beweisplanen werden die Techniken des Planens auf Beweisprobleme der Mathematik angewendet. Der Anfangszustand besteht aus einem Problem, also aus der *Konklusion*, und einer Menge von *Annahmen*. Der Zielzustand ist erreicht, wenn sich in dem aktuellem Planungszustand keine offene Zeile mehr befindet. Die Planungsoperatoren werden *Methoden* genannt.

Methoden Das Konzept der Methode stammt von Alan Bundy [Bun88]. Eine Methode enthält einen kleinen Teil des Wissens zur Lösung oder Vereinfachung mathematischer Probleme, beispielsweise wie eine Fallunterscheidung durchgeführt wird oder wie eine Ungleichung vereinfacht wird. Sie erlaubt es, abstraktes mathematisches Vorgehenswissen zu repräsentieren und in der Beweisplanung zu nutzen. Das Schlußfolgern geschieht nicht mehr auf Kalkülebene, sondern auf einer Ebene auf der menschliches mathematisches Wissen verwendet werden kann.

Im ΩMEGA-System besteht eine Methode aus den Slots *Prämissen*, *Konklusionen*, *Anwendungsbedingungen* und *Beweisschema*:

Die *Prämissen* (*premises*) und die *Konklusionen* (*conclusions*) geben die logische Bedeutung der Methode wieder, in dem Sinne, daß die Beweiszeilen, die der Konklusionen entsprechen, aus den Beweiszeilen, die den Prämissen entsprechen, ableitbar sein sollen. Die Prämissen und Konklusionen werden analog zu der STRIPS-Notation mit $\ominus$ bzw. $\oplus$ gekennzeichnet. Eine $\ominus$-Konklusion wird nach der Anwendung der Methode aus dem Planungszustand als offenes Ziel entfernt, eine $\oplus$-Prämisse wird als offenes Ziel eingefügt. Eine $\ominus$-Prämisse wird als Beweisannahme entfernt, eine $\oplus$-Konklusion als Annahme hinzugefügt. Eine ()-Zeile wird nicht geändert. Eine Methode, die eine offene Zeile entfernt und neue Teilziele einführt, heißt Rückwärtsmethode, eine Methode, die aus einer Annahme neue Annahmen ableitet, Vorwärtsmethode.

Die *Anwendungsbedingungen* (*appl-cond*) geben an, welche Voraussetzungen gelten müssen, damit eine Methode in dem jeweiligen Planungszustand auf eine bestimmte Menge von Zeilen anwendbar ist. In den Anwendungsbedingungen können externe System wie Computeralgebrasystem oder Constraintlöser verwendet werden.

Das *Beweisschema* (*proof schema*) enthält den schematischen Teilbeweis, dessen Instanz bei der Expansion der Methode in den Beweis eingefügt wird. Begründungen im dem Beweisschema können sowohl Schlußregeln und Taktiken als auch Aufrufe von externen Systemen sein.

Ein Beispiel für eine Methode zeigt Abbildung 2.1. Die Methode *Complex-Estimate<-m-b* zerlegt ein Ziel, das mit der Ungleichung $|b| < \epsilon$ matched in vier einfachere Teilziele mit Hilfe der Ungleichung $|a| < e_1$. Die Anwendungsbedingungen fordern, daß es eine Substitution σ und Terme k und l gibt, so daß b als Linearkombination von $\sigma(a)$, also als $b = k * \sigma(a) + l$ darstellbar ist. Das Beweisschema zeigt nur einen vereinfachten Teilbeweis, `fix` faßt eine Reihe von Schritten zusammen. Die Begründung `CAS` von Zeile L6 bedeutet, daß durch ein Computeralgebrasystem aus der Gleichung $b = b$ die Gleichung $|a_\sigma| < \epsilon/(2 * \mathbf{M})$ in einem Schritt abgeleitet wird. Bei der Expansion der Zeile L6 werden die einzelnen Umformungsschritte in den Beweis eingefügt.

Die Anwendungsbedingungen geben an, wann es erlaubt ist, eine Methode anzuwenden. Die Frage, wann es *sinnvoll* ist, eine bestimmte Methode anzuwenden, wird vom Kontrollwissen beantwortet. Dieses ist in Form von *Kontrollregeln* gegeben:

Kontrollregeln Kontrollregeln dienen dazu, die momentane Planungssituation zu analysieren und dementsprechend Hinweise auf das geeignete Vorgehen zu geben.

Sie werden durch einen Kontrollregelinterpreter an den Entscheidungsmöglichkeiten des Planungsprozesses ausgewertet, also bei der Wahl der Methode und der Wahl der Knoten, auf die die Methode angewendet werden soll. An diesen Stellen treffen sie eine Auswahl über die möglichen Alterna-

method: `ComplexEstimate<-m-b`(a, b, e_1, ϵ)			
premises	()L0 $\oplus$L1 $\oplus$L2 $\oplus$L3 $\oplus$L4		
conclusions	$\ominus$ L18		
appl-cond	$\exists\sigma.$`getsubst`$(a, b) = \sigma$ AND $\exists k, l.$`casextract`$(a_\sigma, b) =$ (k,l)		
proof schema	L0. Δ	$\vdash \|a\| < e_1$	(())
	L1. Δ	$\vdash \|k\| \leq \mathbf{M}$	(OPEN)
	L2. Δ	$\vdash \|a_\sigma\| < \epsilon/(2 * \mathbf{M})$	(OPEN)
	L3. Δ	$\vdash \|l\| < \epsilon/2$	(OPEN)
	L4. Δ	$\vdash 0 < \mathbf{M}$	(OPEN)
	L5.	$\vdash b = b$	(Ax)
	L6.	$\vdash b = k * a_\sigma + l$	(`CAS`;L5)
	L18Δ	$\vdash \|b\| < \epsilon$	(`fix`;L1, L2,L3,L4, L5,L6)

Abbildung 2.1: Die Abschätzungsmethode *ComplexEstimate<-m-b*

tiven.

Eine Kontrollregel besteht aus einem Bedingungs- und einem Aktionsteil. *Metaprädikate* im Bedingungsteil analysieren den Planungszustand und die Planungsgeschichte. Beispielsweise bestimmt das Metaprädikat `last-method` die zuletzt angewandte Methode. Ist der Bedingungsteil erfüllt, so wird der Aktionsteil ausgeführt. Folgende Aktionen stehen zum Sortieren der betreffenden Alternativen zur Verfügung: *select* wählt eine Alternative aus, *reject* entfernt eine Alternative und *prefer* bevorzugt eine Alternative gegenüber den restlichen.

Tabelle 2.6 zeigt eine Kontrollregel, die das Wissen kodiert, wie vorgegangen werden soll, um Ungleichungen abzuschätzen. Ist eine einfache (mit der Methode *Solve∗ <-m-b*) oder komplexere (mit *ComplexEstimate<-m-b*) direkte Abschätzung nicht möglich, so muß mit Hilfe der Methode *Unwraphyp-s-f* ein Teilterm aus einer Annahme extrahiert werden, der dann die Abschätzungen ermöglicht.

Methodenwissen (das Wissen um mathematische Vorgehensweisen) und

```
(control-rule
   (kind methods)
   (IF (goal-matches (?goal (?x < ?y))))
   (THEN  (prefer (Solve*<-m-b ?goal)
                  (ComplexEstimate<-m-b ?goal)
                  (Unwraphyp-s-f ?goal))))
```

Tabelle 2.6: Beispiel einer Kontrollregel

Kontrollwissen (das Wissen, wann welche Vorgehensweisen anzuwenden sind) lassen sich zu einer *Beweisstrategie* verbinden:

Strategien Eine *Beweisstrategie* [Mel98b] faßt eine Menge von Methoden und Kontrollregeln zusammen. Sie enthält das Wissen, das nötig ist, um in einem bestimmten Teilbereich der Mathematik erfolgreich Beweise zu führen. Strategische Kontrollregeln analysieren den Planungszustand und wählen dementsprechend eine Strategie aus. Ist zum Beispiel ein offenes Ziel eine Ungleichung, wird eine Strategie gewählt, die mit Hilfe von Abschätzungen Ungleichungen lösen kann.

Korrektheit eines Beweisplans Wurden alle offenen Ziele geschlossen, dann wurde ein Beweisplan gefunden, der das Beweisproblem löst. Ein Beweisplan ist allerdings nur ein *Plan*, er muß daher nicht zwingenderweise korrekt sein. Um die Korrektheit zu überprüfen, muß ein Beweisplan auf Kalkülebene expandiert werden. Dabei wird das instantiierte Beweisschema einer Methode in den Beweis eingefügt, und, wenn die Kalkülebene noch nicht erreicht wurde, rekursiv weiter expandiert. Auf Kalkülebene kann dann ein Verifizierer die Korrektheit der einzelnen Kalkülschritte überprüfen.

Beweisplandarstellung Ein Beweisplan läßt sich als Graph darstellen, dessen Knoten die Beweiszeilen repräsentieren[3] und dessen Kanten die Namen der auf die Zeilen angewandten Methoden tragen. Abbildung 2.2 und

---

[3] Daher werde ich Beweiszeilen im folgenden oft *Knoten* nennen.

2.3 zeigen einen Ausschnitt eines Beweisplans tabellarisch bzw. in Graphendarstellung.

In der graphischen Darstellung geben die Kanten die Richtung der Methode an. Nach unten gerichtete Kanten stehen für eine Rückwärtsmethode; nach oben gerichtete für eine Vorwärtsmethode. Treffen zwei Kanten aufeinander, wie in Tabelle 2.3 bei Methode *Solve*$*_{\leq}$, so ist diese Methode eine Rückwärtsmethode, die eine Annahme benötigt (in diesem Fall L_7).

In ΩMEGAs Plandatenstruktur PDS [BCF$^+$97] wird zusätzlich zu dem Beweis unter anderem die Reihenfolge der Methodenanwendungen gespeichert. Der Vorgänger bzw. Nachfolger eines Schrittes ist der Planungsschritt, der zeitlich unmittelbar vorher bzw. nachher angewendet wurde. In der Graphendarstellung geben die Zahlen in Klammern hinter dem Methodennamen die Reihenfolge der Anwendung wieder. So wurde die Methode *Complex-Estimate*$<$*-m-b* als siebte Methode angewendet.

Die *Kinder* eines Planungsschrittes sind die Knoten, die den Prämissen der Methode des Schrittes entsprechen. So sind die Knoten L_7, L_{15} und L_{16} Kinder von L_9.

Wissensstrukturierung Das Wissen über Mathematik besteht sowohl aus mathematischen Einheiten, wie Axiomen, Definitionen, Beispielen und Beweisen, als auch aus dem Problemlösewissen, also den Methoden und Kontrollregeln. Im ΩMEGA-System ist dieses Wissen ist hierarchisch strukturiert und zu *Theorien* zusammengefaßt. Theorien sind in einer mathematischen Wissensbasis namens MBASE [KF00] gespeichert.

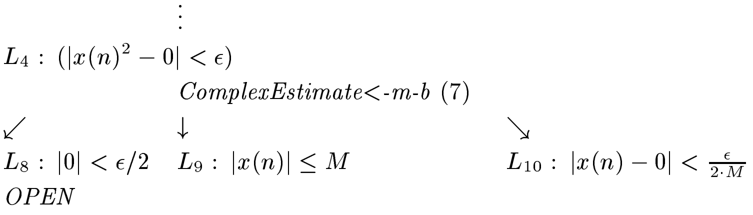
2.2.3 Beweisen von Grenzwertsätzen

Die meisten Beispiele, die ich in meiner Diplomarbeit verwende, sind aus der Domäne der Grenzwertsätze. In diesen Abschnitt werde ich eine kurze Einführung in diese Domäne geben. Für eine detailliertere Darstellung siehe [Mel98a].

Grenzwertsätze treffen Aussagen über den Grenzwert $L = \lim_{x \to a} f(x)$ einer

		$\vdots$	
L_4.	$\vartriangle$	$\vdash (\lvert x(n)^2 - 0\rvert < \epsilon)$	(*ComplexEstimate<-m-b* L8 L9 L10)
L_8.	$\vartriangle$	$\vdash \lvert 0\rvert < \epsilon/2$	(*OPEN*)
L_{10}.	$\vartriangle$	$\vdash \lvert x(n) - 0\rvert < \frac{\epsilon}{2\cdot M}$	(*OPEN*)
L_9.	$\vartriangle$	$\vdash \lvert x(n)\rvert \leq M$	(*Solve*$*_{\leq}$ L_7)
L_{15}.	$\vartriangle$	$\vdash n = n'$	(*Tell*)
L_{16}.	$\vartriangle$	$\vdash M \leq \epsilon'$	(*Tell*)
L_7.	$\vartriangle$	$\vdash (\lvert X(n')\rvert < \epsilon')$	(*Simplify* L_6)
L_6.	$\vartriangle$	$\vdash (\lvert X(n') - 0\rvert < \epsilon')$	(*Unwrap-Hyp* L_5)
		$\vdots$	

Abbildung 2.2: tabellarische Darstellung eines Beweisplans



Funktion f. Die formale Definition von $\lim_{x \to a} f(x)$ ist:

$$\forall \epsilon (0 < \epsilon \to \exists \delta (0 < \delta \wedge \forall x (x \neq a \wedge |x - a| < \delta \to |f(x) - L| < \epsilon))).$$

Eine Möglichkeit, Aussagen über Grenzwerte zu beweisen, ist die Existenz eines δ zu zeigen, so daß die Behauptung $|t_1| < \epsilon$ gilt, wenn die Annahme $|t_2| < \delta$ gilt. Diese Beweise heißen ϵ-δ-Beweise.

Das im ΩMEGA-System realisierte Vorgehen um ϵ-δ-Beweise zu führen, versucht komplizierte Abschätzungen mit Hilfe von Hilfsvariablen auf einfachere Abschätzungen zurückzuführen. Diese einfachen Abschätzungen werden von dem Constraintlöser $\mathcal{C}o\mathcal{SIE}$ [Zim00] verarbeitet: Beim Folgerungstest wird überprüft, ob eine Abschätzung aus den bereits gesammelten Abschätzungen folgt, beim Konsistenztest wird sie sowohl überprüft als auch zu den gesammelten Abschätzungen hinzugefügt .

Die verwendeten Abschätzungsmethoden und ihre Funktion sind

- *ComplexEstimate<-m-b*: Zerlegt eine komplizierte offene Ungleichung in vier einfachere Teilzeile (siehe auch Abbildung 2.1).
- *Solve* <-m-b*: Vereinfacht ein Ziel $t_1 < t_2$ mit der Hilfe einer Annahme $t_1' < t_3$ zu $\sigma(t_2) < \sigma(t_3)$, wenn t_1 und t_1' mit einer Substitution σ unifizierbar sind.
- *Factorial-Estimate-m-b*: Zerlegt die Abschätzung eines Bruches $\frac{t_1}{t_2} < \epsilon$ in die zwei Teilziele $t_1 < \epsilon \cdot M$ und $t_2 > M$.
- *TellCS-m-b/f*: Fügt die Formel eines Ziels (*TellCS-m-b*) oder einer Annahme (*TellCS-m-f*) dem Constraintlöser hinzu, wenn sie mit dem momentanen Constraintzustand konsistent ist.
- *AskCS-m-b*: Überprüft, ob ein Ziel aus dem momentanen Constraintzustand ableitbar bzw. in ihm enthalten ist.

Es gibt eine Reihe von Variationen der Abschätzungsmethoden, die abhängig von den in den Knoten auftretenden Relation sind (zum Beispiel *Solve* <-m-b* und *ComplexEstimate>-m-b*).

Andere verwendete Methoden sind

- *Simplify-m-f/b*: Vereinfacht einen Term mit Hilfe des Computeralgebrasystem MAPLE [CGG$^+$92].
- *Simplify-Inequality-m-f*: Vereinfacht eine Ungleichung.
- *Normalize-s-b*: Normalisiert ein offenes Ziel.
- *UnwrapHyp-s-f*: Extrahiert Teilformeln aus einer Annahme.
- *Def-I* und *Def-E*: Rückwärts- und Vorwärtsanwendung von Definitionsexpansion.
- *Skolemize-m-b*: Eliminierung von Quantoren.
- *ImpliesI-m-b*: Eliminiert Implikationen in offenen Knoten.
- *And-I/And-E*: Zerlegen Konjunktionen in offenen Knoten bzw. Annahmen in ihrer Konjunkte.

Nachdem ich nun den allgemeinen Rahmen vorgestellt habe, in dem sich die Analogie im Beweisplanen abspielt, werde ich im folgendem Abschnitt einen Überblick über die verschiedenen Analogieansätze geben.

2.3 Problemlösen durch Analogie

In der Kognitionswissenschaft gibt es eine Reihe von Modellen der Analogie. Bekannte implementierte Systeme sind zum Beispiel SME (The Structure-Mapping Engine) [FFG86] oder ACME (Analogical Mapping by Constraint Satisfaction) [HT89]. Wie die Namen aber bereits andeuten, liegt der Schwerpunkt der meisten Systeme auf der Abbildung (Mapping) der Objekte und Relationen zwischen den Problemen und nicht so sehr auf dem Problemlöseprozeß. Für das Beweisplanen sind diese Ansätze daher eher ungeeignet.

Eine Definition, die die Anforderungen an die Analogie im Beweisplanen gut charakterisiert, stammt von Carbonell [Car86]:

Definition 2.3: Analogical problem solving consists of transfering knowledge from past problem-solving episodes to new problems that share significant aspects with corresponding past experience and using the transfered knowledge to construct solutions to new problems.

Aus dem Bereich der Künstlichen Intelligenz gibt es verschiedene Ansätze für Analogie im automatischem Beweisen, die diese Sicht der Analogie umzusetzen versuchen:

Einer der ersten die Analogie im automatischen Beweisen anwandten, war Kling [Kli71]. Er setzte Analogie in einem Resolutionsbeweiser ein. Die Analogie diente allerdings nicht zu Konstruktion von Plänen, sondern zum Vorschlagen der Axiome, die ein Resolutionsbeweiser benutzte.

Munyer [Mun81] orientierte sich bei der Konstruktion der neuen Lösung bereits an dem Quellplan. Die Lösungsschritte des Quellplans werden in der Reihenfolge, in der sie angewendet wurden, auf das Zielproblem übertragen.

Mit Analogie im Beweisen von Grenzwertsätzen beschäftigte sich bereits Bledsoe und seine Arbeitsgruppe [BCP86, Ble95]. Die Abbildungen zwischen Quell- und Zielproblem mußten allerdings vom Benutzer vorgegeben werden. Der Beweis des Quellproblems (eine Sequenz von Kalkülschritten) wird auf das Zielproblem übertragen, indem die gleiche Kalkülregel auf die Terme des Ziels angewendet wird, die mit Hilfe der Abbildungen als analog zu den Quelltermen erkannt werden. Läßt sich eine Kalkülregel im Ziel nicht anwenden, so gibt es eine Reihe von möglichen Reparaturen, die auf Kalkülebene arbeiten.

Owen [Owe90] analysierte diese Ansätze und konnte zeigen, daß sie selbst für einfache Analogien nicht ausreichend waren. Er stellte eine Reihe von Erweiterungen vor, der Transfer fand aber weiterhin auf Kalkülebene statt, in diesem Fall Resolution.

Einen wichtigen Fortschritt brachten Systeme, die auf Matchen höherer Ordnung basieren [dlTC87, KW95, Cur95]. So löst Kolbe und Walters Plagiator Probleme durch Analogie im Gleichheitsbeweisen, die vorher nicht übertragbar waren. Allerdings sind sie nicht so sehr an dem eigentlichen Beweis interessiert, sondern extrahieren einen Beweiskern (*Proof-Shell*), der die für den Beweis benötigten Axiome enthält. Aufbauend auf Matchen höherer Stufe wird geprüft, ob der Beweiskern sich auf das Zielproblem übertragen läßt. In diesem Fall kann das Problem durch den neuen Beweiskern gelöst werden.

All diesen Systemen ist gemeinsam, daß sie auf einer sehr kalkülnahen Ebene arbeiten. Der Schwerpunkt liegt auf den Abbildung von Quell- auf Zielsymbolen, und größtenteils werden einzelne Kalkülschritte übertragen. Empirische Untersuchen [Mel93, Mel94] und Aussagen von Mathematikern [Had45, Pol73] legen aber nahe, daß dies nicht die Ebene ist, auf der Menschen Analogie verwenden. Nicht die einzelnen Rechen- bzw Kalkülschritte werden übertragen, sondern das allgemeine Vorgehen wird erneut angewendet. Aus der Sicht der Beweisplanung würde man sagen, daß der Transfer auf Planebene stattfindet.

Carbonell [Car86] betonte mit der *nachvollziehenden Analogie* (derivational analogy) eben diesen Planungsprozeß. Nicht nur die Wahl eines Operators, sondern auch die Gründe (*justifications*), warum der Operator ausgewählt wurde, sowie fehlgeschlagene Plannungsäste sollten beim Transfer berücksichtigt werden.

Ausgehend von Carbonells nachvollziehender Analogie realisierte Veloso [Vel92] eine umfassende Erweiterung des PRODIGY-Systems um eine Analogiekomponente. Allerdings ist das PRODIGY-System kein Beweisplaner, sondern erlaubt nur einfache Planungsdomänen (wie die Transport- oder die Roboterdomäne) und, im Vergleich zum Beweisplanen, nur sehr einfache Methoden.

Melis [Mel95] stellte ein neues Modell für Analogie im Beweisplanen vor. *Reformulierungen* sollen den Quellplan so ändern, daß die Quell- auf die Zielterme matchbar werden. Klassen von Reformulierungen sind u.a. Abstraktion, Normalisierung oder Änderung Zum Beispiel ist das Ersetzen eines Symbols durch ein anderes, die „Standard“-Reformulierung älterer Ansätze, eine Änderungs-Reformulierung. Der Transfer eines linearisierten Quellplans geschieht folgendermaßen: Für den zu übertragenden Quellknoten wird eine Reformulierung gesucht, die ihn auf einen Zielknoten abbildet. Wird eine solche Reformulierung gefunden, wird der Quellplan entsprechend reformuliert. Wird keine Reformulierung gefunden, wird versucht, den Quellplan zu dekomponieren, um Teil davon zu übertragen. Sind die Gründe (justifications) der dann zu übertragenden Methode erfüllt, so wird sie übertragen. Da Reformulierungen die Korrektheit der reformulierten Methode nicht garantieren, muß sie gegebenenfalls modifiziert werden. Der Zyklus beginnt von neuem, bis der Quellplan abgearbeitet ist oder im Zielbeweis alle offenen Ziele geschlossen wurden.

Eine Implementierung des Modells stellen Melis und Whittle mit ABALONE [MW99] vor, einer Analogieerweiterung des Beweisplaners für induktive Beweise CLAM[BvHS91]. Sowohl externe Analogie als auch interne Analogie sowie einige Reformulierungen wurden realisiert. Der Transfer geschieht wie in [Mel95] vorgestellt. Die Reformulierungen werden mit Hilfe eines Matches zweiter Stufe, das nach jedem Schritt erweitert wird, ausgewählt.

Schaarschmidts STRANGER [Sch96] ist eine Analogiekomponente für den Beweisplaner ΩMEGA, die ebenfalls auf [Mel95] basiert. Er erweiterte den Ansatz um eine Phase, in der die für den Beweis benötigten Lemmas gematched werden und implementierte die Reformulierung *add-argument*. Diese Reformulierung verändert die Konklusionen, Prämissen und das Beweisschema einer Methode, wenn beim Matchen einer Funktion die Anzahl der Argumente verdoppelt werden. Die Reformulierungen von Methoden geschehen allerdings auf einem sehr kalkülnahen Niveau.

Das anfängliche Ziel meiner Arbeit war die Erweiterung des STRANGER-Systems. Es hat sich aber sehr schnell herausgestellt, daß die Reformulierung von Methoden nicht ausreicht, um Analogie im Beweisplanen zu ermöglichen. Aber auch die anderen Ansätze ließen sich nicht auf die Situation des Beweisplanens in ΩMEGA übertragen. Wo liegen die Probleme?

Betrachten wir folgende Grenzwertaussagen:

$$\lim_{x \to 3} \frac{3}{1-x} = -\frac{3}{2}$$

und

$$\lim_{x \to 2} \frac{x^3 - 4}{x^2 + 1} = \frac{4}{5}.$$

Auch wenn diese Beweisprobleme sehr unterschiedlich aussehen und ein Match zwischen ihnen nur sehr schwer zu finden sein wird, können beide durch den gleichen Plan gelöst werden. Das heißt, die Vorgehensweise, zuerst zu Matchen und dann mit denen sich aus dem Match ergebenden Reformulierungen den Quellplan anzupassen, ist so im Beweisplanen nicht mehr anzuwenden. Der hohe Abstraktionsgrad von Plänen bringt es mit sich, daß syntaktisch sehr unterschiedliche Probleme trotz der Unterschiede durch den gleichen Plan gelöst werden können, ein krasser Gegensatz zu Owens Behauptung [Owe90]: *„In fact, we can think of the later stages [of analogy] as being the extension of the initial match between the problems to the solutions“*. Wenn keine Aussagen über die Beziehung zwischen Quell- und Zielproblem mit Hilfe des Matchen getroffen werden können, kann dann das mathematischen Wissens der Planebene genutzt werden?

Weiterhin stellt sich die Frage nach den für die Planebene geeigneten Reformulierungen. Können nicht die Vorteile der Planebene ebenfalls für die Reformulierungen ausgenutzt werden?

Wie diese und weitere Probleme durch ein Analogiesystem, das auf Planebene arbeitet, gelöst werden können, werde ich in den folgenden Kapiteln bei der Erörterung von TOPAL darstellen.

Kapitel 3

Der erweiterte Matcher

> Proper understanding is, finally, a grasping of relations. But we understand a relation more distinctly and more purely when we recognize it as the same in widely different cases and between completely heterogeneous objects.
>
> *Arthur Schopenhauer*

In diesem Kapitel werde ich zuerst auf die Anwendung des Matchens im Beweisplanen durch Analogie eingehen und dann dafür benötige Erweiterungen vorstellen.

3.1 Matchen in der Analogie

Das Matchen wird in der Analogie benutzt, um Terme aus dem Quellbeweis mit Termen aus dem Zielbeweis vergleichen zu können.

Dabei wird untersucht, welches die Gemeinsamkeiten und Unterschiede eines Quell- und eines Zielterms bezüglich der Termstruktur sind. Diese Informationen werden im Laufe des Matchprozesses berechnet. Um Terme, wie sie im Beweisen durch Analogie auftreten, miteinander zu vergleichen, sind allerdings Erweiterungen des Matchalgorithmus nötig. Diese stelle ich in Abschnitt 3.4 und 3.5 vor.

Weiterhin können mit Hilfe des Matchens mehrere Quellterme bezüglich desselben Zielterms miteinander verglichen werden. So kann bestimmt werden, welcher Quellterm dem Zielterm am ähnlichsten ist. Dies kann durch eine Kostenfunktion über den Operationen des Matches berechnet werden. Diese stelle ich in Abschnitt 3.3 vor.

Da für die Analogie nur die Ähnlichkeiten der Quellterme bezüglich der Zielterme analysiert werden sollen, reicht es aus, sich auf Matchen im Gegensatz zu Unifikation zu beschränken. Es können also sämtliche im Zielterm auftretenden Symbole als Konstanten behandelt werden. Um aber überhaupt die Terme des Quellbeweises auf die Terme des Zieles matchen zu können, müssen diese variabilisiert werden. Den Prozeß der Variabilisierung erläutere ich im folgenden Abschnitt.

3.2 Variabilisierung

Um Matchen zu ermöglichen, müssen bestimmte Symbole der Quellterme als Variablen deklariert werden.

Die Frage, *welche* Symbole dies sind, kann beispielsweise vom Benutzer beantwortet werden, der eine Liste der betreffenden Symbole zu übergeben hat (siehe [Sch96]). Allerdings wird in diesem Fall vom Benutzer verlangt, den gesamten Analogieprozeß bereits vor der Ausführung zu überschauen und sämtliche Fälle zu berücksichtigen. Da diese Voraussetzung in der Regel nicht gegeben ist, ist diese Lösung nicht sinnvoll.

In TOPAL-X kann der Benutzer zwar auch Symbole vorgeben, dies ist aber nicht notwendig. Erreicht wird dies durch eine vollständige Variabilisierung (abgesehen von den logischen Symbolen) der Quellterme. Im folgendem werde ich variabilisierte Symbole mit Index v kennzeichnen (also zum Beispiel $+_v$).

Der Preis der Variabilisierung ist eine Vergrößerung des Suchraums durch das vermehrte Auftreten von Variablen. Dieser erweist sich aber dank der in den folgenden Abschnitten erläuterten Heuristiken als beherrschbar.

Die Variabilisierung selbst geschieht „lazy“, d.h. erst wenn im Laufe des Transfers auf einen Quellknoten zugegriffen werden muß. Da normalerweise nur eine geringe Zahl von Knoten aufeinander gematched werden müssen, ist es unnötig, den gesamten Quellbeweisplan zu variabilisieren.

Zudem wird über die Variabilisierung Buch geführt: Jedem Knoten wird seine variablisierte Formel zugeordnet (um den Aufwand für mehrfache Variabilisierung zu verhindern) und jedem Symbol seine Variable. Das heißt auch, daß im Gegensatz zur *Indizierung* (siehe zum Beispiel [KW95]), in der mehrfach auftretenden Symbolen jeweils verschiedene Variablen zugeordnet werden, jedes Vorkommen eines Symbols durch die selbe Variable ersetzt wird. Dadurch können Konflikte, also das Ersetzen eines Quellsymbols durch verschiedene Zielterme, erkannt werden. Die Lösungsmenge des erweiterten Matchers wird durch die nichtindizierte Variabilisierung nicht eingeschränkt, da durch Termabbildungen dieselbe Variable auf verschiedene Terme abgebildet werden kann (siehe Abschnitt 3.4).

3.3 Kostenfunktion und Heuristiken

Wie bereits erwähnt, kann der Suchraum durch das Auftreten von Funktionsvariablen extrem anwachsen. Im Beweisen durch Analogie ist es glücklicherweise nicht nötig, alle möglichen Lösungen für ein Matchproblem zu berechnen. Gesucht ist die Lösung, die am meisten über die strukturelle Ähnlichkeit zweier Terme aussagt.

Projektion und Imitation sind verschieden gut geeignet, die Struktur eines Terms zu erhalten. So wird für Beispiel 2.2

$$\{F_{\alpha\to\beta}(x_\alpha) \overset{?}{\mapsto} g_{\alpha\to\beta}(c_\alpha)\}$$

in der ersten Lösung

$$\{F_{\alpha\to\beta} \mapsto \lambda z_\alpha . g_{\alpha\to\beta}(z_\alpha), x_\alpha \mapsto c_\alpha\}$$

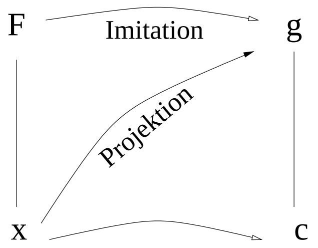


Abbildung 3.1: Beispiel für Imitation/Projektion

die Funktionsvariable F durch eine Funktionskonstante substituiert, während durch die Projektionslösung

$$\{F_{\alpha\to\beta} \mapsto \lambda z_{\alpha}.g_{\alpha\to\beta}(c_{\alpha})\}.$$

das Argument x durch eine Funktionskonstante substituiert wird. Der erste Fall erhält mehr von der Struktur des Terms als der zweite. Betrachtet man die Terme als Bäume (siehe Abbildung 3.1), so sollen Blätter möglichst auf Blätter und Knoten auf Knoten abgebildet werden. Dies kann durch eine geeignete Bewertung von Imitation und Projektion erreicht werden.

Bewertungen können in den Matchalgorithmus eingebaut werden, indem Matchprobleme um eine Kostenkomponente erweitert werden. Ein Matchproblem $\mathcal{T}$ besteht dann nicht mehr nur aus einer Termpaarmenge, sondern zusätzlich aus einer natürlichen Zahl, die die Kosten repräsentiert. Der Matchalgorithmus wird dahingehend geändert, daß erstens bei jeder Anwendung einer Transformationsregel die Kostenfunktion *cost* die Kosten der neu kreierten Matchprobleme berechnet. Zweitens werden alle anwendbaren Transformationsregeln angewendet, dann aber an dem Matchproblem mit den geringsten Kosten weitergearbeitet wird. Dieses Vorgehen entspricht der *uniform cost search* [Dij59]. Da die Voraussetzung erfüllt ist, daß die Kosten der Matchprobleme nie abnehmen, ist nach Dijkstra garantiert, daß die günstigsten Lösungen eines Matchproblems berechnet werden.

Die verwendeten Heuristiken sind:

1. Bevorzuge Imitation von nicht neu eingeführten Funktionsvariablen

gegenüber Projektion. Dadurch werden bevorzugt Funktionssymbole aus dem Quellproblem auf Funktionssymbole aus dem Zielproblem abgebildet.
Beispiel: Für $f_v(x_v)\overset{?}{\mapsto}g(y)$ ist $cost(f_v\overset{?}{\mapsto}\lambda z.g(H_{v_1}(z))) < cost(f_v\overset{?}{\mapsto}\lambda z.z)$.

2. Bevorzuge Projektion gegenüber Imitation von neu eingeführten Funktionsvariablen, den Hilfsvariablen. Dadurch werden bevorzugt die Argumente der Funktionen aufeinander abgebildet.
   Beispiel: Für $H_{v_1}(x_v)\overset{?}{\mapsto}y$ ist $cost(H_{v_1}\overset{?}{\mapsto}\lambda z.z) < cost(H_{v_1}\overset{?}{\mapsto}\lambda z.y)$.

3. Dekomposition: Durch die Buchführung innerhalb der Variabilisierung ist es möglich festzustellen, aus welcher Konstante eine Funktionsvariable entstanden ist. Ist diese gleich der Konstanten, auf die die Variable abgebildet werden soll, so wird Dekomposition bevorzugt. Es wird also versucht, die Argumente direkt, ohne Einführung von Hilfsvariablen, aufeinander zu matchen. Die Kosten der Dekomposition sind 0.
   Beispiel: $\{+_v(x_v, y_v)\overset{?}{\mapsto} + (a, b)\}$ wird in $\{+_v\overset{?}{\mapsto}+, x_v\overset{?}{\mapsto}a, y_v\overset{?}{\mapsto}b\}$ transformiert. Keine Hilfsvariablen werden eingeführt.

Diese drei Fälle können während des Matchprozesses bei der Anwendung der Transformationsregeln erkannt werden.

Andere Bewertungsstrategien wurden unter anderem von Kolbe und Walther [KW95] vorgeschlagen. Sie unterscheiden weitere strukturelle Unterschiede, wie Vertauschen, Duplizieren und Löschen von Argumenten. Diese können allerdings erst identifiziert werden, wenn für alle im Matchproblem vorkommenden Variablen eine Lösung gefunden wurde:

Beispiel 3.1:
Für $f_v(x_v, y_v)\overset{?}{\mapsto}g(a, b)$ ist $\{f_v \mapsto \lambda z_1, z_2.g(H_{v_1}(z_1, z_2), H_{v_2}(z_1, z_2))\}$ die Imitationslösung. Je nach der Substitution der H_{v_j} kann die Reihenfolge der Argumente gleich bleiben ($\{H_{v_1} \mapsto \lambda z_1, z_2.z_1, H_{v_2} \mapsto \lambda z_1, z_2.z_2\}$) oder vertauscht werden ($\{H_{v_1} \mapsto \lambda z_1, z_2.z_2, H_{v_2} \mapsto \lambda z_1, z_2.z_1\}$).

Wir konnten feststellen, daß zur Einschränkung des Suchraums die Heuristiken 1 – 3 ausreichend sind. Sie haben den Vorteil, ohne den zusätzlichen

Verwaltungsaufwand, wie er für die Online-Erkennung der weiteren strukturellen Unterschiede nötig wäre, realisierbar zu sein. Daher haben wir uns auf diese Heuristiken beschränkt.

Leider reicht im Beweisen durch Analogie ein Matchalgorithmus höherer Ordnung wie er vorgestellt wurde in der Regel nicht aus, um alle Terme aufeinander zu matchen. Der Grund ist, daß einem Matchalgorithmus als einzige Operation um Terme zu matchen, die Substitution zur Verfügung steht. Schon für leicht komplexere Terme (zum Beispiel $t_1 \wedge t_2 \overset{?}{\mapsto} t_1$) kann dann bereits keine Lösung mehr gefunden werden. Möchte man solche Term aufeinander „matchen“, so ist es nötig, den Matchalgorithmus zu erweitern. Dem hier vorgestellten erweitertem Matchalgorithmus stehen die zusätzlichen Operationen *Termabbildung* (siehe folgenden Abschnitt) und *Reparaturen* (Abschnitt 3.5) zur Verfügung: Termabbildung bildet Terme auf andere Terme ab, Reparaturen ermöglichen das Hinzufügen und Entfernen von logischen Symbolen inklusive Teilformeln.

3.4 Termabbildungen

Matchen versagt, wenn eine Variable auf verschiedene Terme abgebildet werden muß:

Beispiel 3.2: Für $+_v(x_v, x_v) \overset{?}{\mapsto} +(a, b)$ ist ein möglicher Lösungsschritt die Anwendung der Dekomposition. Das dadurch entstehende Matchproblem $\{x_v \overset{?}{\mapsto} a, x_v \overset{?}{\mapsto} b\}$ kann jedoch nicht gelöst werden, da x_v nicht gleichzeitig auf zwei verschiedene Konstanten abgebildet werden kann.

Strukturell gesehen ist diese Lösung jedoch durchaus wünschenswert, da eine Funktion auf eine Funktion und die Argumente auf die Argumente abgebildet werden. Kolbe und Walther lösen dieses Problem durch Indizierung verschiedene Vorkommen einer Konstante und erlauben so deren Variabilisierung durch verschiedene Variablen. Das von uns gewählte Vorgehen ist die Termabbildung, die einen Term t_1 durch einen anderen Term t_2 ersetzt. Bei t_1 muß es sich jedoch nicht um eine Variable handeln, Konstanten und komplexe Terme sind ebenfalls erlaubt. Zudem kann ein Term mehrere Term-

(5) Termabbildung

$$\begin{array}{rl} & (\{t_v \overset{?}{\mapsto} \lambda\overline{x_k}.t\} \cup S, \mathit{term\_maps}, \mathit{costs}) \\ \Rightarrow & (S, \{t_v \mapsto_T \lambda\overline{x_k}.t\} \cup \mathit{term\_maps}, \mathit{costs} + \mathit{cost}(t_v, \lambda\overline{x_k}.t)) \end{array}$$

Tabelle 3.1: Termabbildungsregel

abbildungen haben.

Um Termabbildungen in dem Matchalgorithmus zu ermöglichen, werden Matchprobleme um eine zusätzliche Komponente erweitert, die die Termabbildungen speichert. Dem Matchalgorithmus wird zudem eine Termabbildungsregel hinzugefügt (siehe Tabelle 3.1). $t_1 \mapsto_T t_2$ steht im folgendem für die Termabbildung des Terms t_1 auf den Term t_2.

Beispiel 3.3: Für Beispiel 3.2 ist $(\{+_v \mapsto +, x_v \mapsto a, \}, (\{x_v \mapsto_T b\}))$ eine Lösung, die sowohl Substitution als auch Termabbildungen beinhaltet. Eine weitere Lösung ist $(\emptyset, (\{+_v(x_v, x_v) \mapsto_T +(a, b)\}))$, die die kompletten Terme aufeinander abbildet.

Da Termabbildung eine sehr mächtige Operation ist (jeder Term kann direkt auf einen anderen Term abgebildet werden), haben wir sie soweit eingeschränkt, daß die Terme keine logischen Symbole enthalten dürfen. Zudem werden Lösungen mit Termabbildungen immer teurer bewertet als Lösungen die nur Substitutionen enthalten.

Dafür wird die Kostenfunktion *cost* folgendermaßen auf Termabbildungen erweitert: Sie liefert nun einen Vektor $(\mathit{subst\_costs}, \mathit{map\_costs})$, wobei $\mathit{subst\_costs}$ die oben definierten Kosten der Substitutionen sind. $\mathit{map\_costs}$ sind die Kosten der Termabbildungen. Eine Ordnung über die Vektoren ist folgendermaßen definiert: (sc_1, mc_1) sei genau dann kleiner als (sc_2, mc_2), wenn $sc_1 < sc_2$, oder $sc_1 = sc_2$ und $mc_1 < mc_2$.

Die Kosten einer Termabbildung sind über die Termtiefe[1] (*depth*) defi-

---

[1] Die Tiefe eines Terms ist analog zur Tiefe eines Baumes die maximale Anzahl der Knoten von der Wurzel bis zum einem Blatt.

niert:

$$cost(t_1 \mapsto_T t_2) := depth(t_1)^2 + depth(t_2)^2.$$

Die Kosten für Termabbildungen steigen quadratisch zur Tiefe des Terms, da es billiger sein soll, mehrere kleine Terme aufeinander abzubilden, als einen großen. Dies entspricht dem Ziel der Substitutionsheuristiken, bevorzugt Blätter auf Blätter und Knoten auf Knoten abzubilden:

Beispiel 3.4:
Für die erste Lösung aus Beispiel 3.3 sind die Kosten $cost(\{x_v \mapsto_T b\}) = (0, 1^2 + 1^2) = (0, 2)$, für die zweite, komplexere Lösung sind die Kosten $cost(\{+_v(x_v, x_v) \mapsto_T +(a, b)\}) = (0, 2^2 + 2^2) = (0, 8)$. Eine dritte Lösung, die wegen der quadratischen Steigerung der Kosten billiger ist als die zweite, ist $(\{+_v \mapsto +\}, \{x_v \mapsto_T a, x_v \mapsto b\})$ mit Kosten $(0, 2 * (1^2 + 1^2)) = (0, 4)$.

3.5 Reparaturen

Trotz Substitution und Termabbildungen findet der erweiterte Matcher keine Lösung für die Klasse von Problemen, deren Terme sich in Hinsicht auf logische Symbole unterscheiden, wie fehlende oder zusätzliche Symbole (zum Beispiel $t_1 \wedge t_2 \overset{?}{\mapsto} t_1$), oder vertauschte Argumente (zum Beispiel $t_1 \wedge t_2 \overset{?}{\mapsto} t_2 \wedge t_1$).

Um auch diese Fälle lösen zu können, werden zu den bisherigen Transformationsregeln die in Tabelle 3.2 und 3.3 dargestellten Regeln hinzugefügt und Matchprobleme um eine Komponente *repairs* erweitert. In dieser Komponente werden die durchgeführten Reparaturen (wie Hinzufügen und Entfernen von Teilformeln) gespeichert. Die Reparaturen erweitern die Fähigkeiten des erweiterten Matchers in unterschiedlicher Weise:

Die *Argumentvertauschung (swap)*, Regel 6a, kreiert ein Matchproblem wie die Dekomposition, allerdings werden die Argumente der logischen Symbole vertauscht.

Das *Hinzufügen (add)*, Regeln 6b1, 6b2 und 6d1, erlaubt es, strukturell kleinere Formeln auf größere zu matchen, indem für binäre logische Symbole zwei Matchprobleme erzeugt werden, in denen die Quellformel jeweils auf das erste bzw. zweite Argument der Zielformel gematched wird. Im Falle von Quantoren wird der Rumpf der Quellformel auf die Zielformel gematched.

Das *Entfernen (del)*, Regeln 6c1, 6c2 und 6d2, ermöglicht die umgekehrte Operation, also größere Terme auf kleinere abzubilden.

Beispiel 3.5: Für $(t_1 \wedge t_2 \overset{?}{\mapsto} t_3 \wedge t_4)$ werden durch die Reparaturen folgende Matchprobleme erzeugt:

Argumentvertauschung: $\{t_1 \overset{?}{\mapsto} t_4, t_2 \overset{?}{\mapsto} t_3\}$.

Hinzufügung: $\{t_1 \wedge t_2 \overset{?}{\mapsto} t_3\}$ und $\{t_1 \wedge t_2 \overset{?}{\mapsto} t_4\}$.

Beseitigung: $\{t_1 \overset{?}{\mapsto} t_3 \wedge t_4\}$ und $\{t_2 \overset{?}{\mapsto} t_3 \wedge t_4\}$.

Auch der Kostenvektor wird um eine Komponente erweitert, der die Kosten der Reparaturen speichert. Die Kosten der einzelnen Reparaturen sind für jedes logische Symbol individuell gewichtbar. Die Voreinstellung ist 0 für die Vertauschung der kommutativen logischen Symbole, ansonsten 1.

Reparaturen wurden bereits von Schaarschmidt [Sch96] vorgestellt, allerdings ist seine Heuristik zum Bestimmen der Operationen nur beschränkt nutzbar. Da nur sehr einfache Hinzufügungen gefunden werden können, sind Probleme wie $(t_1 \wedge t_2 \overset{?}{\mapsto} t_2)$ bereits nicht mehr lösbar.

3.6 Zusammenfassung

Um Knoten aufeinander zu matchen, deren Unterschiede nicht durch Matchen höherer Ordnung erkannt werden können, wurde ein Matchalgorithmus höherer Ordnung erweitert. Diese Erweiterungen erlauben es, Formeln mit komplexen Unterschieden zu vergleichen.

Abschließend muß ich darauf hinweisen, daß trotz der Heuristiken das erweiterte Matchen der zeitintensivste Prozeß in unserem Analogiealgorithmus ist. Dieser und weitere Gründe auf die ich im Laufe dieser Arbeit zu sprechen kommen werde, führten dazu, das Matchen möglichst zu vermeiden und auf weniger syntaktische Unterscheidungsmöglichkeiten zurückzugreifen.

(6a) Argumentvertauschung

$$\begin{aligned} & (\{\lambda\overline{x_k}.op(t_1,t_2)\overset{?}{\mapsto}\lambda\overline{x_k}.op(t_1',t_2')\}\cup S, term\_maps, repairs, costs) \\ \Rightarrow\ & (\{\lambda\overline{x_k}.t_1\overset{?}{\mapsto}\lambda\overline{x_k}.t_2'\}, \{\lambda\overline{x_k}.t_2\overset{?}{\mapsto}\lambda\overline{x_k}.t_1'\}\cup S, term\_maps, \\ & repairs\cup swap(op,t_1,t_2), costs + swap\_cost(op)) \end{aligned}$$

(6b1) Hinzufügung rechts

$$\begin{aligned} & (\{\lambda\overline{x_k}.t\overset{?}{\mapsto}\lambda\overline{x_k}.op(t_1,t_2)\}\cup S, term\_maps, repairs, costs) \\ \Rightarrow\ & (\{\lambda\overline{x_k}.t\overset{?}{\mapsto}\lambda\overline{x_k}.t_1\}\cup S, term\_maps, \\ & repairs\cup add(op,t_2), costs + add\_cost(op)) \end{aligned}$$

(6b2) Hinzufügung links

$$\begin{aligned} & (\{\lambda\overline{x_k}.t\overset{?}{\mapsto}\lambda\overline{x_k}.op(t_1,t_2)\}\cup S, term\_maps, repairs, costs) \\ \Rightarrow\ & (\{\lambda\overline{x_k}.t\overset{?}{\mapsto}\lambda\overline{x_k}.t_2\}\cup S, term\_maps, \\ & repairs\cup add(op,t_1), costs + add\_cost(op)) \end{aligned}$$

(6c1) Beseitigung links

$$\begin{aligned} & (\{\lambda\overline{x_k}.op(t_1,t_2)\overset{?}{\mapsto}\lambda\overline{x_k}.t\}\cup S, term\_maps, repairs, costs) \\ \Rightarrow\ & (\{\lambda\overline{x_k}.t_2\overset{?}{\mapsto}\lambda\overline{x_k}.t\}\cup S, term\_maps, \\ & repairs\cup del(op,t_1), costs + del\_cost(op)) \end{aligned}$$

(6c2) Beseitigung rechts

$$\begin{aligned} & (\{\lambda\overline{x_k}.op(t_1,t_2)\overset{?}{\mapsto}\lambda\overline{x_k}.t\}\cup S, term\_maps, repairs, costs) \\ \Rightarrow\ & (\{\lambda\overline{x_k}.t_1\overset{?}{\mapsto}\lambda\overline{x_k}.t\}\cup S, term\_maps, \\ & repairs\cup del(op,t_2), costs + del\_cost(op)) \end{aligned}$$

op steht für ein beliebiges binäres logisches Symbol.

Tabelle 3.2: Reparaturen

(6d1) Hinzufügung Quantor

$$\begin{array}{rl} & (\{t_1 \overset{?}{\mapsto} op_Q.x_k t_2\} \cup S, term\_maps, repairs, costs) \\ \Rightarrow & (\{t_1 \overset{?}{\mapsto} t_2\} \cup S, term\_maps, \\ & \ repairs \cup add(op_Q), costs + add\_cost(op_Q)) \end{array}$$

(6d2) Beseitigung Quantor

$$\begin{array}{rl} & (\{op_Q.x_k t_1 \overset{?}{\mapsto} t_2\} \cup S, term\_maps, repairs, costs) \\ \Rightarrow & (\{t_1 \overset{?}{\mapsto} t_2\} \cup S, term\_maps, \\ & \ repairs \cup del(op_Q), costs + del\_cost(op_Q)) \end{array}$$

op_Q steht für einen beliebigen Quantor.

Tabelle 3.3: Reparaturen für Quantoren

Kapitel 4

Externe Analogie

> Even in the mathematical sciences, our principal instruments to discover the truth are induction and analogy.
>
> *Pierre Simon Marquis de Laplace*
> *Essai philosophique sur les probabilitès*

In diesem Kapitel stelle ich das Beweisen durch externe Analogie auf Planebene (TOPAL-X) vor, das ein Problem mit Hilfe eines bereits geführten Beweises löst. Tabelle 4.1 zeigt eine erste Übersicht über den Algorithmus der externen Analogie. Sie soll es vor allem möglich machen, die Rollen der einzelnen Phasen der Analogie in den Gesamtalgorithmus einzuordnen.

Nach dem Bestimmen und Laden des Quellproblems (Abschnitt 4.1, Retrieval) wird zunächst der Teil des Quellplanes bestimmt, der übertragen werden soll (Abschnitt 4.2, Theoremassoziation). Dann beginnt der Transfer (Abschnitt 4.3, Transfer), und der Quellplan wird Schritt für Schritt übertragen. Falls sich ein Schritt als nicht übertragbar herausstellt, werden verschiedene Aktionen ausgeführt, die den weitergehenden Transfer ermöglichen sollen (Abschnitt 4.4, Adaptation). Wurden alle Schritte des Quellplans übertragen, wird die Analogie beendet. Falls der Quellplan nicht ausgereicht hat, kann der Zielplan dann noch offene Knoten enthalten.

```
Eingabe: Zielproblem.
Ausgabe: (partieller) Zielplan.

Retrieval: Bestimme und lade Quellplan für Zielproblem.
Theoremassoziation:
    bis ein geeigneter erweiterter_match $\sigma$ zwischen Quell(unter)ziel $g_s$
        und Zieltheorem gefunden wurde:
      Füge Planungsschritte im Zielplan ein.
Transfer:
    solange der Quellplan noch Schritte enthält:
        $P_S$ := nächster Planungsschritt aus Quellplan.
        Bestimme ähnlichsten Zielplanungsschritt $P_T$ zu $P_S$ mit
          Hilfe von $\sigma$.
        wenn $P_T = \emptyset$ dann reformuliere,
          sonst wende $P_T$ im Ziel an.
Ausgabe: (partieller) Zielplan.
```

Tabelle 4.1: Übersicht der analogie-gesteuerten Beweisplankonstruktion

In den folgenden Abschnitten wird die Übersicht detailliert erläutert. Neben kleineren Beispielen werde ich das folgende komplexe Problemen benutzen, um die verschiedenen Phasen der Analogie zu verdeutlichen.

Beispiel 4.1: Tabelle 4.2 zeigt den Quellplan[1] in dem gezeigt wird, daß die Folge $x(n)^2$ gegen 0 konvergiert, wenn die Folge $x(n)$ ebenfalls gegen 0 konvergiert. Formal

$$Thm_Q : \lim_{n \to \infty} x(n) = 0 \to \lim_{n \to \infty} x(n)^2 = 0.$$

Der Beweisplan liest sich wie in der Einführung beschrieben, d.h. der Name der Methode erscheint fettgedruckt, und die Richtung des Schrittes wird durch den Pfeil angezeigt (so ist zum Beispiel L_1 durch Anwendung von Methode *Implies-I* auf den Knoten Thm im ersten Schritt erzeugt worden). Das Zielproblem auf das dieser Plan übertragen werden soll, besagt, daß die

---

[1] Der tatsächliche Beweisplan ist etwas komplizierter, ich habe einige Schritte, die für die Verdeutlichung der Analogie unwichtig sind, weggelassen.

Folge $|x(n)|$ gegen 0 konvergiert, formal

$$Thm_Z : \lim_{n\to\infty} |x(n)| = 0.$$

Diese Formalisierung ist allerdings unvollständig, da die Voraussetzung fehlt, daß die Folge $x(n)$ gegen 0 konvergiert.

Der Anhang enthält die vollständige und von mir kommentierte Ausgabe von TOPAL während des Transfers dieses Beispiels.

4.1 Retrieval des Quellbeweisplanes

Das Ziel des Retrieval ist es zu bestimmen, welcher der bereits gelösten Probleme als Quellproblem dienen soll. Bei einer großen Menge an Problemen, ist es sehr aufwendig, das am besten geeignete Quellproblem zu finden, da das Zielproblem mit jedem potentiellen Quellproblem verglichen werden muß. Ist die Wissensbasis aber geeignet strukturiert, kann die Menge der in Frage kommenden Quellprobleme bereits in einer Vorauswahl erheblich eingeschränkt werden: Alle „zu weit entfernten“ Probleme werden direkt aussortiert [Kol83]. Dem ΩMEGA-System steht die mathematische Wissensbasis MBASE zur Verfügung in der mathematisches Wissen unter anderem nach Theorien gegliedert ist. Wie in der Einführung erwähnt, sind in Theorien sowohl die mathematischen Konzepte eines bestimmten Gebietes als auch das notwendige Vorgehenswissen, das nötig ist, um Beweise in diesem Gebiet zu führen, zusammengefaßt. In einer ersten Vorauswahl werden daher beim Einladen eines neuen Problems die bereits gelösten Probleme innerhalb dessen Theorie angezeigt. Diese sind am ehesten dazu geeignet, die Konstruktion des Zielbeweises zu leiten. Aus dieser Menge kann der Benutzer das konkrete Problem bestimmen, das als Quelle dienen soll. Der entsprechende Beweisplan wird geladen und steht dann als Quellplan zur Verfügung.

Die vollständige Automatisierung des Retrievals war nicht Gegenstand meiner Diplomarbeit. Wir konnten aber feststellen, daß Ansätze, die die Quellprobleme über das Matchen auswählen, nicht ausreichend sind. Pro-

$Thm_Q : \lim_{n \to \infty} x(n) = 0 \to \lim_{n \to \infty} x(n)^2 = 0$

Implies-I (1)

$\downarrow$

$L_1 : \lim_{n \to \infty} x(n)^2 = 0$

Def-I (2)

$\downarrow$

$L_3 : \forall \epsilon \exists k \forall n (0 < \epsilon \to (n > k \to (|x(n)^2 - 0| < \epsilon)))$

Skolemize (3)

$\downarrow$

$L_4 : (0 < \epsilon \to (n > k \to (|x(n)^2 - 0| < \epsilon)))$

Normalize (4)

$\downarrow$

$L_5 : (|x(n)^2 - 0| < \epsilon)$

Complex-Estimate$<$ (9)

$\swarrow$ $\downarrow$ $\searrow$

$L_{10} : |0| < \epsilon/2$ $L_{11} : |x(n)| \leq M$ $L_{12} : |x(n) - 0| < \frac{\epsilon}{2 \cdot M}$

Simplify (14)

bleme mit geringen syntaktischen Unterschieden zwischen den Theoremen können sehr verschiedene Pläne haben. Zum Beispiel ist für $\lim\limits_{x \to 1} 2 \cdot x = 2$ und $\lim\limits_{x \to 1} 2^x = 2$ der Matcher mit der Substitution $times_V \mapsto power$ ausreichend[2]. Die Pläne unterscheiden sich jedoch sehr, da unterschiedliche Methoden angewendet werden. Dieses Verhalten steht im Gegensatz zu einer Grundannahme des typischen fallbasierten Schließens, nämlich daß syntaktisch ähnliche Probleme ähnliche Lösungen besitzen. Es ist ein weiterer Hinweis darauf, daß in der Analogie im Beweisplanen das Matchen nicht mehr die zentrale Rolle spielt.

4.2 Theoremassoziation

Das Ziel dieser Phase ist zum einen die Position der Planungsschritte im Quell- und Zielplan zu bestimmen, ab denen der Transfer beginnen kann, zum anderen Informationen darüber zu gewinnen, wie groß und welcher Art die strukturellen Ähnlichkeiten und Unterschiede zwischen Quell- und Zieltheorem sind.

Das Bestimmen der Positionen der Planungsschritte ist nötig, da es nicht immer möglich ist, direkt den gesamten Quellplan zu übertragen. Die möglichen dabei auftretenden Probleme lassen sich in zwei Kategorien unterteilen:

- **Transfer von Teilplänen**
  Oft kann nicht der gesamte Quellplan, sondern nur ein Teilplan auf das Zielproblem übertragen werden (siehe Abbildung 4.1). Wenn zum Beispiel durch das Quellproblem $t_1 \wedge t_2$ bewiesen werden kann, das Zieltheorem t_2 aber zu t_2' korrespondiert, sollte nur der Teil des Quellplanes übertragen werden, der t_2 beweist.

- **Bestimmung der Anfangsschritte**
  Von Carbonell stammt das Konzept der Bestimmung der Anfangsschritte (*Initial Segment Concatenation* [Car83]). Dieses besagt, daß

---

[2] In $\mathcal{POST}$-Notation wird es deutlicher: $lim(times(2, x), 1, 2)$ und $lim(power(2, x), 1, 2)$.

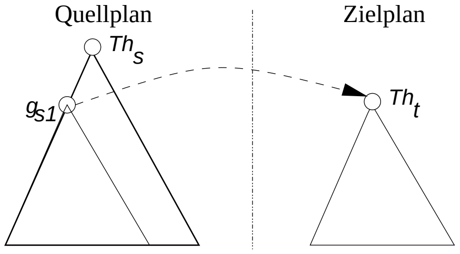


Abbildung 4.1: Übertragung eines Teilplans

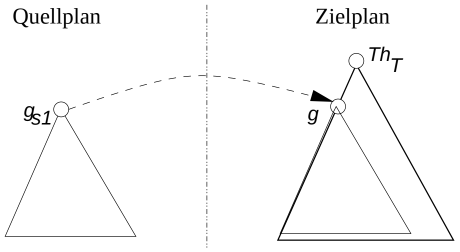


Abbildung 4.2: Bestimmung der Anfangsschritte

Methoden im Zielplan angewendet werden müssen, wenn der Quellplan nicht direkt übertragen werden kann (siehe Abbildung 4.2). So könnte zum Beispiel der Transfer erst nach einer Definitionsanwendung auf das Zieltheorem möglich werden.

Diese beiden Punkte spiegeln sich in unserem Theoremassoziationsalgorithmus (siehe Tabelle 4.3) wider: Zunächst wird versucht, einen geeigneten erweiterten Match zwischen dem Zieltheorem und dem Quelltheorem mit seinen Unterzielen zu finden. Geeignet heißt zum einen, daß die Unterschiede zwischen den ausgewählten Quell- und Zieltheoremen nicht zu groß sind (ausgedrückt durch die Kostengrenzen *costs_limit*). Zum anderen muß die Methode, die auf dem Knoten im Quellplan angewendet wurde, im Zielplan anwendbar sein. Der Transfer soll nicht an einem Schritt beginnen, der nicht übertragbar ist. Erfüllt das Quelltheorem diese Bedingungen nicht, so werden dessen Unterziele überprüft, indem die Tiefe der betrachteten Quell-

```
Eingabe: Quellplan, Zielplan.
Ausgabe: $Matcher(T_S \overset{?}{\mapsto} T_T)$, Quelltheoremknoten $T_S$, Zieltheoremknoten $T_T$.

Initialisiere Tiefe: $d := 1$.
Wiederhole
    $\mathcal{N}_Z :=$ offene Zielknoten.
    $\mathcal{N}_Q :=$ Quellknoten mit Tiefe $d$.
    $d := d + 1$.
    $valid\_steps := \{\ Matcher(N_{S_i} \overset{?}{\mapsto} N_{T_j}) | N_{S_i} \in \mathcal{N}_Q, N_{T_j} \in \mathcal{N}_Z,$
                     Methode von $N_{S_i}$ erfüllt Anwendungsbedingungen
                     im Zielplan,
                     $cost(Matcher(N_{S_i} \overset{?}{\mapsto} N_{T_j})) < costs\_limit\}$
bis $valid\_steps \neq \emptyset$ oder $d \geq depth\_limit$.
Wenn $d = depth\_limit$
  dann füge Planungsschritt im Zielplan ein, wiederhole Theoremassoziation
  sonst Ausgabe: billigster $Matcher(N_{S_i} \overset{?}{\mapsto} N_{T_j}) \in valid\_steps, N_{S_i}, N_{T_j}$.
```

Tabelle 4.3: Theoremassoziation

knoten erhöht wird. Dieser Zyklus wird, wenn nötig, solange wiederholt, bis die Kostengrenze *depth_limit* überschritten wird. Wird kein geeigneter Match gefunden, so wird im Zielplan ein Rückwärtsschritt geplant. Dann wird die Theoremassoziation wiederholt, um die Quelltheoreme mit den neu erzeugten offenen Knoten im Zielplan zu vergleichen.

Durch das Einbeziehen der Unterziele des Quellplans kann der Transfer von Teilplänen realisiert werden. Die Bestimmung der Anfangsschritte wird durch das Einfügen von Planungsschritten im Ziel möglich gemacht.

Die Entscheidung, die Tiefe der betrachteten Unterziele schrittweise zu erhöhen, anstatt direkt das Zieltheorem mit allen Quelltheoremen bis zum Tiefenlimit zu matchen, erfolgte aus dem Grund, daß der Transfer so früh wie möglich beginnen sollte. Ansonsten könnten wichtige Schritte übersprungen werden.

Die zweite Funktion der Theoremassoziation ist es, Informationen über die strukturellen Ähnlichkeiten von Ziel- und Quelltheorem zu finden. Diese können später genutzt werden, um Lemmas vorzuschlagen, die im Zielplan nicht vorhanden sind, aber für den Transfer benötigt werden. Diese Informationen werden durch den erweiterten Matcher gespeichert.

Beispiel 4.2: Weiterführung von $X(n)^2 \mapsto |X(n)|$
In der Theoremassoziation wird zunächst versucht, die Konklusion des Quellproblems

$$Thm_Q : \lim_{n\to\infty} x(n) = 0 \to \lim_{n\to\infty} x(n)^2 = 0$$

auf die Konklusion des Zielproblems

$$Thm_Z : \lim_{n\to\infty} |x(n)| = 0$$

zu matchen. Der Matcher ist erfolgreich, allerdings kann die Methode des ersten Quellplanungsschrittes *Implies-I* nicht im Zielplan angewendet werden, da Thm_Z keine Implikation ist. Daher werden die Kinder von Thm_Q überprüft. Bereits die Methode *Def-I* von Schritt L_1 kann im Zielplan angewendet werden. Die Theoremassoziation ist beendet, und der Transfer kann ab dem Quellknoten L_1 beginnen. Der gefundene Match[3] ist

$$(\{\lambda z_1, z_2.{z_1}^{z_2}) \mapsto \lambda z_1, z_2.|z_1|\}, \emptyset, \emptyset, (0, 0, 2)).$$

Andere Ansätze erlauben nur eine sehr eingeschränkte Theoremassoziationsphase: Kolbe und Walter [KW94] ermöglichen als zusätzlichen Anfangsschritt nur das Einfügen einer Definitionsanwendung. Für Gleichheitsbeweise scheint dies ausreichend zu sein, aber für komplexere Beweise wie sie im ΩMEGA-System geplant werden, ist die Anwendung einer einzelnen Methode eine zu starke Einschränkung, da, wie aus dem Beispiel ersichtlich, beliebige (und oft auch mehr als eine) Methoden eingefügt werden müssen.

Koehler [Koe96] schlug vor, zu zeigen, daß aus dem Quelltheorem das Zieltheorem folgt, d.h. die Implikation $T_S \to T_T$ zu beweisen. Wenn erfolgreich, sollte dieser Beweis als Anfangsschritt in den Zielbeweis eingefügt werden. Dieses Vorgehen ist dann sinnvoll, wenn man davon ausgehen kann, daß die Implikation relativ einfach zu beweisen ist, eine Annahme die bei

---

[3] Substitutionen wie $x_v \mapsto x$ habe ich der Übersicht halber weggelassen.

komplexeren Beweisen aber nicht erfüllt sein muß. Ansonsten wird ein neues Problem erzeugt, welches womöglich schwieriger zu beweisen ist, als das Zielproblem selbst. Das Planungsproblem wird nur verlagert und nicht vereinfacht. Beispielsweise wäre für $X(n)^2 \mapsto |X(n)|$ die zu beweisende Implikation

$$(\lim_{n\to\infty} x(n) = 0 \to \lim_{n\to\infty} x(n)^2 = 0) \to (\lim_{n\to\infty} |x(n)| = 0)$$

und diese kann von ΩMEGA nicht gelöst werden.

4.3 Transfer der Quellbeweisschritte

Bevor ich den eigentlichen Transferalgorithmus erläutere, werde ich kurz auf die Verwaltung der Beziehungen zwischen Quell- und Zielknoten eingehen.

4.3.1 Verwaltung der Korrespondenzen

Während des Transfers ist es oft nötig zu bestimmen, welcher Knoten aus dem Zielplan einem bestimmten Knoten aus dem Quellplan entspricht. Um über diese Korrespondenzen zwischen Quell- und Zielknoten Buch zu führen, wird eine Knotentabelle benutzt, eine Datenstruktur, die jedem Quellknoten N_S eine Menge von Zielknoten $\mathcal{N}_\mathcal{T}$ zuordnet. $\mathcal{N}_\mathcal{T}$ soll dabei die Menge von Knoten darstellen, in welcher der zu N_S korrespondierende Knoten aus dem Zielplan zu finden ist. Grund der Verwendung dieser Datenstruktur ist die Überlegung, daß es nicht nötig ist, sämtliche Knoten des Zielbeweises zu betrachten, um den Knoten im Zielplan zu finden, der dem Knoten des Quellplans entspricht. Sondern es reicht aus, nur die Knoten des Zielplans zu berücksichtigen, die sich in den gleichen planungsbedingten Abhängigkeiten befinden, also von den jeweils korrespondierenden Methoden erzeugt wurden.

Die Operationen auf Knotentabellen sind

- *initialisiere*$((\mathcal{N}_{S_1}, \mathcal{N}_{T_1}) \ldots, (\mathcal{N}_{S_m}, \mathcal{N}_{T_m}))$:
  Bei der Initialisierung wird eine neue Knotentabelle kreiert und die

übergebenen Knotenmengen eingefügt. Die $\mathcal{N}_{S_j}$ bzw. $\mathcal{N}_{T_j}$ bestehen jeweils aus einer Menge von Quell- bzw. Zielknoten, und jeden Knoten aus $\mathcal{N}_{S_j}$ wird die Menge $\mathcal{N}_{T_j}$ zugeordnet.

- $f\ddot{u}ge\_hinzu(\mathcal{N}_1, \mathcal{N}_2, Knotentabelle)$:
  Diese Operation fügt die Menge $\mathcal{N}_1$ von Quellknoten und die dazu korrespondierende Menge an Zielknoten $\mathcal{N}_2$ in die Knotentabelle ein.

- $\ddot{a}ndere(K_1, \mathcal{N}_{neu}, Knotentabelle)$:
  Diese Operation erlaubt es, bestehende Korrespondenzen zu ändern. War vor der Ausführung dem Knoten K_1 die Menge $\mathcal{N}_{alt}$ zugeordnet, so sind K_1 anschließend die Zielknoten $\mathcal{N}_{neu}$ zugeordnet. War die Menge $\mathcal{N}_{alt}$ auch anderen Quellknoten zugeordnet, so ist diesen Quellknoten anschließend die Menge $\mathcal{N}_{alt}/\mathcal{N}_{neu}$ zugeordnet. Die Knoten $\mathcal{N}_{neu}$ werden also aus der Menge der korrespondierenden Knoten entfernt.

- $kor\_Kn(N, Knotentabelle)$:
  Durch diese Operation erhält man die Menge $\mathcal{N}$ der dem Quellknoten N zugeordneten Zielknoten. Spreche ich im folgenden von den korrespondierenden Knoten für N, so meine ich die Menge $\mathcal{N}$.

Eine Knotentabelle wird durch eine Hashtabelle realisiert, wodurch die Operationen $f\ddot{u}ge\_hinzu$, $\ddot{a}ndere$ und kor_Kn in konstanter Zeit und die Initialisierung in linearer Zeit abhängig von der Anzahl der übergebenen Mengen ausführbar sind.

Durch die Berücksichtigung der planungsbedingten Abhängigkeiten ergibt sich eine Einschränkung des Suchraums bei der Wahl der Knoten. Diese wäre theoretisch nicht nötig, da in jedem Schritt sämtliche Knoten des Zielplans überprüft werden könnten. Durch die Einschränkung wird aber der Transferalgorithmus erheblich effizienter, da weniger Formeln gematched werden müssen (siehe Kapitel 7).

Beispiel 4.3: In Abbildung 4.3 ist der Transfer eines Beweisplanes nach einigen übertragenen Planungsschritten dargestellt. Die Korrespondenzen

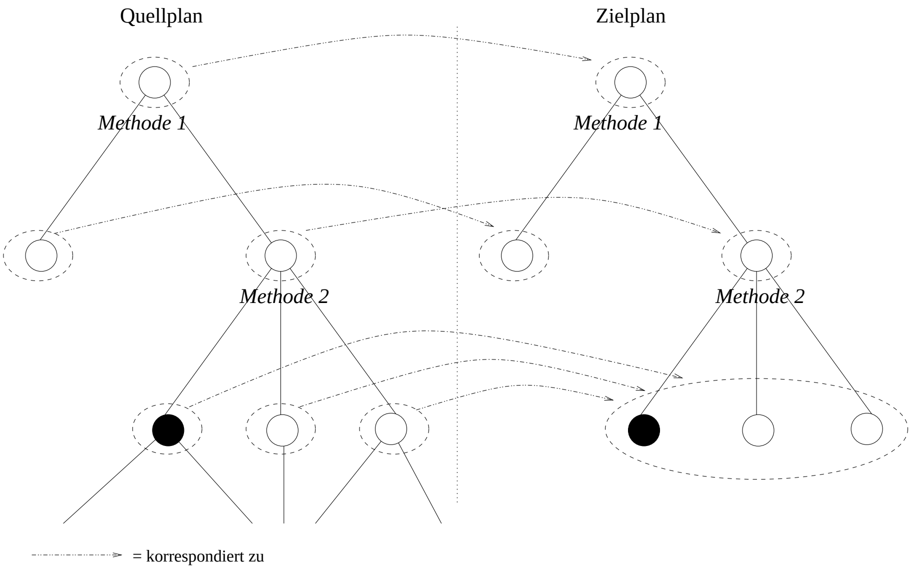


Abbildung 4.3: Beispiel für in einer Knotentabelle gespeicherte Korrespondenzen

sind durch gestrichelte Linien angedeutet. Nun soll der Schritt eines Kindes der bereits angewandten Methode *Methode 2* übertragen werden (der dunkel unterlegte Knoten). Es muß daher im Zielplan der Knoten bestimmt werden, der zu dem Quellknoten korrespondiert. Berücksichtigt man die planungsbedingten Abhängigkeiten, reicht es aus, nur die Knoten im Zielplan zu untersuchen, die von der Zielmethode erzeugt wurden, die der Quellmethode entspricht.

4.3.2 Der Transferalgorithmus

Der Transfer eines Quellplanungsschrittes besteht im Wesentlichen im Nachspielen der beiden Planungswahlmöglichkeiten: die Wahl der Methode und die Wahl der Knoten auf die die Methode angewendet werden soll.

Für Wahl der Methode wird angenommen, daß die Quellmethode auch im Zielplan sinnvoll anwendbar ist. Es wird daher dieselbe Methode ausgewählt. Sollte sich später herausstellen, daß diese Entscheidung falsch war, kann sie nachträglich geändert werden (siehe Abschnitt 4.4 über Adatationen).

Bei der Wahl der Knoten müssen die Knoten aus dem Zielplan gewählt werden, die den Knoten aus dem Quellplan entsprechen. Durch die Knotentabelle kann die Menge der für einen Quellknoten in Frage kommenden Zielknoten bereits eingeschränkt werden. Gibt es nach dieser ersten Auswahl immer noch mehrere Alternativen, so muß aus dieser Menge von Knoten derjenige ausgewählt werden, der am ehesten dem Quellknoten entspricht. Es muß daher bestimmt werden, welcher der Knoten aus dem Zielplan der ähnlichste bezüglich dem gegebenen Quellknoten ist. Unser erster Ansatz war der „Standardansatz“, die Bestimmung der Ähnlichkeit über das Matchen. In diesem Vorgehen wird der Quellterm auf die vorhandenen Zielterme gemachted und der kostengünstigste Match ausgewählt.

Allerdings mußten wir feststellen, daß trotz der Heuristiken des Matchers oft falsche Knoten ausgewählt wurden und die zu übertragende Methode nicht auf die ausgewählten Knoten anwendbar war, aber auf einen Knoten mit teurerem Match. Der Grund für die inkorrekte Auswahl lag aber nicht in unseren möglicherweise ungeeigneten Heuristiken, sondern darin, daß diese syntaktisch orientierte Sichtweise ignoriert, welchem Zweck eine Methode dient. Zum Beispiel ist die Bevorzugung von $div(2,x)$ auf $power(2,x)$ mit $div \mapsto power$ anstelle auf $div(times(3,x), plus(x,2))$ keine gute Wahl, wenn die aktuelle Methode des Quellteilplans Brüche und nicht Potenzen abschätzt. In einem anderen Planungskontext kann sie aber durchaus geeignet sein. Was also eigentlich geprüft werden sollte, ist abhängig von der Funktion der Methode. Zum Beispiel, welcher der zur Wahl stehenden Terme ist ein Bruch, oder welcher der Terme läßt sich als eine Linearkombination eines bestimmten anderen Termes darstellen. Nicht die konkrete syntaktische Form der Terme ist entscheidend, sondern ein abstrakteres Kriterium, das von der gerade zu übertragenen Methode abhängt. Dieses Kriterium ist in den Anwendungsbedingungen der Methode repräsentiert. In TOPAL geschieht die primäre Auswahl der Knoten also nicht rein syntaktisch durch Matchen, sondern gewissermaßen „planungskontextbezogen“

durch die Überprüfung der Knoten auf das Erfüllen der Anwendungsbedingungen der Methode.

Auf das Matchen wird zurückgegriffen, wenn die „planungskontextbezogen“ Auswahl nicht streng genug sein sollte (es also mehrere Knoten gibt, die die Anwendungsbedingungen erfüllen). Auch wenn kein Knoten die Anwendungsbedingungen erfüllt, wird das Matchen genutzt, um die Knoten zu bestimmen, auf denen Adaptationen durchgeführt werden sollen. In diesen Fällen stehen keine andere Unterscheidungsmöglichkeit zur Verfügung.

Tabelle 4.4 zeigt den detaillierten Transferalgorithmus. Vor dem Beginn des Transferzyklus findet eine Initialisierungsphase statt (siehe Tabelle 4.5), in der das Quellproblem geladen, die Theoremassoziierung durchgeführt und die Knotentabelle initialisiert wird. Bei der Initialisierung der Knotentabelle werden die sich aus der Theoremassoziation ergebenen Theoreme miteinander in Beziehung gesetzt und den restlichen Knoten des Quellplans die Prämissen des Zielproblems zugeordnet.

Nach der Initialisierung beginnt der Transferzyklus. Dieser wird durchlaufen, bis entweder der Quellplan komplett abgearbeitet wurde (d.h. der übergebene Quellplanungsschritt ist die leere Menge) oder keine offenen Ziele mehr im Zielplan vorhanden sind. In diesen Fällen wird die Analogie beendet und der vollständig oder teilweise gelöste Zielbeweisplan zurückgegeben. Andernfalls wird der nächste zu übertragende Schritt des Quellplans bestimmt.

Für die Übertragung eines Quellplanungsschrittes werden zunächst sämtliche möglicherweise dem Quellplanungsschritt entsprechende Zielplanungsschritte berechnet. Diese Menge von Planungsschritten wird dann auf die Menge der anwendbaren Schritte eingeschränkt. Sind mehrere Schritte anwendbar, wird durch Matching aus der Menge der anwendbaren Schritte der syntaktisch ähnlichste zum Quellplanungsschritt ausgewählt.

Tabelle 4.6 zeigt den Algorithmus zur Berechnung sämtlicher Schritte. Gäbe es jeweils nur einen korrespondierenden Zielknoten zu einem Quell-

Eingabe: Quellplanungsschritt, Quellplan, Zielplan, Matcher, Knotentabelle.
Ausgabe: (partieller) Zielplan.

```
Wenn Quellplanungsschritt = $\emptyset$ oder es keine offenen Ziele im Zielplan gibt,
 dann Ausgabe: Zielplan.
Sonst:
    // Bestimme Methode:
        $M$ := Methode aus Quellschritt.
    // Bestimme alle Schritte:
        alle_schritte := bestimme_alle_schritte(Quellplanungsschritt,
                                                Knotentabelle).
    // Bestimme anwendbare Schritte:
        anwendbare_schritte := {schritt | schritt $\in$ alle_schritte
                                und Methode von schritt ist auf
                                Konklusion und Prämissen von
                                schritt anwendbar}.
    // Wähle aus:
        Wenn |anwendbare_schritte| = 1,
         dann wahl_schritt := schritt $\in$ anwendbare_schritte.
        Wenn |anwendbare_schritte| > 1,
        dann wahl_schritt := schritt $\in$ anwendbare_schritte
                             mit geringsten Matchingkosten
                             bzgl. Quellschritt.
    // Reformuliere:
        Wenn anwendbare_schritte = $\emptyset$ dann reformuliere.
    // Anwendung:
        wende wahl_schritt im Zielplan an,
        aktualisiere Knotentabelle,
        transfer(nächster_schritt(Quellschritt), Quellplan,
                 Zielplan, Matcher, Knotentabelle).
```

Tabelle 4.4: detaillierter Transferalgorithmus

```
Eingabe: Zielplan.
Ausgabe: (partieller) Zielplan.

// Retrieval:
Bestimme und lade Quellplan für Zielproblem.
// Theoremassoziation:
(Matcher,$T_S$,$T_T$) := Theoremassoziation auf Quell- und Zielplan.
Initialisiere Knotentabelle.
// Transfer:
transfer(erster_schritt(Quellplan), Quellplan, Zielplan, Matcher, Knotentabelle).
```

Tabelle 4.5: Initialisierungsphase

knoten, so würde dem Quellplanungsschritt

$$(M, K, (P_{s_1}, \ldots, P_{s_m}))$$

der Zielplanungsschritt (T steht für eine Knotentabelle)

$$(M, kor\_Kn(K, T), (kor\_Kn(P_{s_1}, T), \ldots, kor\_Kn(P_{s_m}, T)))$$

entsprechen. Da es aber zu einem Quellknoten mehrere korrespondierende Zielknoten geben kann, muß für jede dieser Möglichkeiten ein eigener Zielplanungsschritt erzeugt werden. Abbildung 4.4 verdeutlicht das Vorgehen. Zuerst wird für jeden korrespondierenden Zielknoten K_j zur Quellkonklusion K ein Hilfsplanungsschritt $(M, K_j, \emptyset)$ erzeugt, der noch keine Prämissen enthält (Schritt 1). Dann wird das Kreuzprodukt über die Mengen der korrespondierenden Knoten der Quellprämissen gebildet, um sämtliche Kombinationen von möglichen Prämissen zu erfassen. Die Mengen von Prämissen $\mathcal{P}_1, \ldots, \mathcal{P}_n$ werden dann mit den Hilfsplanungsschritten kombiniert, indem für jeden Hilfsplanungsschritt $(M, K_j, \emptyset)$ die Menge von Planungsschritten $\{(M, K_j, \mathcal{P}_1), \ldots, (M, K_j, \mathcal{P}_n)\}$ erzeugt wird (Schritt 2).

Aus der Menge aller Schritte werden diejenigen Schritte (M, K, P) ausgewählt, für die die Methode M mit Konklusion K und Prämissen P die Anwendungsbedingungen erfüllt. Abhängig davon wieviele Schritte anwendbar sind, können drei Fälle auftreten:

Eingabe: Quellschritt $(M, K, \{P_{s_1}, \ldots, P_{s_m}\})$, Knotentabelle T.
Ausgabe: Menge von Zielschritten.

```
HILFSschritte:= $\{(M, K_1, \emptyset), \ldots, (M, K_i, \emptyset) | \{K_1, \ldots, K_i\} = kor\_Kn(K, T)\}$.
$\{\mathcal{P}_1, \ldots, \mathcal{P}_n\} := kor\_Kn(P_{s_1}, T) \times \ldots \times kor\_Kn(P_{s_m}, T)$
$alle\_schritte := \emptyset$.
Für alle $\mathcal{P} \in \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$:
        $alle\_schritte := alle\_schritte \cup \{(M, K_1, \mathcal{P}), \ldots, (M, K_i, \mathcal{P})\}$
                        $| \{(M, K_1, \emptyset), \ldots, (M, K_i, \emptyset)\} = hilfs\_schritte\}$
Ausgabe: $alle\_schritte$.
```

Tabelle 4.6: Bestimmung sämtlicher Schritte

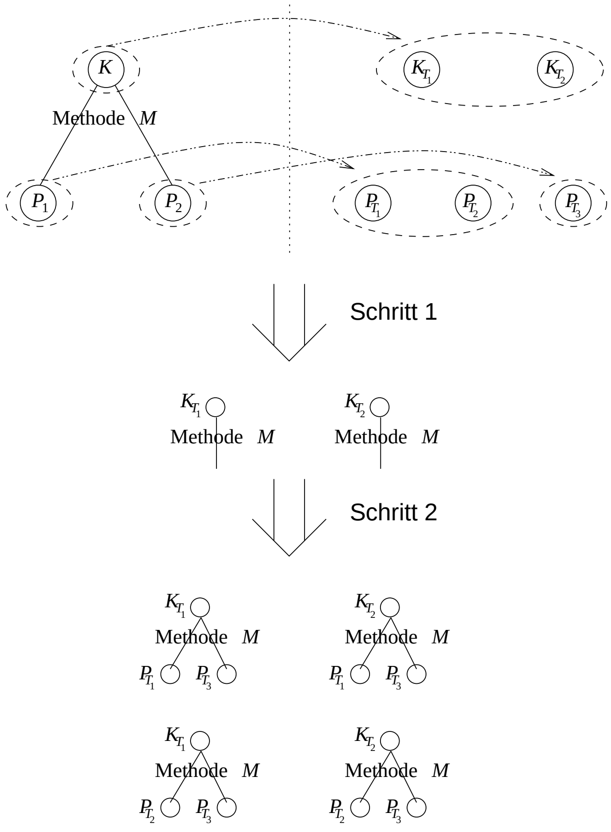


Abbildung 4.4: Berechnung aller Schritte

- Es gibt nur einen anwendbaren Schritt. Dieser wird im Zielplan angewendet und der Transfer wird mit dem nächsten Quellschritt fortgeführt.
- Es gibt mehr als einen anwendbaren Schritt. In diesem Fall werden der Quellschritt und die anwendbaren Zielschritte durch den erweiterten Matcher miteinander verglichen. Zunächst wird, wenn vorhanden, die Konklusion des Quellschrittes mit der des Zielschritts gematched. Gibt es einen Zielschritt mit einem günstigsten Match wird dieser ausgewählt. Andernfalls werden jeweils die Prämissen des Quellschrittes auf die Prämissen der Zielschritte gematched, bis eine Wahl möglich ist. Sind alle Prämissen abgearbeitet und ist immer noch keine Wahl möglich, wird zufällig ein Schritt ausgewählt. Der ausgewählte Schritt wird angewendet, der Transfer fortgeführt.
- Es gibt keinen anwendbaren Schritt, d.h. der Quellschritt ist nicht direkt auf den Zielplan übertragbar. In diesem Fall wird versucht, den Plan anzupassen, indem zum Beispiel Methoden eingefügt werden oder der Schritt übersprungen wird. Eine detaillierte Erläuterung der dann möglichen Aktionen gebe ich im nächsten Abschnitt.

Während des Transfers wird nach jeder Anwendung eines Schrittes die Knotentabelle aktualisiert: Bei einem Rückwärtsschritt werden die im Quellplan erzeugten Prämissen auf die im Zielplan entstandenen Prämissen eingeschränkt, bei einem Vorwärtsschritt analog die erzeugten Konklusionen.

Nach Ende des Transferzyklus sind entweder alle Schritte des Quellplans abgearbeitet oder alle Ziele im Zielplan geschlossen worden. Die Kontrolle wird wieder an den Planer zurückgegeben und es kann versucht werden, eventuell vorhandene offene Ziele automatisch oder interaktiv zu planen.

4.4 Adaptation des Quellbeweisplanes

Ziel der Adaptation (oder auch Reformulierung) ist es, die Lösung des Quellproblems so an die neue Planungssituation im Zielproblem anzupassen, daß die Lösung auf das Zielproblem übertragbar wird und dieses zumindest teilweise löst. Sowohl die Methoden als auch der Plan können dabei verändert werden. Drei Fragen müssen für die Realisierung einer Adaptationsphase geklärt werden: 1. Welche Objekte sollen adaptiert werden? 2. Wie werden die Adaptationen ausgewählt, die angewendet werden sollen? 3. Wann werden die Adaptationen ausgeführt?

- **Welche Objekte sollen adaptiert werden?**
  Als Objekte der Adaptation stehen sowohl der Quellplan selbst als auch die angewendeten Methoden zur Verfügung.

  Adaptation der Methoden bedeutet die Konklusion, Prämissen, Anwendungsbedingungen und das Beweisschema so abzuändern, daß die Methoden auf das Zielproblem anwendbar werden und der Quellplan mit den reformulierten Methoden das Zielproblem löst.

  Problematisch ist, daß das Beweisschema einer Methode sich auf einer relativ niedrigen Abstraktionsebene befinden kann und zum Teil uninstantiiert ist, da sich einige Symbole erst über Konklusion, Prämissen und Anwendungsbedingungen ergeben.

  Dies führt zum einen zum Problem des Ursprungs von Symbolen, d.h. der Zusammenhang zwischen den in der Methode und den in den Knoten auftretenden Symbolen ist nicht klar. So müssen die in den Knoten verwendeten Symbole nicht in der Methode auftreten: Zum Beispiel wurde aus der Theoremassoziation die Information gewonnen, daß f auf g gematched wurde. Diese Information ist bei der Reformulierung der Methode aber nicht verwendbar, wenn dort f nicht verwendet wird, sondern nur Funktionsvariablen, die erst bei der Anwendung der Methode durch f instantiiert werden. Das heißt, notwendige Adaptation können nicht durchgeführt werden. Weiterhin können in der Methode auftretende Symbole nicht ohne weiteres durch Adaptationen die sich aus dem Theoremmatch ergeben haben, reformuliert werden. Auch wenn zum Beispiel eine Reformulierung bewirkt, daß $+$ durch $*$ ersetzt werden soll, so kann die Anwendung dieser Reformulierung in der Methode falsch sein, wenn das Symbol $+$ durch Anwendung eines

Lemmas wie der Dreiecksungleichung[4] in der Methode erzeugt wird, also von den Theoremen unabhängig ist.

Zum anderen ist es schwer zu beurteilen, ob die Adaptation sinnvoll ist, die Methode also anwendbar bleibt und das Beweisschema zu einem korrekten und das Problem lösenden Beweis expandiert, da die Methode uninstantiiert weder auf Anwendbarkeit noch auf Korrektheit überprüfbar ist.

Von Interesse ist die Adaptation von Methoden vor allem, da dadurch neue Methoden konstruiert werden können (siehe [Mel98c, Sch96]). In diesen Ansätzen wurden aber die Fragen zur Überprüfbarkeit und dem Ursprung von Symbolen nicht berücksichtigt.

In meiner Diplomarbeit wird die Reformulierung von Methoden nicht weiter verfolgt, da der Fokus auf der Analogie auf Planebene liegt, also auch Adaptationen auf Planebene untersucht werden. Dabei stehen als Reformulierungen Überspringen, Einfügen und Ersetzen von Planungsschritten zur Verfügung. Die Reformulierung geschieht demnach auf einem sehr hohen Abstraktionsniveau. Da die Methoden selbst nicht verändert werden, entfallen die damit verbundenen Probleme des Ursprungs von Symbolen und die Überprüfung des Beweisschemas. Durch das Arbeiten auf Planebene kann zudem der Planer benutzt werden, um geeignete Adaptationen zu finden und anzuwenden.

Fraglich erscheint zunächst, ob ohne Änderung von vorhandenen Methoden tatsächlich neue Probleme gelöst werden können. Jedoch ist eine Forderung des Beweisplanens, abstrakte Methoden zu entwerfen, die auf eine große Menge von Problemen anwendbar sind. Ist diese Bedingung erfüllt, so kann tatsächlich eine große Anzahl von neuen Beweisen analog geführt werden, wie unsere Evaluation (siehe Kapitel 7) zeigen wird.

- **Wie werden die Adaptationen ausgewählt, die angewendet werden sollen?**

---

[4] Die Dreiecksungleichung besagt, daß $|A+B| \leq |A| + |B|$.

Für die Auswahl von Adaptationen gibt es verschiedene Ansätze: Die erste Möglichkeit ist, die Adaptationen nur aufgrund lokaler Unterschiede zwischen Quell- und Zielknoten, also über das Matching, zu bestimmen. Dies ist allerdings problematisch, da zusätzlich zu den lokalen Unterschieden folgende Faktoren eine Rolle spielen:

- **Die Art des Knotens:** Abhängig davon, ob der Knoten ein offenes Ziel oder eine Annahme ist, sind für denselben Unterschied verschiedene Aktionen nötig. Soll zum Beispiel eine zusätzliche Konjunktion im Ziel eliminiert werden, so muß für einen offenen Knoten die Methode *And-I* angewendet werden, für ein Annahme die Methode *And-I*.
- **Die Position der Unterschiede:** Abhängig von der Position des Unterschiedes zwischen Quell- und Zielknoten können die Stellen an denen der Quellplan reformuliert werden muß, variieren. Ist C zum Beispiel eine zusätzliche Konjunktion in einer Teilformel der Definition, wie $(F \equiv A \wedge C \rightarrow B)$, so sollten nur die Teile des Quellplanes reformuliert werden, die A betreffen.
- **Die zu übertragende Methode:** Die nötigen Adaptationen können je nach Methode unterschiedlich sein. So kann zum Beispiel eine zusätzliche Konjunktion C in einer Annahme $A \wedge C$ den Transfer behindern, wenn der Quellschritt A als Prämisse braucht, oder es kann die zusätzliche Konjunktion C nötig sein, wenn zum Beispiel *Modus-Pones* mit der Implikation $(A \wedge C \rightarrow B)$ auf die Annahme angewendet werden soll.

Es reicht also nicht aus, nur die lokalen, sich aus dem Matchen ergebenen Unterschiede zu betrachten.

Im ABALONE [MW99], dem Analogiesystem des Beweisplaners für induktive Beweise CLAM, werden zusätzlich zum Matching auch verletzte Vorbedingungen und Rechtfertigungen berücksichtigt, um die Reformulierung auszuwählen. Ist zum Beispiel die Vorbedingung, daß eine Formel C „trivial“ beweisbar ist, verletzt, so wird C als Lemma vorgeschlagen. In ABALONE gibt es ca. zehn Regeln, die ausdrücken, welche Adaptation unter welchen Bedingungen durchgeführt werden sollen. Um solch gezielte Regeln zu entwerfen, ist eine genaue a-priori Analyse sowohl der Domäne als auch der Strategie nötig. In dieser müssen die Methoden, Rechtfertigungen und möglichen Unterschiede der Knoten untersucht und daraus Aktionen, die den Transfer wieder ermöglichen, abgeleitet werden. Werden dann Methoden hinzugefügt oder geändert, so muß die Analyse erneut durchgeführt werden. In einem System wie CLAM, das nur eine Beweisstrategie (in CLAMs Fall die Induktion) und die dafür nötigen Methoden zur Verfügung stellt, ist dieser Ansatz anwendbar, da mit einer fixen Menge von Methoden gearbeitet wird.

Jedoch ist einer der Vorteile des ΩMEGA-Systems dessen Erweiterbarkeit auf neue Theorien. Neue Methoden und Anwendungsbedingungen können von jedem Benutzer hinzugefügt werden. In einer solchen Umgebung kann eine a-priori Analyse nicht vorausgesetzt werden. Das Ziel dieser Diplomarbeit war es daher, allgemeine Heuristiken für die Auswahl von Adaptationen zu finden und gleichzeitig die Möglichkeit domänenspezifischer Adaptationsregeln zu bieten.

- **Wann werden die Adaptationen ausgeführt?**
  Eine weitere Designentscheidung betrifft den Zeitpunkt der Reformulierungen. Reformulierungen können sowohl vor als auch während des Transfers erfolgen.

  Die Adaptation vor dem Transfer durchzuführen [Sch96] ist problematisch, da zu diese Zeitpunkt die Information aus der Planungssituation des Zielplanes nicht zur Verfügung steht. Es wären also zwei Analysephasen nötig, eine erste um zu erkennen, ob und wie der Quellplan reformuliert werden muß, eine zweite während des Transfers, die erkennt, ob die Planungsschritte anwendbar sind.

  Überlagert man Adaptation und Transfer, so ist die erste Analyse unnötig. Es ist ausreichend, während des Transfers die Analyse durchzuführen und erst bei einem fehlgeschlagenen Transferschritt Adaptationen anzuwenden.

4.4.1 Der Algorithmus

Tabelle 4.7 zeigt den Adaptationsalgorithmus. Zuerst wird bestimmt, welcher der möglichen Zielschritte dem zu übertragenen Quellschritt entspricht. Da das Erfüllen der Anwendungsbedingungen nicht als Unterscheidungskriterium herangezogen werden kann (kein Schritt ist anwendbar, der Grund für die Suche nach Adaptationen), wird die Auswahl durch den erweiterten Matcher getroffen.

Ist der Schritt bestimmt, werden vier Adaptationsmöglichkeiten der Reihe nach auf sinnvolle Anwendbarkeit überprüft: Anwendung von Domänenwissen (Abschnitt 4.4.2), Lemmavorschlag (Abschnitt 4.4.3), Überspringen des Schrittes (Abschnitt 4.4.4) und Einfügen von Schritten (Abschnitt 4.4.5). Obwohl das Überspringen immer durchgeführt wird, wenn die anderen Adaptationen nicht erfolgreich sind, ist es sinnvoll, bereits vorher auf dessen Anwendbarkeit zu testen, da das Einfügen von Schritten im Gegensatz zum Überspringen eine teure Adaptation ist. Dadurch wird der Transfer deutlich beschleunigt.

Kann keine der Adaptationen angewendet werden, so wird der Planungsschritt übersprungen und der Transfer weitergeführt. Der Transfer wird nicht komplett abgebrochen, da vielleicht andere Teile des Quellplans übertragen werden können.

In den nächsten vier Abschnitten erläutere ich die verschiedenen Adaptationen im Detail.

4.4.2 Anwenden von Domänenwissen

Die erste Möglichkeit der Adaptation besteht in der Auswertung von Domänenwissen. Dieses Adaptationsdomänenwissen liegt in der Form von Kontrollregeln vor, die von dem Kontrollregelinterpreter des Planers ausgewertet werden.

Im Bedingungsteil der Kontrollregeln können durch Metaprädikate Informationen über die momentane Planungs- und Analogiesituation gewonnen werden. Sämtliche in der Planung verwendeten Metaprädikate können

Eingabe: Zielschritte, Quellschritt, Matcher, Quellplan, Zielplan,
Knotentabelle T.
Ausgabe: (partieller) Zielplan

```
// Bestimme Planungsschritt:
schritt := kostengünstigster Schritt aus Zielschritten bzgl. Quellschritt.
// Domänenwissen:
aktion := Domänenwissenvorschlag.
Wenn aktion,
  dann wende aktion auf Zielplan an, fahre mit Transfer fort.
// Lemmavorschlag:
fehlender_knoten := fehlt_knoten(schritt, Quellschritt, $T$).
Wenn fehlender_knoten,
    dann lemma_vorschlag( fehlender_knoten,Zielplan, Quellplan,
                                  Quellschritt, Matcher, $T$).
// Überspringen:
späterer_schritt := späterer_schritt_anwendbar( schritt, Zielplan,
                                                      Quellplan, $T$).
Wenn späterer_schritt,
  dann überspringe( späterer_schritt, Zielplan, Quellplan,
                          Quellschritt, Matcher, $T$).
// Einfügen:
einfügung := suche_Einfügung(Quellschritt, Zielplan, Quellplan).
Wenn einfügung,
  dann füge_ein( einfügung, Zielplan, Quellplan,Quellschritt,
                    Matcher, $T$).
// Keine Adaptation gefunden:
überspringe( schritt, Zielplan, Quellplan, Quellschritt, Matcher, $T$).
```

Tabelle 4.7: Adaptations-Algorithmus

benutzt werden, zusätzlich sind implementiert:

- *current_source_method(M)*: Falls M instantiiert ist, wird geprüft, ob M die aktuelle Quellmethode ist, ansonsten wird M an die Methode des gerade zu übertragenden Quellschrittes gebunden.
- *extended_match_contains(x,y)*: Dieses Metaprädikat überprüft, ob der erweiterte Match entweder die Substitution $x \mapsto y$ oder die Termabbildung $x \mapsto_T y$ enthält.
- *violated_application_condition(C)*: Falls C instantiiert ist, wird geprüft, ob C die Anwendungsbedingung ist, die für den Fehlschlag des Transfers verantwortlich ist, ansonsten wird C an diese Anwendungsbedingung gebunden.

Folgende Aktionen stehen als Adaptation zur Verfügung:

- *replace_current_method_by(M)*: Ersetzt die übertragene Methode durch eine andere Methode M.
- *skip_step*: Überspringt den aktuellen Schritt (siehe Abschnitt 4.4.4).
- *insert_steps(Ms)*: Fügt die Schritte *Ms*, wobei *Ms* eine Liste von Methoden ist (siehe Abschnitt 4.4.5).

Ein Beispiel für eine Adaptationskontrollregel zeigt Tabelle 4.8. Diese prüft, ob im Quellschritt mit der Methode $Solve*_{<}$ eine Kleiner-Ungleichung abgeschätzt wurde, während im Ziel eine Größer-Ungleichung auftritt, und ersetzt in diesem Fall die Abschätzungsmethode.

Durch das Adaptationsdomänenwissen, das in den Kontrollregeln ausdrückbar ist, ist es möglich, Wissen zu verwenden, das aus einer Domänenanalyse gewonnen wurde. Dadurch können gezielte Reformulierungen wie in ABALONE [MW99] realisiert werden. Allerdings wird eine solche Analyse in unserem Ansatz nicht vorausgesetzt, da durch die drei folgenden Heuristiken allgemeine Probleme des Transfers erkannt und passende Adaptationen vorgeschlagen werden können.

```
(control-rule
        (kind adaption)
        (if (and current_source_method(solve*<)
                 extended_match_contains(<,>)))
        (then select(replace_current_method_by(solve*>))))
```

Tabelle 4.8: Beispiel einer Adaptationskontrollregel

Eingabe: Quellschritt, Knotentabelle.
Ausgabe: Knoten K oder $false$.

Wenn für $P \in \ominus$Prämissen von Quellschritt gilt:
  $\mathit{kor\_Kn}(P,\text{Knotentabelle}) = \emptyset$,
dann *Ausgabe:* P,
sonst *Ausgabe:* $\emptyset$.

Tabelle 4.9: Funktion *fehlender_knoten*

4.4.3 Vorschlagen eines Lemma

Kann ein Planungsschritt nicht übertragen werden, da eine benötigte Vorbedingung im Zielplan fehlt, so kann diese Vorbedingung dem Zielplan hinzugefügt und dann der Schritt übertragen werden. Später muß die hinzugefügte Vorbedingung bewiesen werden.

In der Funktion *fehlender_knoten* (siehe Tabelle 4.9) wird geprüft, ob für eine Quellannahme, die im gerade zu übertragenden Planungsschritt eine Vorbedingung ist, kein Knoten im Zielplan existiert (die Knotentabelle also die leere Menge zurückgibt). In diesem Fall wird auf die Quellannahme der erweiterte Match angewendet. Ist die Benutzerinteraktion aktiviert, kann der Benutzer entscheiden, ob diese Formel als offenes Lemma in den Zielplan eingefügt oder der momentane Schritt übersprungen werden soll. Ohne Benutzerinteraktion wird der Knoten automatisch eingefügt. Danach wird der Quellschritt übertragen.

Für Vorbedingungen aus dem Quellplan, die Teil*ziele* sind, werden keine Knoten vorgeschlagen. Ein fehlendes Ziel deutet darauf hin, daß der gerade zu übertragende Teilbeweis im Zielplan keine Entsprechung hat. Er ist also überflüssig und kann übersprungen werden.

Beispiel 4.4: Weiterführung von $X(n)^2 \mapsto |X(n)|$
Für die Übertragung des vierten Schrittes, der Anwendung der Methode *Def-E*, kann kein entsprechender Knoten im Zielplan gefunden werden. Es fehlt der korrespondierenden Zielknoten der Quellannahme L_2, da die Formalisierung des Zielproblems unvollständig ist. Durch die Anwendung des Matches mit der Substitution

$$\{\lambda z_1, z_2.{z_1}^{z_2}) \mapsto \lambda z_1, z_2.|z_1|, n_v \mapsto n, =_v \mapsto =, 0_v \mapsto 0, x_v \mapsto x\}$$

auf den Quellknoten

$$L_2 : \lim_{n_v \to \infty} x_v(n_v) =_v 0_v$$

wird

$$\mathit{lemma} : \lim_{n \to \infty} x(n) = 0$$

als einzufügender Knoten vorgeschlagen. Nach dem Einfügen kann der Schritt übertragen werden, der Transfer wird weitergeführt.

Diese Form des Lemmavorschlags, die eine (erweiterte) Substitution auf eine Quellformel anwendet, um eine Zielformel zu erzeugen, wird in zahlreichen Systemen verwendet [Sch96, KW94, MW99]. Sie beruht auf der Annahme, daß der Match zwischen Quell- und Zielknoten ausreichende Informationen zur Verfügung stellt. Im Abschnitt 4.3.2 habe ich gezeigt, daß die Informationen des Matchings nicht ausreichen, um die Knoten für den Transfer auszuwählen, und auf planungskontextbezogene Informationen zurückgegriffen wird. Ebenso sollten im Fall des Lemmavorschlags zusätzliche Information genutzt werden, die sich aus der Planungssituation im Zielbeweis ergeben. Dies ist über die Einführung von *Metavariablen* [HBS92] möglich. Dabei wird in eine Formel eine Metavariable eingesetzt, für die im Laufe des Beweises über die Anwendungsbedingungen der Methoden Einschränkungen gesammelt werden. Aus den gesammelten Einschränkungen kann dann ein konkreter Term konstruiert werden, durch den die Metavariable instantiiert wird.

Leider war die Behandlung von Metavariablen im ΩMEGA-System zur Zeit der Erstellung dieser Diplomarbeit noch nicht so weit fortgeschritten, daß dieser Ansatz implementiert werden konnte. Das Vorschlagen von Lemmas ist demnach nur in der „einfacheren“ Form implementiert.

4.4.4 Überspringen von Schritten

Ein weiterer Grund, weshalb ein Planungsschritt nicht übertragbar ist, kann darin bestehen, daß dieser Schritt im Zielplan überflüssig ist. In diesem Fall kann er ausgelassen und der Transfer mit dem nächsten Schritt fortgeführt werden.

Tabelle 4.10 zeigt die Funktion, die überprüft, ob ein Schritt übersprungen werden kann. Wir unterscheiden zwischen Vorwärts- und Rückwärtsschritten. Zum einen kann ein einzelner Rückwärtsschritt überflüssig und einer seiner Nachkommen anwendbar sein. Dies kann zum Beispiel auftreten, wenn durch den Quellschritt ein Knoten vereinfacht wird, diese Vereinfachung aber im Ziel nicht nötig ist. Zum anderen braucht ein Rückwärtsschritt nicht übertragen zu werden, wenn seiner Konklusion kein Knoten im Zielplan zugeordnet ist, das heißt wenn es für diese Konklusion keinen entsprechenden offenen Knoten im Zielplan gibt. Der Teilplan, der im Quellbeweis diese Konklusion geschlossen hat, kann also übersprungen werden. Dieser Fall kann beispielsweise auftreten, wenn das Zielproblem einfacher als das Quellproblem ist und weniger Teilziele erzeugt wurden.

Um zu testen, ob der Rückwärtsschritt übersprungen werden kann, werden zunächst seine Nachkommen bis zur Tiefe *skip_limit* (standardmäßig 3) bestimmt. Diese Nachkommen werden dann auf Übertragbarkeit geprüft: Die Methoden der Nachkommen werden auf die Konklusion des Zielschrittes und den Zielknoten, die zu den Prämissen des gerade zu überprüften Nachkommens korrespondieren, angewendet. Wir wenden die Methoden nicht nur auf die Prämissen des aktuellen Zielschrittes an, da die Methoden Prämissen benötigen können, die nicht in den Prämissen des Zielschrittes enthalten sein müssen, sondern anderen Annahmen im Zielbeweis entsprechen können. Ist die Methode eines Nachkommen im Zielplan anwendbar, so kann der nicht übertragbare Rückwärtsschritt übersprungen werden. Die neu gefundene Korrespondenz zwischen der Konklusion des anwendbaren Nachkom-

```
Eingabe: Zielschritt $(M_Z, K_Z, \mathcal{P}_Z)$, Quellschritt $(M_Q, K_Q, \mathcal{P}_Q)$, Matcher,
         Quellplan, Zielplan, Knotentabelle $T$.
Ausgabe: $true$,$false$ oder ein Tupel von Knoten.

Wenn $M_Z$ eine Rückwärtsmethode ist,
 dann
     Wenn $kor\_Kn(K_Z, T) = \emptyset$ // kein Zielknoten
      dann Ausgabe: $True$.
     Sonst:
         // Bestimme Nachkommen:
         $kinder$ := Nachkommen von $K_Q$ bis Tiefe $skip\_limit$.
         // Teste auf Anwendbarkeit:
         Wenn für ein $kind \in kinder$ gilt:
             Methode von $kind$ ist auf $K_Z$ und
              $kor\_Kn(Prämissen(kind), T)$ anwendbar,
           dann Ausgabe: $(kind, K_Z)$.
         sonst  // kein Nachkomme ist anwendbar
           Ausgabe: $False$.
Sonst // $M_Z$ ist Vorwärtsmethode
     // Bestimme Vorfahren:
     $eltern$ := Vorfahren von $\mathcal{P}_Q$ bis Höhe $skip\_limit$.
     // Teste auf Anwendbarkeit:
     Wenn für ein $elt \in eltern$ gilt:
         Methode von $elt$ ist auf $\mathcal{P}_Z$ anwendbar,
      dann Ausgabe: $true$
     sonst  // kein Vorfahre ist anwendbar
       Ausgabe: $false$.
```

Tabelle 4.10: Anwendbarkeit der Überspring-Adaptation

```
Eingabe: $überspring\_ergebnis$, Zielschritt, Quellschritt, Matcher, Quellplan,
         Zielplan, Knotentabelle $T$.
Ausgabe: (partieller) Zielplan

// Aktualisiere Knotentabelle:
Wenn $überspring\_ergebnis = (N_S, N_T)$,
  dann $T := ändere(N_S, \{N_T\}, T)$.
// Führe Transfer fort:
$transfer$(Quellplan, Zielplan, $nächster\_Schritt$(Quellschritt),Matcher,$T$).
```

Tabelle 4.11: Anwendung der Überspring-Adaptation

men und der Konklusion des Zielschrittes wird zurückgegeben, um später die Knotentabelle zu aktualisieren.

Vorwärtsschritte werden auf ähnliche Weise auf Überspringen überprüft: Die Vorfahren bis zur Höhe *skip_limit* werden bestimmt, und deren Methoden auf Anwendbarkeit getestet. Ist die Methode eines Vorfahrens anwendbar, so kann der Vorwärtsschritt übersprungen werden.

Kann ein Vorwärts- oder Rückwärtsschritt übersprungen werden, so wird die Knotentabelle aktualisiert und der Transfer mit dem nächsten Schritt fortgeführt (siehe Tabelle 4.11).

Beispiel 4.5: Weiterführung von $X(n)^2 \mapsto |X(n)|$
Abbildung 4.5 zeigt einen schematischen Ausschnitt des Transfer während der Übertragung des neunten Planungsschrittes, der Anwendung der Methode *Complex-Estimate<*. Auf den Zielknoten

$$L_{5'} : \ ||x(n)| - 0 < \epsilon$$

ist *Complex-Estimate<* nicht anwendbar. Daher werden die Methoden der Kinder von L_5 auf Anwendbarkeit geprüft. Die Methode *Simplify* von Knoten L_{12} ist aber anwendbar. Der Schritt von L_5 kann also übersprungen werden. Im späteren Verlauf des Transfers werden die Schritte von Knoten L_{10} und L_{11} betrachtet. Da sie keinen korrespondierenden Knoten im Zielplan haben, können die dazugehörigen Teilpläne übersprungen werden.

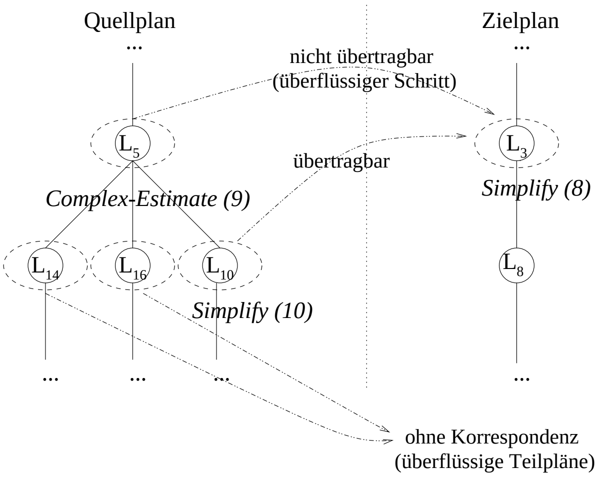


Abbildung 4.5: Beispiel für die Überspring-Adaptation

Nach diesen Adaptationen wird der restliche Quellplan ohne Schwierigkeiten übertragen. Alle offenen Knoten des Zielproblems können geschlossen werden. Den Beweisplan für das Zielproblem zeigt Tabelle 4.12.

4.4.5 Einfügen von Schritten

Oft ist ein Planungsschritt nicht direkt übertragbar, sondern erst nachdem zusätzliche Schritte im Ziel durchgeführt werden, wie zum Beispiel eine Normalisierung. Kennt man die speziellen Bedingungen unter denen zuerst eine bestimmte Methode angewendet werden muß und anschließend der Schritt übertragen werden kann, so kann man dieses Wissen in den Adaptationsregeln kodieren.

Um auch die Fälle behandeln zu können, die nicht von Adaptationsregeln abgedeckt werden, habe ich einen Einfügungsalgorithmus entworfen, der mit Hilfe des Planers von ΩMEGA eine geeignete Folge von einzufügenden Schritten plant. Ist ein Zielschritt $(M, K, \mathcal{P})$ nicht anwendbar, so wird ein neuer, gegenüber dem aktuellen Zielplanungsraum reduzierter Planungsraum *hp* kreiert. Dann wird versucht, in *hp* eine Folge von Planungsschritten

$Thm_Z : \lim_{n\to\infty} |x(n)| = 0$ **Theoremassoziation**

Def-I (1)

$\downarrow$

$L_1 : \forall\epsilon\exists k\forall n(0 < \epsilon \to (n > k \to (||x(n)| - 0| < \epsilon)))$

Skolemize (2)

$\downarrow$

$L_2 : (0 < \epsilon \to (n > k \to (||x(n)| - 0| < \epsilon)))$

Normalize (3)

$\downarrow$

$L_3 : (||x(n)| - 0| < \epsilon)$ **Überspringen**

Simplify (8)

$\downarrow$

$L_8 : |x(n)| < \frac{\epsilon}{2\cdot M}$

Solve∗ $<$ (9)

```
Eingabe: Zielschritt $(M, K, \mathcal{P})$, Zielplan.
Ausgabe: einzufügender Plan oder $false$.

Wenn $M$ Rückwärtsmethode,
  dann
    $hp$ := Planungsraum mit Konklusion:= $K$,
                             Annahmen:= $hyps(K) \cup \mathcal{P}$.
    $plane\_adaptation$(alle Methoden, Zielschritt,$hp$).
Sonst
    $hp$ := Planungsraum mit Konklusion:= bel. offenes Ziel aus
                                          Zielplan,
                             Annahmen:= $\mathcal{P}$.
    $plane\_adaptation$(alle Methoden, Zielschritt,$hp$).
```

Tabelle 4.13: Bestimmung des reduzierten Planungsraums

zu finden, die den Zielschritt anwendbar macht.

Tabelle 4.13 zeigt die Funktion, die den reduzierten Planungsraum *hp* erzeugt. *hp* enthält nicht alle Knoten aus dem aktuellen Zielplanungsraum, sondern nur die, die bezüglich des anzuwendenden Zielschrittes relevant sind: Ist der Schritt ein Rückwärtsschritt, so enthält der Hilfsplanungsraum den zu schließenden offenen Knoten K, zu K gehörenden Annahmen und die Prämissen $\mathcal{P}$ des Schrittes. Ist der Schritt ein Vorwärtsschritt, enthält *hp* die Prämissen $\mathcal{P}$ des Schrittes, und da jeder Planungsraum ein offenes Ziel benötigt, wird für *hp* ein beliebiges Ziel ausgewählt.

In dem Planungsraum *hp* wird dann mit der Funktion *plane_adaptation* (siehe Tabelle 4.14) durch Breitensuche geplant, bis der Zielschritt anwendbar wird oder ein Tiefenlimit (standardmäßig 3) überschritten wird. Der erste Parameter der Funktion, die Liste von Methodenlisten $(\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n)$ gibt die Menge der zu überprüfenden Methodenanwendungen an. Beim ersten Aufruf von *plane_adaptation* werden alle verfügbaren Methoden übergeben. Jedes $\mathcal{M}_i$ steht für eine Folge von Methoden, die angewendet werden und einen Planungszustand erzeugen, der darauf überprüft wird, ob eine erfolgreiche Terminierungsbedingung erreicht wurde. Es gibt zwei erfolgreiche Terminierungsbedingungen: Die erste ist die Anwendbarkeit des Zielschritts.

Es wurde also eine Folge von Methoden gefunden, die die Fortführung des Transfers ermöglicht. Die zweite Endbedingung ist das Schließen aller offenen Ziele in *hp*. In diesem Fall braucht der Zielschritt nicht mehr angewendet zu werden, da das dazugehörige Ziel auch ohne ihn geschlossen werden konnte. Dieser Fall gilt nur für Rückwärtsschritte, da bei Vorwärtsschritten ein beliebiges Ziel gewählt wurde, das zufällig geschlossen werden könnte. Ist eine Endbedingung erfüllt, so gibt *plane_adaptation* den Planungsraum *hp* zurück. Ist keine Endbedingung erfüllt, so werden die Folgezustände berechnet (siehe Tabelle 4.15), d.h. für jede der zur Verfügung stehenden Methoden $(M_1, \ldots M_m)$ wird eine neue Folge von Methodenanwendungen erzeugt, indem jeweils M_1 bis M_m an $\mathcal{M}_1$ angehängt werden. Diese neuen Folgen werden dann gemäß der Breitensuche an die zu überprüfenden Methodenanwendungen angehängt. Waren die Methoden der Liste $\mathcal{M}_1$ nicht anwendbar, so wird *plane_adaptation* rekursiv mit den restlichen Folgen von Methoden $(\mathcal{M}_2, \ldots, \mathcal{M}_n)$ aufgerufen bis das Tiefenlimit überschritten wurde oder keine Methodenfolgen mehr anwendbar sind. In diesem Fall gibt die Funktion $false$ zurück. Die Tiefe wird bestimmt über die Anzahl der angewendeten Methoden, also $length(\mathcal{M}_1)$.

Wurde in *hp* eine Folge von Methodenanwendungen gefunden, so wird der Plan in den Zielplan eingefügt, der Zielschritt übertragen und der Transfer mit der nächsten Methode weitergeführt. Ansonsten wird der Zielschritt übersprungen.

Die Entscheidung, einen Hilfsplanungsraum zu kreieren und nicht direkt in dem Zielplan die Adaptationen zu planen, wurde getroffen, um den Suchraum möglichst klein zu halten. Es werden beim Planen nur die Knoten betrachtet, die für den zu übertragenden Planungsschritt nötig sind (also die $\ominus$-Knoten des Schritts) und gegebenenfalls deren Annahmen.

Die Suchstrategie Breitensuche wurde anstelle von Strategien mit günstigeren Speicherbedarf (wie Tiefensuche) gewählt, da wir möglichst wenig und möglichst keine unnötigen Methoden anwenden wollten. Bei Tiefensuche kann zum Beispiel zufällig eine Vorwärtsmethode auf die Annahmen angewendet werden und erst anschließend die vielleicht allein ausreichende

```
Eingabe: Liste der zu prüfenden Methodenfolgen $M := (\mathcal{M}_1, \ldots, \mathcal{M}_n)$,
         anzuwendender Zielschritt $schritt$, Hilfsplan $hp$.
Ausgabe: einzufügender Plan oder $false$.

Wenn $\mathcal{M} \neq \emptyset$ und $length(\mathcal{M}_1) < adapt\_limit$,
 dann
      wenn Anwendung von $\mathcal{M}_1$ in $hp$ erfolgreich,
       dann
          wenn $übertragbar(schritt, hp)$
                oder ($rückwärts\_anwendung$
                      und keine offenen Ziele in $hp$)
           dann Ausgabe: aktueller Plan,
          sonst
              $backtracking$,
              $plane\_adatation( cons( rest(\mathcal{M}),$
                                       $folgezustände(\mathcal{M}_1)),$
                                 $schritt, hp).$
      Sonst $plane\_adaptation(rest(\mathcal{M})), schritt, hp).$
Sonst Ausgabe: $false$.
```

Tabelle 4.14: Funktion *plane_adaptation*

```
Eingabe: angewendete Methoden $\mathcal{M}_A$.
Ausgabe: Liste von Folgezuständen $folgezustände$.

$folgezustände := \emptyset$.
$\mathcal{M}_V$ := alle zu Verfügung stehenden Methoden.
Für alle $M \in \mathcal{M}_V$
      $folgezustände := cons(cons(\mathcal{M}_A, M)), folgezustände).$
Ausgabe: $folgezustände$.
```

Tabelle 4.15: Berechnung der Folgezustände

Rückwärtsmethode.

Eine interessante Frage ist, ob nicht zielgerichteter nach den einzufügenden Schritten gesucht werden könnte. Das Ziel der Einfügungsadaptation ist, die $\ominus$-Knoten des zu übertragenden Schrittes mit Hilfe von Methoden so umzuformen, daß die Methode des Schrittes anwendbar wird. Das heißt, es müssen die Anwendungsbedingungen erfüllt und die Methodenknoten auf die $\ominus$-Knoten matchbar sein.

Eine Möglichkeit wäre, die Unterschiede zwischen den Knoten des Zielplanungsschrittes und denen des Quellschrittes zu berechnen, und nach jeder Methodenanwendung zu überprüfen, ob die Unterschiede sich verringert haben. Eine andere Möglichkeit wäre, nur die Methoden anzuwenden, die tatsächlich zur Reduzierung der Unterschiede beitragen. In beiden Fällen könnten die Unterschiede zum Beispiel durch die Kosten des erweiterten Matches berechnet werden. Allerdings würden wir dadurch wiederum den Planungskontext außer acht lassen und auf das gleiche Problem wie beim Nachspielen des Knotenauswahlpunktes stoßen: Durch das Matching allein können nur sehr unsichere Aussagen getroffen werden. Zudem ist das Matching teuer, so daß die unsichere Zielgerichtetheit mit einem hohen Zeitaufwand erkauft werden würde. Dieses Problem stellt sich für jeden Algorithmus, der die Unterschiede zwischen den Knoten berechnen soll, er muß effizient sein, um nicht die Vorteile durch erhöhten Zeitaufwand aufzuwiegen. Wir lassen in *plane_adaptation* ohnehin nur eine kleine Planungstiefe (standardmäßig 3) von einzufügenden Schritten zu, da bei zu großen Unterschieden eher der gesamte Quellplan ungeeignet ist als nur der aktuelle problematische Schritt. Bei dieser Planungstiefe ist der Suchaufwand vertretbar, zumal nur eine Teilmenge der Methoden tatsächlich anwendbar sein wird.

Beispiel 4.6: Tabelle 4.16 zeigt eine Situation in der ein Schritt eingefügt werden muß, damit die zu übertragende Methode *Factorial-Estimate* anwendbar wird. Die Methode ist nicht auf den Knoten L_5' anwendbar, da ein zusätzlicher Faktor vor dem Bruch steht. Um die einzufügenden Schritte zu bestimmen, wird zuerst der Planungsraum *hp* kreiert, der als Konklusion die

Quellplan

Hilfsplanungsraum

1. Versuch

Quellplan abgearbeitet wurde.

4.5 Zusammenfassung

Ich habe ein System zum Übertragen von Beweisen auf Planebene für externe Analogie (TOPAL-X) vorgestellt. TOPAL-X erlaubt es, neue Probleme mit Hilfe anderer, bereits gelöster Probleme zu beweisen. Die analoge Übertragung geschieht auf Planebene, das heißt die Anwendungen der Planungsoperatoren des Quellplanes werden für den Zielplan nachgespielt. Auch die Adaptation des Quellplanes wird auf Planebene durchgeführt. Anwendungen von Planungsoperatoren werden eingefügt, ersetzt oder übersprungen. Der Benutzer kann mit Hilfe von Adaptationsregeln, die Domänenwissen enthalten, Reaktionen auf bestimmte Situationen vorgeben. TOPAL-X kann aber auch selbstständig die Planungssituation analysieren und geeignete Adaptationen durchführen. Die Analyse und die Adaptation geschehen auf der Planebene durch Benutzung des Planers.

Andere Analogiesysteme fokussieren auf dem Matchingprozess als entscheidende Heuristik für das Nachspielen der Entscheidungspunkte. Dieser Schwerpunkt wurde für TOPAL-X verlagert, da die syntaktisch orientierte Heuristik die wirklich wichtige Information, nämlich die Planungssituation, nicht genügend berücksichtigt. Dies ist erst möglich durch Einbeziehung der Anwendungbedingungen der Methoden.

Unsere Evaluation (Kapitel 7) wird exemplarisch anhand der Domäne der Grenzwertsätze zeigen, daß mit Hilfe von TOPAL-X eine große Menge von Problemen per Analogie bewiesen werden können.

Kapitel 5

Interne Analogie

> Comparisons are of great value in so far as they reduce unknown relations to known relations.
>
> *Arthur Schopenhauer*

Unter *interner Analogie im Beweisen* bezeichnet man das Übertragen von Teilbeweisen innerhalb *eines* Beweises. Wurde zum Beispiel eine Richtung des Äquivalenzsatzes über Grenzwerte von Folgen

$$\lim(x(n)) = 0 \equiv \lim |x(n)| = 0$$

bewiesen, so kann diese Teillösung benutzt werden, um die andere Richtung zu zeigen.

Der Algorithmus der internen Analogie ist dem der externen Analogie sehr ähnlich. Die Unterschiede bestehen lediglich im Retrieval und in der Bestimmung der Quellplanungsschritte. Tabelle 5.1 zeigt die Initialisierungsphase der internen Analogie und den Aufruf des Transfers. Der Transferalgorithmus entspricht dem der externen Analogie (siehe Tabelle 4.4).

Eingabe: Quellknoten T_S, Zielknoten T_T, aktueller partieller Plan *plan*.
Ausgabe: (partieller) Zielplan.

```
// Retrieval:
// Kein Quellknoten angegeben?
Wenn $T_S := \emptyset$, dann $T_S :=$ Konklusion des ersten Planungsschrittes
                                des aktuellen Plans, dessen Methode auf
                                $T_T$ anwendbar ist.
// Kein Quellknoten gefunden?
Wenn $T_T := \emptyset$, dann Ausgabe: Plan.
  Sonst:
  // Quellknoten gefunden!
  // Theoremassoziation:
  matcher:= Matche $T_S$ und $T_T$.
  Initialisiere Knotentabelle mit $T_S$ und $T_T$.
  // Transfer:
  tplan:= Teilplan für $T_S$.
  transfer(erster_schritt(tplan), plan, plan, matcher, Knotentabelle).
```

Tabelle 5.1: Initialisierungsphase und Aufruf des Transfers in der internen Analogie

5.1 Retrieval

Die interne Analogie wird mit dem aktuellen (partiellen) Beweisplan und dem Zielknoten T_T, der durch interne Analogie bewiesen werden soll, aufgerufen. Gibt der Benutzer keinen Quellknoten T_S an, dessen Lösung übertragen werden soll, so muß zuerst ein geeigneter Knoten bestimmt werden: Der aktuelle Beweisplan wird durchsucht, bis der erste Planungsschritt gefunden wird, dessen Methode auf den Zielknoten anwendbar ist. Wird kein geeigneter Planungsschritt gefunden, wird die interne Analogie beendet und der unveränderte Plan zurückgegeben. Konnte ein Quellplanungsschritt bestimmt werden, so wird die Konklusion dieses Planungsschrittes als Quellknoten gewählt. Anschließend werden Quell- und Zielknoten aufeinander gematched. Wie in der externen Analogie wird dieser initiale Match später bei Übertragen der Planungsschritte benutzt. Die Knotentabelle wird mit dem Quell- und Zielknoten initialisiert.

5.2 Transfer

Die Transferphase der internen Analogie unterscheidet sich von der der externen Analogie dadurch, daß die zu übertragenden Planungsschritte auf andere Weise bestimmt werden müssen. Es müssen die Planungsschritte übertragen werden, die für den Beweis des Quellknotens nötig waren. Daher wird nach der Bestimmung des Quellknotens der Teilplan *tplan* berechnet, der diesen Knoten beweist. *tplan* ergibt sich aus der Teilmenge der Planungsschritte des Beweises, die den Knoten selbst oder einer seiner Voraussetzungen beweisen. Während des Transfers ist der nächste Schritt nicht der Nachfolger des gerade übertragenden Schrittes, sondern das nächste Element aus *tplan*. Der Transfer endet, wenn alle Schritte aus *tplan* übertragen wurden oder kein Schritt mehr übertragen werden kann.

Beispiele für das Beweisplanen durch interne Analogie werden in Kapitel 7.2 gegeben.

Kapitel 6

Analogie als Strategie

> Wenn du dich mit mehreren Feinden auseinandersetzen mußt, dann [...] konzentriere nicht dein ganzes Machtpotential auf eine einzige Strategie. Führe im Rahmen einer Gesamtkriegslist mehrere Pläne gleichzeitig durch.
>
> *Anonym*
> *Strategeme*

In diesem Kapitel gehe ich auf die Frage ein, wann es sinnvoll ist, Analogie im Beweisplanen einzusetzen. Wie die Experimente in Kapitel 7 zeigen werden, kann analogie-gesteuertes Beweisplanen Pläne finden, die ohne spezielle Kontrollregeln nicht gefunden werden können. Gleichzeitig ist Planen durch Analogie aber deutlich aufwendiger als Planen mit diesen Kontrollregeln. Realisiert man Analogie als Strategie, ist es möglich mit Hilfe von strategischen Kontrollregeln genau die Situationen zu erkennen, in denen der Einsatz der Analogie sich lohnt. Daher beschreibe ich, nachdem ich im folgenden Abschnitt kurz auf den Aufbau einer Strategie eingehe, wie sich externe (Abschnitt 6.2) und interne Analogie (Abschnitt 6.3) als Strategie darstellen lassen. Für beide Analogiestrategien gebe ich verschiedene strategische Kontrollregeln an, die unterschiedlichen Einsatzmöglichkeiten der

Analogie entsprechen.

Durch den Einsatz von Strategien läßt sich die Analogieprozedur zudem elegant modularisieren. So ist der Teil der Analogie, der nach einzufügenden Planungsschritten sucht, ebenfalls als Strategie realisierbar. Darauf gehe ich in Abschnitt 6.4 ein. Der Nutzen der Strategien für Analogie beschränkt sich aber nicht darauf, daß die Analogie oder Teile von ihr als Strategien implementiert werden können. Da durch Strategien eine zusätzliche Abstraktionsebene in die Planung eingeführt wird, bietet es sich an, auch den Transfer auf Strategieebene durchzuführen. Dies erläutere ich in Abschnitt 6.5. Da dieser Teil meiner Diplomarbeit nicht implementiert wurde, konnte ich leider keine empirischen Untersuchungen durchführen. Ich erwarte zwar einen Performanzanstieg, da ΩMEGA eine größere Anzahl von Beweisproblemen lösen kann, ich kann diesen aber nicht durch Experimente belegen.

6.1 Aufbau einer Strategie

Konzeptuell wird in MULTI [MM00], ΩMEGAs Strategieplaner, eine Strategie als ein Verfeinerungsalgorithmus gesehen, dessen Verhalten durch Parameter spezifiziert wird. Tabelle 6.1 zeigt die Definition einer Strategie, die zum Lösen linearer Ungleichungen verwendet wird. Die Parameter die eine Strategie bestimmen, heißen *Anwendungsbedingung*, *Algorithmus* und *Parameter*. Ähnlich wie bei einer Methode gibt die *Anwendungsbedingung* an, wann eine Strategie angewendet werden kann. So kann die Strategie aus Tabelle 6.1 nur auf Ungleichungen angewendet werden. Der *Algorithmus* spezifiziert den Verfeinerungsalgorithmus, den die Strategie benutzt. Im Beispiel ist dies der Beweisplaner von ΩMEGA, *PPlanner*. Die *Parameter* geben zusätzliche Parameter an, die für den Verfeinerungsalgorithmus nötig sind, im Beispiel die verwendeten Methoden, Kontrollregeln und eine Terminierungsbedingung. In dem Beispiel aus Tabelle 6.1 soll die Strategie beendet werden, wenn der Beweisplan keine offenen linearen Ungleichungen mehr enthält.

Der Planungsalgorithmus mit mehreren Strategien ist wie folgt: Zuerst

Strategie: LöseLineareUngleichungen		
Anw.bed.	Lineare-Ungleichung	
Algorithmus	_PPlanner_	
Parameter	Methoden	_Solve, Solve*,ComplexEstimate,..._
	Kontrollregeln	_Stop-mit-Fokus,schnelle-Instantiierung,..._
	Terminierungsbedingung	Keine-Lineare-Ungleichung

Tabelle 6.1: Eine Strategie zum Lösen linearer Ungleichungen

machen die Strategien, deren Anwendungsbedingung erfüllt ist, Angebote an den *MetaReasoner*. Der *MetaReasoner* wertet dann die strategischen Kontrollregeln aus und ordnet dementsprechend die Liste der Angebote. Diejenige Strategie, die das erste Angebot der geordneten Liste gemacht hat, wird dann mit den entsprechenden Parametern aufgerufen. Eine Strategie kann ihre Ausführung unterbrechen und eine Anfrage an den MetaReasoner stellen, die von anderen Strategien bearbeitet werden kann. Ist die Anfrage erfüllt, nimmt die unterbrochene Strategie ihre Arbeit wieder auf.

Für die Realisierung und den Einsatz der Analogie als Strategie müssen folgende Fragen beantwortet werden:

- Wann kann Analogie aufgerufen werden, d.h. was sind die Anwendungsbedingungen der Analogie?
- Was sind sinnvolle Parameter?
- Wann sollte Analogie aufgerufen werden, d.h. wie muß das entsprechende strategisches Kontrollwissen formuliert sein?

Im folgenden werde ich diese Punkte für die externe und interne Analogie besprechen.

6.2 Externe Analogie als Strategie

Tabelle 6.2 zeigt die externe Analogie als Strategie. Als Verfeinerungsalgorithmus wird der Algorithmus der externen Analogie, TOPAL-X, benutzt.

Strategie: ExterneAnalogie		
Anw.bed.	Potentieller-Quellplan-vorhanden	
Algorithmus	TOPAL-X	
Parameter	Quellplan	*plan*
	Reformulierungsdomänenwissen	*ref_rules*
	Ressourcen	*ress*

Tabelle 6.2: Externe Analogie als Strategie

Die Anwendungsbedingung stellt sicher, daß ein Quellplan vorhanden ist, da ansonsten die externe Analogie keinen Transfer durchführen kann. Im Gegensatz zu anderen Strategien, wie zum Beispiel die Strategie **Löse-LineareUngleichung** aus Tabelle 6.1, die in jedem Zyklus von MULTI den Planungszustand erneut auswerten muß, braucht die Anwendungsbedingung der externen Analogie nur einmal im Laufe des Planens ausgewertet werden, da sich das Quellproblem nicht ändern wird. Diese eine Auswertung kann aber sehr teuer sein, da abhängig von der Menge der vorhandenen Pläne eine große Anzahl von Vergleichen durchgeführt werden muß.

Um den Benutzer die Möglichkeit zu geben, selbst ein Quellproblem zu spezifizieren, ist einer der Parameter der Analogiestrategie die zu benutzende Quelle. Ein weiterer möglicher Parameter ist das verwendete Reformulierungskontrollwissen. Dieses kann entweder vom Benutzer vorgeschlagen oder abhängig von der Domäne ausgewählt werden. Weiterhin ist es sinnvoll, abhängig von den zur Verfügung stehenden Ressourcen die Anzahl der möglichen Reformulierungen zu beschränken, da gerade das Einfügen von Planungsschritten eine aufwendige Reformulierung ist. Die Terminierungsbedingung über einen Parameter zu spezifizieren, ist nicht nötig. Der Analogiealgorithmus muß dann aufhören, wenn entweder die zugeteilten Ressourcen verbraucht wurden (dieses Verhalten wurde in TOPAL noch nicht berücksichtigt), wenn alle offenen Knoten im Zielbeweis geschlossen werden oder wenn der Quellplan abgearbeitet wurde. Es gibt keinen Grund, dieses Standardverhalten zu ändern und von einem Parameter abhängig zu machen.

```
(control-rule (kind strategy)
           (if (and (or (user-choice-external-analogy ?source)
                         (get-source ?source))
                      (and (or (user-choice-reformulation-knowledge ?rules)
                                (domain-reformulation-knowledge ?rules))
                            (or (user-choice-ressources ?ressources)
                                (available-ressources ?ressources)))))
           (then select(external-analogy(?source ?rules ?ressources))))
```

Tabelle 6.3: Analogie als Benutzervorschlag

Wann aber ist es sinnvoll, externe Analogie im Beweisplanen anzuwenden? In den folgenden drei Abschnitten stelle ich drei mögliche Situationen vor und gebe die dazugehörige strategische Kontrollregel an.

6.2.1 Analogie als Benutzervorschlag

In diesem Fall wählt der Benutzer die Analogie explizit als Strategie, um einen Beweisplan zu finden. Er kann sowohl das Quellproblem als auch zu verwendende Reformulierungskontrollwissen und die der Analogie zur Verfügung gestellten Ressourcen spezifizieren. Gibt der Benutzer kein Quellproblem an, so wird dies innerhalb der Theorie gesucht. Wurden keine Reformulierungskontrollregeln ausgewählt, wird das für die Domäne vorhandenen Kontrollwissen genutzt. Da die Analogiestrategie vom Benutzer gewählt wurde, werden ihr auch sämtliche Ressourcen zur Verfügung gestellt, falls der Benutzer kein andere Wahl getroffen hat. Tabelle 6.3 zeigt die dazugehörige Kontrollregel.

6.2.2 Analogie in Domänen mit ungenügend Kontrollwissen

Eine weitere sinnvolle Anwendung der Analogie besteht in den Fällen, in denen kein domänenspezifisches Kontrollwissen zur Verfügung steht. Die strategische Kontrollregel in Tabelle 6.4 überprüft daher, ob in der aktuellen Theorie Planungskontrollregeln zur Verfügung stehen. Ist dies nicht der Fall, kann die Analogiestrategie mit dem zur Verfügung stehenden Refor-

```
(control-rule (kind strategy)
              (if (and (and (current-theory ?theory)
                            (no-control-rules-available ?theory))
                       (domain-reformulation-knowledge ?rules)))
              (then select(external-analogy nil ?rules middle)))
```

Tabelle 6.4: Analogie in Domänen mit ungenügend Kontrollwissen

```
(control-rule (kind strategy)
              (if (and (other-strategies-failed)
                       (and (available-ressources ?ressources)
                            (domain-reformulation-knowledge ?rules))))
              (then select(external-analogy nil ?rules ?ressources)))
```

Tabelle 6.5: Analogie als letzter Versuch

mulierungswissen und einer beschränkten Menge an Ressourcen aufgerufen werden. Der Analogie wird nicht die komplette Menge der zur Verfügung stehenden Ressourcen zugeteilt, da die Anwendung der Analogie in diesem Fall unsicher ist. Auch ohne ausreichend Planungskontrollwissen kann unter Umständen ein Plan gefunden werden. Daher ist es sinnvoll, auf andere Strategien zurückzugreifen, wenn die Anwendung der Analogie zu teuer wird. Da der Analogie keine Quelle übergeben wird, wählt sie automatisch den Quellplan, der sich aus der Auswertung der Anwendungsbedingung ergeben hat.

6.2.3 Analogie als „letzter Versuch“

Eine weitere Möglichkeit Analogie einzusetzen, ist als „letzter Versuch“, d.h. wenn durch die anderen Strategien kein Beweisplan gefunden werden konnte. In diesem Fall ist nicht ausreichend Kontrollwissen vorhanden, so daß der Einsatz der Analogiestrategie mit den kompletten restlichen Ressourcen gerechtfertig ist. Die Auswahl des Reformulierungswissens und der Quelle geschieht wie für die Analogie in Domänen mit ungenügend Kontrollwissen. Die dazugehörige Kontrollregel zeigt Tabelle 6.5.

6.3 Interne Analogie als Strategie

Ein wesentlicher Unterschied der internen zu externen Analogie als Strategie betrifft die Anwendungsbedingung. Zwar muß in beiden Fällen eine Quelle vorhanden sein, allerdings reicht es für die interne Analogie nicht aus, einmal während des Planens nach einem Quellplan zu suchen. Sondern es müssen nach jedem Planungsschritt die neu erzeugten Knoten überprüft werden, ob sie analog zu einem anderen Teilbeweis geschlossen werden können. Diese Überprüfung kann mit Hilfe der Anwendungsbedingung der Methoden durchgeführt werden, da die Überprüfung der Anwendungsbedingungen der Methoden in der Regel nur geringe Kosten hat.

Ein weiterer Unterschied zu der externen Analogie besteht darin, daß zum Zeitpunkt der Anwendung der Strategie die Quellpläne noch nicht vollständig geführt sein müssen. Wird beispielsweise die Konjunktion $t_1 \wedge t_2$ eliminiert und lassen sich voraussichtlich t_1 und t_2 analog beweisen, so läßt sich der Transfer noch nicht durchführen, da weder t_1 noch t_2 durch einen Teilplan geschlossen sind. Erst nachdem t_1 oder t_2 geschlossen wurden, kann der dazugehörige Teilplan auf den anderen Knoten übertragen werden. Das bedeutet, die Anwendung der internen Analogiestrategie schließt einen Knoten zunächst unsicher. Ob die Anwendung der internen Strategie eine gute Wahl war, kann unter Umständen erst später festgestellt werden.

Tabelle 6.6 zeigt die Definition der internen Analogie als Strategie. Der Verfeinerungsalgorithmus ist TOPAL-I. Im Gegensatz zur externen Analogie, die auf die Wurzel des Beweises angewendet wird, muß für die interne Analogie der Zielknoten zusätzlich angegeben werden. Ansonsten entspricht die Definition der der externen Analogie.

Die strategischen Kontrollregeln für die interne Analogie zeigen Tabelle 6.7 und 6.8. Interne Analogie durch Benutzeraufruf (Tabelle 6.7) ist ähnlich zur externen Analogie definiert. Der Benutzer muß beim Aufruf einen Zielknoten übergeben. Wenn erwünscht, können auch der Quellknoten und die zu verwendenden Reformulierungskontrollregeln und Ressourcen spezifiziert werden.

Strategie: InterneAnalogie		
Anw.bed.	Potentielle-Quell-und-Zielknoten-vorhanden	
Algorithmus	TOPAL-I	
Parameter	Quelleknoten	*source*
	Zielknoten	*target*
	Reformulierungsdomänenwissen	*ref_rules*
	Ressourcen	*ress*

Tabelle 6.6: Interne Analogie als Strategie

```
(control-rule
      (kind strategy)
      (if (and (or (user-choice-internal-analogy ?node-s ?node-t)
                   (and (get-source-node ?node-s)
                        (get-target-node ?node-t)))
               (and (or (user-choice-reformulation-knowledge ?rules)
                        (domain-reformulation-knowledge ?rules))
                    (or (user-choice-ressources ?ressources)
                        (available-ressources ?ressources)))))
      (then select(internal-analogy(?node-s ?node-t ?rules ?ressources))))
```

Tabelle 6.7: Interne Analogie als Benutzervorschlag

```
(control-rule
       (kind strategy)
       (if (and (similar-node ?node-s ?node-t)
                (and (and (current-theory ?theory)
                          (no-control-rules-available ?theory))
                     (domain-reformulation-knowledge ?rules))))
       (then select(internal-analogy(?node-s ?node-t ?rules middle))))
```

Tabelle 6.8: Interne Analogie in Domänen mit ungenügend Kontrollwissen

Die automatische Anwendung der internen Analogie ist über die Kontrollregel in Tabelle 6.8 gesteuert. Kann ein neu erzeugter offener Knoten `?node-t` voraussichtlich analog zu einem anderen Knoten `?node-s` geschlossen werden und ist nicht ausreichend Kontrollwissen vorhanden, so soll der Teilplan von `?node-s` auf `?node-t` übertragen werden. Wie in der externen Analogie kann auch hier die Anzahl der zu verwendenden Ressourcen beschränkt werden.

Durch die Anwendung des Strategiekonzeptes läßt sich nicht nur die externe und interne Analogie als Strategie darstellen, sondern sie erlaubt auch die Ausgliederung eines Teils der Analogie, der Suche nach den einzufügenden Methoden.

6.4 Reformulierung als Strategie

Auf der Suche nach geeigneten Planungsschritten, die in den Zielplan eingefügt werden, um Methoden aus dem Quellplan anwendbar zu machen, wird im Algorithmus der Analogie der Planer benutzt. Dieser Aufruf des Planers innerhalb der Analogie läßt sich mittels einer Anfrage an den Meta-Reasoner als Strategie ausgliedern. Dazu stellen die Analogiestrategien eine Reformulierungsanfrage wenn Schritte eingefügt werden sollen und unterbrechen ihre Ausführung. Die Reformulierungsanfrage enthält den Knoten, an dem die Reformulierung eingefügt werden soll, die Methode die anwendbar werden soll und eine Reformulierungsgrenze, eine natürliche Zahl die die

Strategie: Reformulierung		
Anw.bed.	Reformulierungsanfrage-gestellt	
Algorithmus	*PPlanner*	
Parameter	Tiefe	max. Reformulierungstiefe *adapt_limit*
	Reformulierungsobjekt	Knoten *knoten*
	Terminierungsbedingung	*anwendbar(ziel_meth)*

Tabelle 6.9: Reformulierung als Strategie

Anzahl der maximal einzufügenden Schritte angibt. Die Reformulierungsstrategie wird dann aktiv, wenn eine Reformulierungsanfrage vorliegt und versucht diese zu erfüllen. Tabelle 6.9 zeigt die entsprechende Definition.

Die Reformulierungsstrategie ist anwendbar, wenn eine Reformulierungsanfrage von einer Analogiestrategie gestellt wurde. Sie versucht durch höchstens soviel Anwendungen von Methoden auf den Knoten *knoten* oder seine Nachfolger wie durch *adapt_limit* vorgegeben, die Zielmethode *ziel_meth* auf den aktuellen Planungszustand matchbar zu machen. Wird diese Terminierungsbedingung erreicht oder das Reformulierungslimit überschritten, gibt die Reformulierungsstrategie die Kontrolle an die Analogiestrategie zurück.

6.5 Transfer auf Strategieebene

In TOPAL findet der Transfer auf Methodenebene statt. In MULTI wird durch Strategien aber eine abstraktere Planungsebene als durch Methoden erreicht. Es ist naheliegend, diese höhere Abstraktionsstufe für die Analogie auszunutzen.

Der Algorithmus von TOPAL läßt sich folgendermaßen für den Transfer auf Strategieebene erweitern: Wird während des Transfers auf ein Quellknoten getroffen, auf den eine Strategie angewendet wurde, so wird zuerst versucht, diese Strategie auch auf den Zielknoten anzuwenden. Ist die Anwendung erfolgreich, zum Beispiel weil sie eine Normalisierungsstrategie war, für die ausreichend Kontrollwissen vorhanden ist, so wird der Transfer mit dem nächsten Knoten fortgeführt. Ist die Anwendung nicht erfolgreich, so wird in die Strategie „hinabgestiegen“, d.h. der Strategieknoten wird expandiert und die Methodenanwendungen innerhalb der Strategie werden Schritt für Schritt mit dem Algorithmus von TOPAL nachgespielt. An den problematischen Stellen, die die Anwendung der Strategie scheitern ließen, kann dann reformuliert werden.

Durch den Transfers auf Strategieebene wird es möglich, den aufwendigen Analogiealgorithmus auf die Stellen zu beschränken, für die nicht genügend Kontrollwissen vorhanden ist. Meine Erwartung ist, daß sich dadurch die Anwendungskosten der Analogie reduzieren lassen.

6.6 Zusammenfassung

In diesem Kapitel habe ich vorgestellt, wie Analogie als Strategie realisiert werden kann. Für die externe Analogie habe ich drei, für die interne Analogie zwei Situationen beschrieben, an denen eine sinnvolle Anwendung der Analogie möglich ist. Die Strategien wurden allerdings noch nicht implementiert. Trotzdem läßt sich erkennen, daß die zusätzliche Abstraktionsebene der Strategie für die Analogie Vorteile bringt: Zum einen lassen sich Teile der Analogie, im vorgestelltem Fall die Suche nach einzufügenden Reformulierungen durch den Planer, elegant modularisieren und vom restlichen Tranferalgorithmus trennen. Zum anderen wird die Anwendung der Analogie auf die Situationen beschränkt, in denen durch Analogie gesteuertes Problemlösen wirklich nötig ist.

Kapitel 7

Evaluierung

> Die Vermannichfaltigung eines jeden einzelnen Versuches ist also die eigentliche Pflicht eines Naturforschers. Er hat gerade die umgekehrte Pflicht eines Schriftstellers, der unterhalten will.
>
> *Johann Wolfgang von Goethe*
> *Der Versuch als Vermittler von Objekt und Subjekt*

In diesem Kapitel werde ich die Ergebnisse der Experimente vorstellen, die ich mit TOPAL durchgeführt habe. Die Experimente umfaßten eine Menge von mehr als 25 Beweisproblemen, die zum Großteil dem Lehrbuch für Analysis von Bartle und Sherbert [BS92] und zu einem geringen Teil dem Buch über Halbgruppen und Automaten von Deussen [Deu71] entnommen wurden.

Tabellen 7.1 bis 7.4 geben eine Übersicht über die verwendeten Sätze, Beispiele und Übungsaufgaben. Ich habe nicht nur Experimente durchgeführt, wie sie sich aus dem Aufbau der Lehrbücher ergeben, also als Quellprobleme die Beispiele, als Zielproblem die Übungsaufgaben, sondern auch Übungsaufgaben auf Beispiele übertragen. Dadurch erhielt ich eine große

Satz 3.2.4 (Seq-Geq-Zero): Ist (x_n) eine konvergente Folge und gilt für alle n $x_n \geq 0$, dann gilt auch $\lim(x_n) \geq 0$.
Satz 3.2.5 (Seq-Major): Sind (x_n) und (y_n) konvergente Folgen von reellen Zahlen und gilt $x_n \leq y_n$ für alle $n \in \mathbb{N}$, dann gilt $\lim(x_n) \leq \lim(y_n)$.
Satz 3.2.6 (Seq-Sandwich): Ist (x_n) eine konvergente Folge und gilt $a \leq x_n \leq b$ für alle $n \in \mathbb{N}$ und fixe $a, b \in \mathbb{R}$, dann gilt auch $a \leq \lim(x_n) \leq b$.
Satz 4.2.4(a) (Limit-Theorems): Seien $f, g : \mathbb{R} \to \mathbb{R}$ Funktionen und $c \in \mathbb{R}$. Außerdem sei $b \in\mathbb{R}$. Besitzen f und g die Grenzwerte $\lim\limits_{x \to c} f = l_1$ und $\lim\limits_{x \to c} g = l_2$, dann gilt: (Lim+): $\lim\limits_{x \to c}(f + g)(x) = l_1 + l_2$ (Lim-): $\lim\limits_{x \to c}(f - g)(x) = l_1 - l_2$ (Lim*): $\lim\limits_{x \to c}(f \cdot g)(x) = l_1 \cdot l_2$ (Lim-Mult): $\lim\limits_{x \to c}(b \cdot f)(x) = b \cdot l_1$
Satz 5.2.1(a) (Cont-Theorems): Seien $f, g : \mathbb{R} \to \mathbb{R}$ Funktionen. Außerdem seien $b, c \in\mathbb{R}$. Sind f und g stetig in c, dann sind auch folgende Funktionen stetig in c: (Cont+): $(f + g)$ (Cont-): $(f - g)$ (Cont*): $(f \cdot g)$ (Cont-Mult):$(b \cdot f)$

Tabelle 7.1: Verwendete Sätze aus der Grenzwertdomäne

Menge von Transferproblemen, die sowohl einfache als auch schwierige Analogieprobleme umfaßte, und konnte feststellen, welche Heuristiken in TOPAL wann Wirkung zeigen.

Mein Ziel war es, zu überprüfen, ob die „semantische“ Bestimmung der Knoten (d.h. über die Anwendungsbedingungen der Methoden) und die Berücksichtigung der planungsbedingten Abhängigkeiten (d.h. nur Quell- und Zielknoten zu vergleichen, die aus den korrespondierenden Planungsschritten erzeugt wurden) die Performanz des Beweisens durch Analogie steigert. Zu diesem Zweck wurden für jedes Analogieproblem vier Variationen der Analogieprozedur untersucht:

Beispiel 4.1.7.a (Lim-Const): Es gilt $\lim\limits_{x \to c} b = b$ für alle $c \in \mathbb{R}$.
Beispiel 4.1.7.b (Lim-Id): Es gilt $\lim\limits_{x \to c} x = c$ für alle $c \in \mathbb{R}$.
Beispiel 4.1.7.b-Variation (Lim-Id): Es gilt $\lim\limits_{x \to c} x = \frac{c}{2} + \frac{c}{2}$ für alle $c \in \mathbb{R}$.
Beispiel 4.1.7.c (Lim-Square): Es gilt $\lim\limits_{x \to c} x^2 = c^2$ für alle $c \in \mathbb{R}$.
Beispiel 4.1.7.c-Variation (Lim-Square-Var): Es gilt $\lim\limits_{x \to c} x \cdot x = c \cdot c$ für alle $c \in \mathbb{R}$.
Beispiel 4.1.7.d (Lim-Const-Div): Es gilt $\lim\limits_{x \to c} \frac{1}{x} = \frac{1}{c}$ falls $c > 0$.
Beispiel 4.1.7.e (Lim-Quotient): Es gilt $\lim\limits_{x \to 2} \frac{x^3-4}{x^2+1} = \frac{4}{5}$.

Tabelle 7.2: Verwendete Beispiele aus der Grenzwertdomäne

Übung 3.1.7 (Seq-Absval): Zeige, daß für jede Folge (x_n) gilt: Wenn $\lim(x_n) = 0$, dann gilt auch $\lim \lvert x_n \rvert = 0$.
Übung 3.1.10 (Seq-Sub): Zeige, daß $\lim(\frac{1}{n} - \frac{1}{n+1}) = 0$ ist.
Übung 4.1.8 (Lim-Cube): Zeige, daß $\lim\limits_{x \to c} x^3 = c^3$ gilt für jedes $c \in \mathbb{R}$.
Übung 4.1.10.a (Lim-Frac1): Zeige, daß gilt $\lim\limits_{x \to 2} \frac{1}{1-x} = -1$.
Übung 4.1.10.b (Lim-Frac2): Zeige, daß gilt $\lim\limits_{x \to 1} \frac{x}{1+x} = \frac{1}{2}$.
Übung 4.1.10.d (Lim-Frac3): Zeige, daß gilt $\lim\limits_{x \to 1} \frac{x^2-x+1}{x+1} = \frac{1}{2}$.
Übung 4.1.10.e (Lim-Frac4): Zeige, daß gilt $\lim\limits_{x \to 3} \frac{3}{1-x} = -\frac{3}{2}$.
Übung 4.1.10.f (Lim-Frac5): Zeige, daß gilt $\lim\limits_{x \to -4} \frac{1}{x+3} = -1$.

Tabelle 7.3: Verwendete Übungen aus der Grenzwertdomäne

Satz 4.8: Sind ρ, σ zwei symmetrische Relationen, so ist $\rho \cap \sigma$ symmetrisch.
Satz 5.3: Sind ρ, σ zwei Links- oder Rechtskongruenzen oder Kongruenzen, so ist $\rho \cap \sigma$ Links bzw. Rechtskongruenz bzw. Kongruenz.

Tabelle 7.4: Verwendete Sätze aus Halbgruppen und Automaten

- Nutzen der Anwendungsbedingungen der Methoden und der Planabhängigkeiten (S+P)[1]: Beide Heuristiken werden benutzt. Dies entspricht dem Standardvorgehen in TOPAL.
- nur Planabhängigkeiten (P): Es werden nur die planungsbedingten Abhängigkeiten genutzt. Die Auswahl der Knoten geschieht über das Matchen.
- nur Anwendungsbedingungen der Methoden (S): Es wird die Auswahl der Knoten über die Anwendungsbedingungen der Knoten genutzt, aber sämtliche Knoten des Zielplanes werden betrachtet.
- nur Matchen der Knoten (M): Die Auswahl der Knoten geschieht ausschließlich über das Matchen von Quell- und Zielknoten, keine planungsbedingten Abhängigkeiten werden verwendet.

Die Ergebnisse der analogen Übertragungen habe ich nach sechs Kriterien bewertet:

- Benötigte Zeit der Übertragung in Sekunden (Sek.),
- Anzahl der Versuche, eine Methode auf den Planungszustand zu matchen (Meth. Match.),
- Anzahl der angewandten Methoden (angw. Meth.),
- Anzahl der Knoten, die vom erweitertem Matcher gematched wurden (gem. Kn.),
- Anzahl der angewandten Reformulierungen (angw. Ref.),
- Anzahl der offengebliebenen Knoten (offene Kn.).

Abhängig von der Komplexität des Quellbeweises habe ich die Analogieprobleme in Probleme niedriger, mittlerer und hohen Schwierigkeit gegliedert. Einfache Probleme (zum Beispiel LIM-ID) können durch einen kurzen

---

[1] Den abgekürzten Begriff in Klammern werde ich in den tabellarischen Darstellungen verwenden.

Plan gelöst werden, in welchem nur wenige und einfache domänenspezifische Methoden (zum Beispiel *Tell-CS*) angewendet werden. Die Quellpläne für Probleme mittlerer Schwierigkeit (zum Beispiel LIM-FRAC 1-5) sind länger und domänenspezifische Methoden mittlerer Komplexität (zum Beispiel *Solve**) werden angewendet. Schwierige Probleme sind solche, in deren Beweisplan mindestens einmal die Methode *Complex-Estimate* angewendet wurde. Die Beweispläne sind zudem deutlich länger. Zu den schwierigen Probleme zählen u.a. LIM-SQUARE, LIM-MULT, LIM-PLUS und LIM-TIMES.

Im folgenden Abschnitt stelle ich für jeden Schwierigkeitsgrad einige Ergebnisse des analogen Transfers in den vier verschiedenen Variationen vor und erläutere die Ergebnisse. Für die Experimente wurde kein Reformulierungskontrollwissen verwendet.

7.1 Empirische Untersuchung: Externe Analogie

7.1.1 Einfache Analogieprobleme

Tabelle 7.5 und 7.6 zeigen zwei einfache Analogieprobleme: die Übertragung des Beweisplans des LIM-ID-Problems auf das LIM-ID-Problem (Tabelle 7.5) und auf eine Variation, die einen zusätzlichen Vereinfachungsschritt im Zielplan nötig macht (Tabelle 7.6).

Läßt sich der Beweisplan ohne Reformulierung übertragen, liegen die benötigten Zeiten (8 bis 16 Sekunden) nahe beieinander. Aufgrund der hohen Anzahl von Knotenmatchings (22 anstelle 1 bis 9) dauert der Transfer in Variante M am längsten.

Muß der Quellplan reformuliert werden, so zeigt der rein matchbasierte Ansatz M die ersten Schwierigkeiten: Die geeignete Reformulierung wird nicht gefunden, statt dessen werden zwei andere Reformulierungen eingefügt, die aber drei Knoten offen lassen. Zudem ist Variante M deutlich langsamer (40 Sekunden anstatt 14 bis 17).

Variante S+P ist in beiden Fällen die schnellste, also auch trotz größerer Anzahl von Methodenmatchings schneller als Variante P.

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	8	45	16	1	0	0
P	12	40	16	9	0	0
S	10	63	16	5	0	0
M	16	40	16	22	0	0

Tabelle 7.5: Quelle: Lim-Id, Ziel: Lim-Id

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	14	86	20	3	1	0
P	17	81	20	11	1	0
S	17	103	20	9	1	0
M	40	107	23	52	2	3

Tabelle 7.6: Quelle: Lim-Id, Ziel: Lim-Id-Var.

7.1.2 Analogieprobleme mittlerer Schwierigkeit

Tabelle 7.7 und 7.8 zeigen die Resultate für Analogieprobleme mittlerer Schwierigkeit. Sowohl die Unterschiede zwischen Quell- und Zielproblem sind größer, als auch die Pläne etwas komplexer als im Lim-Id-Beispiel.

Auch hier liegen die benötigten Zeiten nahe beieinander, wenn keine Reformulierung eingefügt werden muß. Werden Reformulierungen benötigt, so zeigen Variation S und M deutliche Performanzeinbußen: Sehr viel mehr Methoden werden auf Anwendbarkeit geprüft (über 270 anstelle ungefähr 90) und trotz einer größeren Anzahl von angewendeten Reformulierungen bleiben mehr Knoten offen.

Variation S+P und P konnten den Quellplan erfolgreich übertragen. Die beiden noch offenbleibenden Knoten entstehen durch das Einfügen der Methode *Simplify-Inequality* und sind die trivialen Aussagen $1 > 0$ und $5 > 0$.

Auch in diesen Beispielen benötigt die S+P Variante die geringste Zeit.

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	16	56	19	6	0	0
P	20	50	19	11	0	0
S	17	64	19	10	0	0
M	24	50	19	21	0	0

Tabelle 7.7: Quelle: LIM-FRAC1, Ziel: LIM-FRAC5

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	25	96	22	6	1	2
P	41	88	22	14	1	2
S	51	286	28	6	4	4
M	56	273	28	19	4	4

Tabelle 7.8: Quelle: LIM-FRAC1, Ziel: LIM-FRAC2

7.1.3 Schwierige Analogieprobleme

Tabelle 7.9 bis 7.14 zeigen die Ergebnisse für schwierige Beispiele, d.h. Beispiele mit langen Beweisplänen und mindestens einer Anwendung der Methode *Complex-Estimate*.

Die Tendenz, daß die S+P und P Variation eine deutlich bessere Performanz zeigen, setzt sich fort: Variante M zeigt nur noch in zwei Beispielen (Tabelle 7.9 und 7.10) annehmbare Ergebnisse, für komplizierte Beispiele wurde ein Zeitlimit von 30 Minuten überschritten; auch Variante S benötigt deutlich mehr Zeit, bis zu zehnmal so viel wie Variante S+P. Der Grund für das vermeintlich gute Ergebnis der Variante S in Tabelle 7.11 (120 Sekunden im Gegensatz zu 148 bzw. 295) liegt daran, daß der Transfer früh einen ungeeigneten Teilplan überträgt und daher ein Großteil der Planungsschritte einfach übersprungen werden.

Erstmals treten auch Fälle auf (Tabelle 7.11 und 7.14), in der Variante P erheblich schlechtere Resultate zeigt als Variante S+P: Eine größere Anzahl

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	23	69	27	8	0	0
P	32	51	27	29	0	0
S	37	161	27	33	0	0
M	60	51	27	107	0	0

Tabelle 7.9: Quelle: LIM-SQUARE, Ziel: LIM-SQUARE-VAR.

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	80	396	46	5	1	1
P	92	378	46	29	1	1
S	94	489	47	27	1	1
M	117	375	41	86	7	1

Tabelle 7.10: Quelle: LIM-SQUARE-VAR., Ziel: LIM-CUBE

von Reformulierungen werden angewandt (7 bzw. 3 anstelle von 1 bzw. 0), aber mehr Knoten bleiben offen (2 bzw. 1 anstelle von 1 bzw. 0). Die Vorteile der S+P Variante zeigen sich anscheinend umso deutlicher, je komplexer die Beweisprobleme werden.

7.2 Empirische Untersuchung: Interne Analogie

Tabelle 7.15 und 7.16 zeigen die Ergebnisse für zwei Beweisprobleme, die durch die Anwendung interner Analogie gelöst wurden.

In erstem Problem, Theorem 3.2.6 (SEQ-SANDWICH, Tabelle 7.15), wird der bereits geführte Teilbeweis der Ungleichung $a \leq x_n$ auf die noch zu beweisende Ungleichung $x_n \leq b$ übertragen. Die Ergebnisse entsprechen denen der externen Analogie: Variante S+P und P zeigen vergleichbare gute Werte (12 und 16 Sekunden Übertragungsdauer), Variante S und M sind deutlich langsamer (40 und 67 Sekunden).

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	148	626	76	12	1	1
P	295	1240	134	30	7	2
S	120	465	27	60	15	4
M	-	-	-	-	-	-

Tabelle 7.11: Quelle: LIM-MULT, Ziel: LIM-PLUS

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	94	325	73	13	5	0
P	99	292	73	38	5	0
S	358	3007	132	194	5	0
M	-	-	-	-	-	-

Tabelle 7.12: Quelle: LIM-PLUS, Ziel: LIM-MULT

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	60	154	59	15	0	0
P	68	119	59	42	0	0
S	192	2498	97	126	0	0
M	308	126	53	491	6	2

Tabelle 7.13: Quelle: LIM-PLUS, Ziel: LIM-MINUS

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	57	154	59	15	0	0
P	250	968	121	44	3	1
S	530	3072	150	149	5	1
M	-	-	-	-	-	-

Tabelle 7.14: Quelle: LIM-MINUS, Ziel: LIM-PLUS

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	12	194	44	13	0	0
P	16	182	44	24	0	0
S	40	644	44	47	0	0
M	67	242	43	237	2	0

Tabelle 7.15: Problem: SEQ-SANDWICH-ZERO

	Sek.	Meth. Match.	angw. Meth.	gem. Kn.	angw. Ref.	offene Kn.
S+P	38	213	55	5	0	0
P	39	211	55	5	0	0
S	130	575	72	65	7	1
M	99	190	43	159	8	2

Tabelle 7.16: Problem: SEQ-ABSVAL

Die etwas kompliziertere Übung 3.1.7 (SEQ-ABSVAL, Tabelle 7.16), mit einem längerem Beweisplan, bestätigt diese Tendenz. In dieser Übung wird mit Hilfe des Teilbeweises für $\lim(x_n) = 0 \to \lim|x_n| = 0$ der Beweis für $\lim|x_n| = 0 \to \lim(x_n) = 0$ geführt. Variante S und M schaffen es nicht, den Teilbeweis erfolgreich zu übertragen (1 bzw. 2 Ziele bleiben offen) und benötigen bis zu zwei Minuten, während Variante S+P und P den Transfer in unter 40 Sekunden erfolgreich durchführen.

7.3 Diskussion der empirischen Ergebnisse

Tabelle 7.17 zeigt eine grafische Übersicht über die Durchschnittswerte für die benötigte Transferzeit der verschiedenen Analogievariationen. Variation S+P, also die Bestimmung der Knoten über die Anwendungsbedingungen der Methoden und die Berücksichtigung der planungsbedingten Abhängigkeiten, zeigt in allen Fällen die beste Performanz. Je schwieriger die Probleme wurden, desto deutlicher wurde der Vorteil der Kombination dieser beiden Heuristiken: Bei weniger Aufwand, sowohl in Zeit, Methoden- und Knotenmatchings, wurden qualitativ bessere Beweispläne gefunden.

Unerwartet für mich waren die deutlich besseren Ergebnisse der Variation P gegenüber Variation S. Durch das Berücksichtigen der planungsbedingten Abhängigkeiten kann anscheinend die Menge der potentiellen Knoten sehr viel stärker und schneller eingeschränkt werden als durch die Auswahl über die Anwendungsbedingungen der Methoden. Zudem stößt Variante S auf Probleme, wenn eine Methode auf keine der möglichen Knotenkombinationen anwendbar ist, also wenn Reformulierungen angewendet werden müssen. In diesem Fall muß auf das Matchen zurückgegriffen werden und auch hier gibt es eine sehr viel größere Menge an potentiellen Knoten als für Variante P. Durch die Berücksichtigung der planungsbedingten Abhängigkeiten wird die Menge an Knoten bereits so weit eingeschränkt, daß nur selten ungeeignete Knoten gewählt werden.

Variante P schlägt fehl, wenn durch das Matchen allein keine Aussagen über die geeigneten Knoten getroffen werden kann. Dieser Fall scheint vor allem in komplexen Problemen aufzutreten. Dann schafft die zusätzliche Berücksichtigung der Anwendungsbedingungen der Methoden Abhilfe.

7.4 Vergleich mit anderen Analogiesystemen

Wie gut ist die Performanz von TOPAL im Vergleich zu anderen Analogiesystemen? Sind die verwendeten Heuristiken nötig oder können die Probleme durch andere System schneller und besser gelöst werden?

Es war leider nicht möglich, TOPAL und andere Systeme die gleiche Menge von Problemen übertragen zu lassen und die Ergebnisse zu vergleichen. Die Art und Weise, auf der die verschiedenen Systeme Probleme lösen, sind doch sehr unterschiedlich. So benutzt Kolbe und Walters Plagiator Gleichheitsbeweise, ABALONE induktive Beweise. Der Aufwand, Probleme zu konstruieren, die mit möglichst vielen Systemen lösbar sind, schien mir für eine Diplomarbeit unangemessen. So kann ich im folgenden die Analogieansätze nur „theoretisch“ vergleichen.

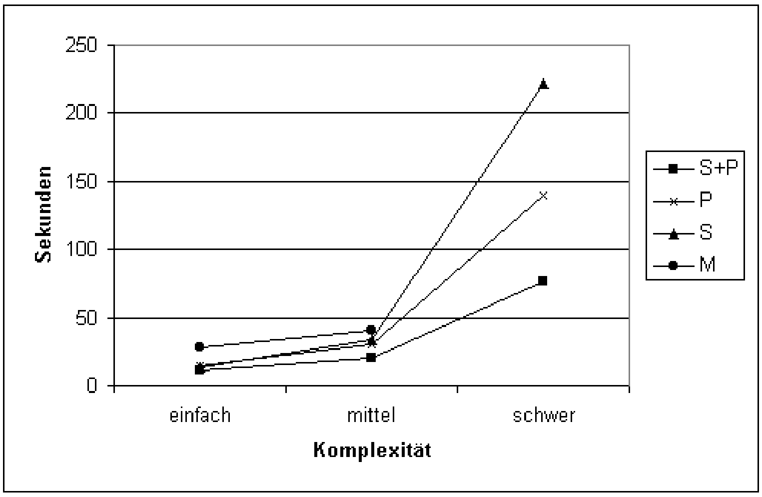


Tabelle 7.17: Performanz der verschiedenen Analogievariationen

Wirft man einen Blick auf die in Lehrbüchern verwendeten Beweise, so wird es deutlich, daß für Problemlösen durch Analogie ein Matcher höherer Ordnung nötig ist. Ansonsten könnten nur sehr einfache Probleme analog übertragen werden. Die Beschränkung auf Matchen erster Ordnung ist ein großer Nachteil der Systeme von Kling [Kli71], Munyer [Mun81], Bledsoe [Ble95] und Owen [Owe90].

Kolbe und Walters Plagiator [KW95] ist ein Analogiesystem auf der Basis eines Matchers höherer Ordnung. Ein Beweis wird dadurch „übertragen", daß der zunächst komplett variabilisierte Beweiskern schrittweise instantiiert wird. Inwieweit dieser Ansatz auf Beweiser übertragen werden kann, die keine Gleichheitsbeweise führen, ist fraglich. Kolbe und Walter machen dazu keine genaue Aussagen.

Das STRANGER-System [Sch96] verfügt über die Möglichkeit, Methoden zu reformulieren. Dadurch ist es möglich, neue Methoden zu konstruieren und Beweispläne zu finden, für die nicht genügend Methodenwissen zur Verfügung steht. TOPAL kann Probleme nur dann erfolgreich analog übertragen, wenn die Methoden allgemein genug sind. Eine Erweiterung von TOPAL um die Möglichkeit der Reformulierung von Methoden ist sinnvoll.

Allerdings wählen STRANGER und ABALONE [MW99] Reformulierungen aufgrund des Matchergebnisses aus. Wie das bereits in der Einführung erwähnte Beispiel von

$$\lim_{x \to 3} \frac{3}{1-x} = -\frac{3}{2}$$

auf

$$\lim_{x \to 2} \frac{x^3-4}{x^2+1} = \frac{4}{5}$$

zeigt, das einen komplizierten Match hat, aber durch den gleichen Plan lösbar ist, ist diese Heuristik nicht immer anwendbar. TOPAL löst dieses Problem dadurch, daß Reformulierungen in erster Linie durch den Planer und nicht durch das Matching bestimmt werden. Dies ermöglicht es, auch solche Beweisprobleme analog zu lösen, die kaum syntaktische Ähnlichkeiten besitzen. Wie die verwendeten Transferprobleme zeigen, ist die eine Anforderung, die in der Mathematik die Regel ist.

7.5 Vergleich mit dem Beweisplaner

Wie verhält durch Analogie gesteuertes Planen zum „blinden“ Planen oder zum durch Kontrollregeln gesteuertes Planen? Wiegt der zusätzliche Aufwand der Analogie die Vorteile der Einschränkung des Suchraums auf oder soll Analogie immer wenn möglich angewandt werden?

Um diese Fragen zu klären, wollte ich die Ergebnisse des Beweisens durch Analogie mit dem Planen mit und ohne Kontrollwissen vergleichen. Allerdings findet ΩMEGAs Beweisplaner ohne jegliches Kontrollwissen keinen einzigen Plan für die verwendeten Probleme. Ich habe daher eine Kontrollregel geschrieben, die die relevanten Methoden für die Grenzwertdomäne auswählt und in eine geeignete Reihenfolge bringt. Dies entspricht einer Suche mit wenig und unzureichendem Kontrollwissen. Für das Planen mit Kontrollwissen habe ich die in [Mel98a] vorgestellten Kontrollregeln verwendet, durch die ein Großteil der Grenzwertprobleme gelöst werden kann. Die Analogie benutzte die Standardvariante S+P.

Tabelle 7.18 und 7.19 zeigen die Ergebnisse des Beweisens durch Analogie im Vergleich zum Beweisen durch ΩMEGAs Planer. Für sehr einfache Probleme ist das Planen mit wenig Kontrollwissen am schnellsten. Die Auswertung von komplexen Kontrollregeln kostet Zeit und bei einfachen Beispielen scheint dieser Aufwand den größeren Suchraum beim Planen mit wenig Kontrollwissen aufzuwiegen. Planen durch Analogie ist erheblich langsamer (8 Sekunden anstelle 2 und 3), der zusätzliche Aufwand zeigt sich deutlich.

Bei Problemen mittleren Schwierigkeitsgrad zeigen sich bereits Auswirkungen des Suchraums. Im Vergleich zum Planen durch Analogie und dem Planen mit Kontrollwissen, benötigt der Planer mit schlechtem Kontrollwissen die längste Zeit (28 Sekunden anstelle 21). Analogie und Planen mit Kontrollwissen liegen gleichauf.

Bei schwierigen Problemen kann gut ausgewähltes Kontrollwissen seine Trümpfe ausspielen. Im Fall der Grenzwertdomäne ist es schnell auswertbar und schränkt den Suchraum so stark ein, daß der Zeitaufwand für komplexe Probleme nur um ein Drittel ansteigt (auf 31 Sekunden). Mit wenig Kontrollwissen kann kein Plan mehr gefunden werden. Die Analogie braucht mehr als dreimal so lang wie bei den einfachen Beispielen, 77 Sekunden. Dies liegt vor allem an der höheren Anzahl von Knotenmatchings und an der Suche nach geeigneten Reformulierungen.

Beweisen durch Analogie sollte also vor allem dann angewendet werden, wenn kein oder nur schlechtes Kontrollwissen zur Verfügung steht, da dann die Einschränkung des Suchraums den zusätzlichen Aufwand an Berechnungen, den die Analogie durchführt, lohnt.

	einfach	mittel	schwierig
Analogie	8 Sek.	21 Sek.	77 Sek.
Planen mit wenig Kontrollwissen	2 Sek.	28 Sek.	-
Planen mit viel Kontrollwissen	3 Sek.	21 Sek.	31 Sek.

Tabelle 7.18: Analogie im Vergleich zum Planer

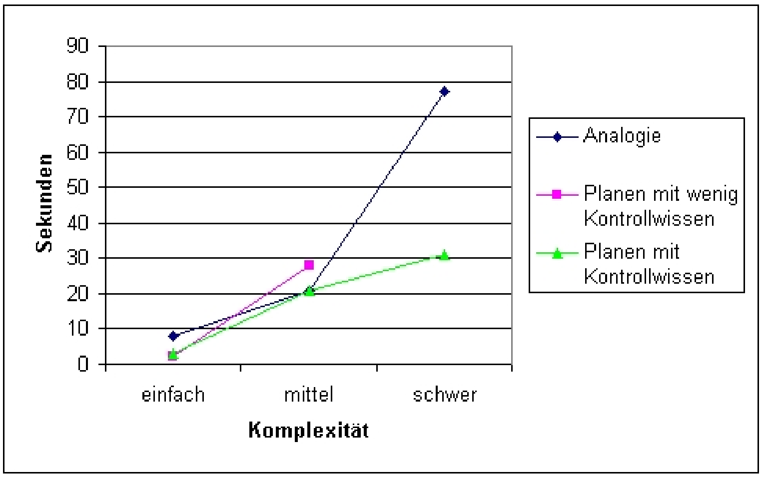


Tabelle 7.19: Grafischer Vergleich zwischen Analogie und Planer

Kapitel 8

Zusammenfassung und Ausblick

> It is proper in philosophy to consider the similar, even in things far distant from each other.
>
> *Aristoteles*

8.1 Zusammenfassung

In dieser Diplomarbeit habe ich TOPAL vorgestellt, ein Analogiesystem zum analogen Transfer von Beweisen auf der Planebene. TOPAL baut auf der analogie-gesteuerten Beweisplankonstruktion von Melis [Mel95] auf. Melis' Ansatz wurde bereits in den Analogiesystemen STRANGER [Sch96] und ABALONE [MW99] realisiert. Um den Anforderungen, wie sie innerhalb des Beweissystems ΩMEGA bei der Planung komplexer Probleme (zum Beispiel innerhalb der Grenzwertdomäne) auftreten, gerecht zu werden, mußte der Ansatz in verschiedener Hinsicht erweitert werden:

Erweiterung eines Matchalgorithmus
Im Problemlösen durch Analogie wird häufig auf Matchen zurückge-

griffen, um die Unterschiede zwischen Quell- und Zielproblem zu erkennen. Im Beweisplanen sind diese Unterschiede allerdings häufig so komplex, daß selbst ein Matcher höherer Ordnung keinen Match finden kann, obwohl die dazugehörigen Probleme durchaus analog lösbar wären. Durch meine Erweiterungen eines Matchalgorithmus höherer Ordnung ist es möglich, solche Unterschiede zu erkennen: Termabbildungen erlauben das Abbilden eines Terms auf verschiedene andere Terme; Reparaturen ermöglichen das Hinzufügen, Entfernen und Vertauschen von Teilformeln. Durch verschiedene Heuristiken wird zudem die Suche beschleunigt.

Ausnutzen der Planebene

In den meisten Analogiesystemen werden durch das Matching die notwendigen Anpassungen der Quelllösung an das Zielproblem berechnet. Für das Beweisplanen ist dieser Ansatz aber nur bedingt geeignet, da sich Probleme, die nur sehr schwer auf einander matchbar sind, durchaus durch den gleichen Plan lösen lassen. Daher werden in TOPAL der Transfer und die Auswahl von Reformulierungen in erster Linie über Informationen der Planebene gesteuert. Dies sind zum einen planungsbedingte Abhängigkeiten (d.h. welches Ziel wurde von welchem Schritt erzeugt?) und zum anderen Anwendbarkeiten der Methoden (d.h. auf welche Ziele ist die zu übertragende Methode anwendbar?) Auch die Reformulierung geschieht auf Planebene, durch Einfügen, Überspringen und Ersetzen von Planungsschritten. Dadurch ist es möglich, die benötigten Reformulierungen mit Hilfe des Planers zu bestimmen.

Analogie als Strategie

Analogie kann in ein Beweissystem als Problemlösestrategie eingebunden werden. Ich habe vorgeschlagen, wie Analogie als Strategie realisiert werden kann und strategisches Kontrollwissen für verschiedene Situationen, in denen die Anwendung der Analogie sinnvoll ist, vorgestellt. Dieser Teil der Diplomarbeit wurde allerdings nicht implementiert.

Durch eine Evaluierung konnte ich belegen, daß die Ausnutzung der Planebene die Performanz der Analogie steigert: Pläne werden schneller übertragen als in matchbasierten Ansätzen und die angewendeten Reformulierungen passen die Pläne auf geeignete Weise an. Auch bei komplexen Transferproblemen bleiben diese guten Ergebnisse erhalten.

8.2 Offene Fragen und Ausblick

Im Rahmen dieser Diplomarbeit konnte ich einige Fragen zur Analogie im Beweisplanen beantworten, aber natürlich sind viele hinzugekommen und einige offen geblieben. Im folgenden stelle ich die meiner Meinung nach interessantesten und wichtigsten offenen Punkte kurz vor.

Verwendung von Metavariablen
Die momentane Implementierung des Lemmavorschlags erlaubt nur das Einfügen komplett instantiierter Formeln. Oft ist aber die genaue Form eines Lemma noch nicht bekannt, sondern ergibt sich erst im Laufe des Beweises. Mit Hilfe von Metavariablen können Teilterme einer Formel zunächst „offen“ gelassen, d.h. mit Metavariablen instantiiert werden. Durch die Auswertung der Anwendungsbedingungen der Methoden können für diese Variablen Einschränkungen gesammelt werden, die genaue Form des Lemma im Zielproblem bestimmen.

Retrieval von Quellplänen
Eine weitere interessante Frage betrifft den Retrieval des Quellplans. Wie kann effizient aus einer möglicherweise sehr großen Menge an Quellplänen derjenige gefunden werden, der für das Lösen des Zielproblems am besten geeignet ist? Ob Indizierungsmechanismen wie aus [Vel92] für die Analogie im Beweisplanen von großem Nutzen sind, ist meiner Meinung nach eher fraglich. Eine Grundannahme des typischen fallbasierten Schließens, nämlich daß syntaktisch ähnliche Probleme ähnliche Lösungen besitzen, ist im Beweisplanen zu oft nicht erfüllt. Sinnvoller erscheint mir der Rückgriff auf mathematisches Wissen, das für TOPAL in der in Wissensbasis MBase zur Verfügung

steht. Dies wäre beispielsweise die Zugehörigkeit eines Problems zu einem bestimmten mathematischen Teilgebiet, innerhalb dessen ein Quellplan zu finden sein sollte, oder die Verwendung domänenspezifischer Prädikate.

Backtracking in der Analogie
In TOPAL ist der Transferalgorithmus „starr“, d.h. wird im Laufe des Transfers festgestellt, daß für den gerade übertragenden Teilplan der falsche Zielknoten ausgewählt wurde, kann diese Entscheidung nicht mehr rückgängig gemacht werden. Der Teilplan kann nur noch übersprungen werden. Durch Backtracking bezüglich der Knotenwahl könnten solche falschen Entscheidungen rückgängig gemacht werden.

Reformulierung von Methoden
TOPAL erlaubt nur die Reformulierungen auf Planebene. Oft wird es aber so sein, daß auch Methoden angepaßt werden müssen. Meiner Einschätzung nach werden sich bei der Reformulierung von Methoden viele Konzepte älterer, matchbasierter Ansätze der Analogie verwenden lassen, da sich das Beweisschema einer Methode auf einer kalkülnahen Ebene befindet. Wie aber Anwendungsbedingungen reformuliert werden können, ist eine offene Frage.

In dieser Arbeit habe ich eine Analogiekomponente für das Beweissystem ΩMEGA vorgestellt. Hat ΩMEGA das Glück, ein bereits gelöstes Problem zu finden, das mit seinem aktuellem Beweisproblem verwandt ist, hat es dieses Glück auch verdient, da es das alte Problem verwenden kann. So kann man sagen, daß ΩMEGA jetzt zumindest in Ansätzen Pólyas Ratschlag befolgt.

Anhang A

Ausführliches Beispiel

Dieser Abschnitt enthält die Ausgabe der Analogie während des Transfers des Planes für das Quellproblem

$$Thm_Q: \lim_{n \to \infty} x(n) = 0 \to \lim_{n \to \infty} (x(n)^2) = 0$$

auf das Zielproblem

$$Thm_Z: \lim_{n \to \infty} |x(n)| = 0.$$

Wichtige Schritte habe ich kommentiert.

```
OMEGA: prove limit-dipl-ex-t
Changing to proof plan LIMIT-DIPL-EX-T-1
OMEGA: solve-analogous-to-proof limit-dipl-ex-s-1

Starting external analogy!
Retrieving source for LIMIT-DIPL-EX-T-1:
Retrievel successfull: Source is LIMIT-DIPL-EX-S-1
Now analyzing the correspondance between the goals:
Proposing result:
Source goal <pdsn+node L2> with formula
(limseq [lam n.(power (xn n) 2)] 0)
corresponds to target goal <pdsn+node THM> with formula
(limseq [lam n.(absval (yn n))] 0) with the substitution
{(power-var --> [lam ?h179768 ?h179769.(absval ?h179768)])
```

```
(limseq-var --> limseq) (xn-var --> [lam ?h179922.(yn ?h179922)])
(0-var --> 0)},term maps NIL and repairs NIL.
```

Nach dem Retrieval wird die korrekte Korrespondenz zwischen Quell- und Zielknoten gefunden: Nicht das Quelltheorem selbst, sondern das durch die Anwendung der Methode *Implies-I* erzeugte Teilziel entspricht dem Zieltheorem.

```
------------------------
Next source-step: Method (method IMPI-M-B) on conclusion THM and
premises NIL with parameter NIL
1 step is calculated.
0 steps are matching.
The first calculated step is
 (method:  (method IMPI-M-B) goal:
           (THM MISSING) premises:  NIL
           mmatching: NIL).
Step (method:  (method IMPI-M-B) goal:  (THM MISSING)
      premises:  NIL mmatching:  NIL)
could not be transfered! Repairing!
Now checking if there is a premises missing...
... there is no premises missing.
Now checking if we can skip steps...
...there is no corresponding target goal...
... therefore skipping subproof of
<ANode THM <Justified by ImpI-m from (L2)>>.
```

Der Transfer beginnt mit dem ersten Schritt. Durch die Theoremassoziation wurde aber nicht dem Quelltheorem `THM`, sondern dem Teilziel `L2` das Zieltheorem zugeordnet. Daher gibt es für das Quelltheorem keinen korrespondierenden Zielknoten, der dazugehörige Schritt wird also übersprungen. Der folgende Planungsschritt wird ohne Probleme übertragen.

```
------------------------
Next source-step: Method (method DEFNI-M) on conclusion L2 and
premises NIL with parameter NIL
1 step is calculated.
1 step is matching.
Applying step (method:  (method DEFNI-M) goal:  (L2 THM) premises:
```

```
               NIL mmatching:  <PLAN+METH-MATCHING>):
 The (method DEFNI-M) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method DEFNE-M) on conclusion NIL and
premises (L1) with parameter NIL
1 step is calculated.
0 steps are matching.
The first calculated step is (method:  (method DEFNE-M) goal:  NIL
                              premises:  ((L1 MISSING)) mmatching:
                              NIL).
Step (method:  (method DEFNE-M) goal:  NIL premises:  ((L1 MISSING))
      mmatching:  NIL) could not be transfered! Repairing!
Now checking if there is a premises missing...
... premises missing!
Proposing node <pdsn+node |L1-PROP1|> with formula (limseq (yn) 0).
Now inserting the node in the proof plan.
```

Die Definitionselimination kann nicht übertragen werden, da die entsprechende Annahme im Zielproblem fehlt. Durch die Anwendung des Matchings auf die Quellannahme wird ein entsprechendes Lemma vorgeschlagen und eingefügt. Dann wird erneut versucht, die Definitionselimination zu übertragen, diesmal erfolgreich.

```
------------------------
Next source-step: Method (method DEFNE-M) on conclusion NIL and
premises (L1) with parameter NIL
1 step is calculated.
1 step is matching.
Applying step (method:  (method DEFNE-M) goal:  NIL premises:
               ((L1 |L1-PROP1|)) mmatching:
               <PLAN+METH-MATCHING>):
 The (method DEFNE-M) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method FORALLE-META-M-F) on conclusion NIL
and premises (L4) with parameter NIL
1 step is calculated.
1 step is matching.
```

```
Applying step (method: (method FORALLE-META-M-F) goal: NIL
premises: ((L4 L2)) mmatching: <PLAN+METH-MATCHING>):
 The (method FORALLE-META-M-F) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method EXISTSE-M-A) on conclusion L3 and
premises (L5) with parameter NIL
1 step is calculated.
1 step is matching.
Applying step (method: (method EXISTSE-M-A) goal: (L3 L1)
               premises: ((L5 L3)) mmatching:
               <PLAN+METH-MATCHING>):
 The (method EXISTSE-M-A) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method FORALLE-META-M-F) on conclusion NIL
and premises (L6) with parameter NIL
1 step is calculated.
1 step is matching.
Applying step (method: (method FORALLE-META-M-F) goal: NIL
remises: ((L6 L4)) mmatching: <PLAN+METH-MATCHING>):
 The (method FORALLE-META-M-F) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (supermethod SKOLEMIZE-S-B) on
conclusion L7 and premises NIL with parameter NIL
1 step is calculated.
1 step is matching.
Applying step (method: (supermethod SKOLEMIZE-S-B) goal: (L7 L5)
               premises: NIL mmatching: <PLAN+METH-MATCHING>):
 The (supermethod SKOLEMIZE-S-B) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (supermethod NORMAL-S-B) on conclusion L11
and premises NIL with parameter NIL
1 step is calculated.
1 step is matching.
```

```
Applying step (method:  (supermethod NORMAL-S-B) goal:  (L11 L9)
               premises:  NIL mmatching:  <PLAN+METH-MATCHING>):
 The (supermethod NORMAL-S-B) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method ANDE-M-F) on conclusion NIL and
premises (L16) with parameter NIL
2 steps are calculated.
1 step is matching.
Applying step (method:  (method ANDE-M-F) goal:  NIL premises:
               ((L16 L14)) mmatching:  <PLAN+METH-MATCHING>):
 The (method ANDE-M-F) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method TELLCS-M-B) on conclusion L14 and
premises NIL with parameter NIL
2 steps are calculated.
1 step is matching.
Applying step (method:  (method TELLCS-M-B) goal:  (L14 L12)
               premises:  NIL mmatching:  <PLAN+METH-MATCHING>):
 The (method TELLCS-M-B) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (supermethod UNWRAPHYP-S-F)
on conclusion NIL and premises (L8) with parameter NIL
1 step is calculated.
1 step is matching.
Applying step (method:  (supermethod UNWRAPHYP-S-F) goal:  NIL
               premises:  ((L8 L6)) mmatching:
               <PLAN+METH-MATCHING>):
 The (supermethod UNWRAPHYP-S-F) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method SIMPLIFY-M-F) on conclusion NIL and
premises (L29) with parameter NIL
2 steps are calculated.
1 step is matching.
```

```
Applying step (method:  (method SIMPLIFY-M-F) goal:  NIL premises:
               ((L29 L27)) mmatching:  <PLAN+METH-MATCHING>):
 The (method SIMPLIFY-M-F) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method TELLCS-M-B) on conclusion L23 and
premises NIL with parameter NIL
2 steps are calculated.
1 step is matching.
Applying step (method:  (method TELLCS-M-B) goal:  (L23 L21)
               premises:  NIL mmatching:  <PLAN+METH-MATCHING>):
 The (method TELLCS-M-B) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method ANDI-M-B) on conclusion L28 and
premises NIL with parameter NIL
1 step is calculated.
1 step is matching.
Applying step (method:  (method ANDI-M-B) goal:  (L28 L26)
               premises:  NIL mmatching:  <PLAN+METH-MATCHING>):
 The (method ANDI-M-B) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method TELLCS-M-B) on conclusion L31 and
premises NIL with parameter NIL
2 steps are calculated.
2 steps are matching.
Sorting the steps.
Applying step (method:  (method TELLCS-M-B) goal:  (L31 L30)
               premises:  NIL mmatching:  <PLAN+METH-MATCHING>):
 The (method TELLCS-M-B) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method TELLCS-M-B) on conclusion L32 and
premises NIL with parameter NIL
1 step is calculated.
1 step is matching.
```

```
Applying step (method:  (method TELLCS-M-B) goal:  (L32 L29)
               premises:  NIL mmatching:  <PLAN+METH-MATCHING>):
 The (method TELLCS-M-B) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method COMPLEXESTIMATE<-M-B)
on conclusion L17 and premises NIL with parameter (L30( ))
Searching for the corresponding node of <pdsn+schematic-node L30>
in (<pdsn+schematic-node L28>).
1 step is calculated.
0 steps are matching.
The first calculated step is (method:  (method COMPLEXESTIMATE<-M-B)
                              goal:  (L17 L15) premises:  NIL
                              mmatching:  NIL).
Step (method:  (method COMPLEXESTIMATE<-M-B) goal:  (L17 L15)
      premises:  NIL mmatching:
      NIL) could not be transfered! Repairing!
Now checking if there is a premises missing...
... there is no premises missing.
Now checking if we can skip steps...
...trying to match with the following target steps...
... a following method will be applicable...
... therefore skipping source step
<ANode L17
<Justified by COMPLEXESTIMATE<-M from (L33 L34 L35 L36)>>!
```

Die Methode *ComplexEstimate* kann im Zielplan nicht angewendet werden. Daher wird überprüft, ob und welche Reformulierung angewendet werden kann. Da keine Annahme im Zielbeweis fehlt, wird kein Lemma vorgeschlagen. Eine nachfolgende Methode ist anwendbar, die Methode *ComplexEstimate* braucht also nicht im Zielplan angewendet werden und kann übersprungen werden.

```
------------------------
Next source-step: Method (method TELLCS-M-B) on conclusion L36 and
premises NIL with parameter NIL
1 step is calculated.
0 steps are matching.
The first calculated step is (method:  (method TELLCS-M-B) goal:
```

```
                             (L36 MISSING) premises:  NIL
                             mmatching:  NIL).
Step (method:  (method TELLCS-M-B) goal:  (L36 MISSING) premises:
      NIL mmatching:  NIL) could not be transfered! Repairing!
Now checking if there is a premises missing...
... there is no premises missing.
Now checking if we can skip steps...
...there is no corresponding target goal...
... therefore skipping subproof of
<ANode L36 <Justified by TELLCS-M>>.
```

Da die Methode *ComplexEstimate* übersprungen wurde, gibt es für Teile des Quellplans keine entsprechenden offenen Knoten im Zielplan. Diese unnötigen Teilpläne können auch übersprungen werden.

```
------------------------
Next source-step: Method (method SIMPLIFY-M-B) on conclusion L34 and
premises NIL with parameter NIL
1 step is calculated.
1 step is matching.
Applying step (method:  (method SIMPLIFY-M-B) goal:  (L34 L15)
               premises:  NIL mmatching:  <PLAN+METH-MATCHING>):
 The (method SIMPLIFY-M-B) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method SOLVE*<-M-B) on conclusion L37 and
premises (L30) with parameter NIL
1 step is calculated.
1 step is matching.
Applying step (method:  (method SOLVE*<-M-B) goal:  (L37 L31)
               premises:  ((L30 L28)) mmatching:
               <PLAN+METH-MATCHING>):
 The (method SOLVE*<-M-B) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method TELLCS-M-B) on conclusion L38 and
premises NIL with parameter NIL
2 steps are calculated.
```

```
2 steps are matching.
Sorting the steps.
Applying step (method:  (method TELLCS-M-B) goal:  (L38 L32)
               premises:  NIL mmatching:  <PLAN+METH-MATCHING>):
 The (method TELLCS-M-B) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Next source-step: Method (method TELLCS-M-B) on conclusion L39 and
premises NIL with parameter NIL
1 step is calculated.
1 step is matching.
Applying step (method:  (method TELLCS-M-B) goal:  (L39 L33)
               premises:  NIL mmatching:  <PLAN+METH-MATCHING>):
 The (method TELLCS-M-B) is applied!!!!!!!!!!
Transfer of planning step successfull.
------------------------
Replay tooked 42.87 seconds.
Matched 5 nodes.
Used 71 matching attempts.
Applied 32 methods.
We applied 4 reformulations.

Replay successfull! Target LIMIT-DIPL-EX-T-1 is solved!
But we had to insert  a lemma.

OMEGA:
```

Der Transfer wurde erfolgreich abgeschlossen, durch das Einfügen eines Lemma und Überspringen einiger Quellschritte konnte das Zielproblem geplant werden.

Literaturverzeichnis

[And80] Peter B. Andrews: *Transforming Matings into Natural Deduction Proofs.* In: *Proceedings of the 5th International Conference on Automated Deduction*, S. 281–292. Springer Verlag, 1980.

[And86] Peter B. Andrews: *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Academic Press, 1986.

[Ano90] Anonym: *Strategeme.* Scherz Verlag, 1990. Aus dem Chinesischem von Harro von Senger.

[BCF$^+$97] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann und V. Sorge: ΩMEGA*: Towards a mathematical assistant.* In: William McCune (Herausgeber): *Proceedings of the 14th Conference on Automated Deduction*, LNAI, Nummer 1249 in LNAI, S. 252–255, Townsville, Australia, 1997. Springer Verlag.

[BCP86] B. Brock, S. Cooper und W. Pierce: *Some Experiments with Analogy in Proof Discovery.* Tech.Rep. AI-347-86, Microelectronics and Computer Technology Corporation, Austin, TX, 1986.

[Ble95] W.W. Bledsoe: *A Precondition Prover for Analogy.* BioSystems, 34:225–247, 1995.

[BS92] R. Bartle und D. Sherbert: *Introduction to real analysis.* J. Wiley & Sons, New York, 2 Auflage, 1992.

[Bun88] Alan Bundy: *The Use of Explicit Plans to Guide Inductive Proofs.* In: Ewing L. Lusk und Ross A. Overbeek (Herausgeber): *Proceedings of the 9th Conference on Automated Deduction*, LNCS, Nummer 310 in LNCS, S. 111–120, Argonne, Illinois, USA, 1988.

[BvHS91] A. Bundy, F. van Harmelen, J. Hesketh und A. Smaill: *Experiments with proof plans for induction.* Journal of Automated Reasoning, 7:303–324, 1991.

[Car83] J.G. Carbonell: *Learning by Analogy: Formulating and Generalizing Plans from Past Experience.* In: R.S. Michalsky, J.G. Carbonell und T.M. Mitchell (Herausgeber): *Machine Learning: An Artificial Intelligence Approach*, S. 137–162. Tioga, Palo Alto, 1983.

[Car86] J.G. Carbonell: *Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition.* In: R.S. Michalsky, J.G. Carbonell und T.M. Mitchell (Herausgeber): *Machine Learning: An Artificial Intelligence Approach*, S. 371–392. Morgan Kaufmann Publ., Los Altos, 1986.

[CGG$^+$92] Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet, Benton L. Leong, Michael B. Monagan und Stephen M. Watt: *First leaves: a tutorial introduction to Maple V.* Springer Verlag, Berlin, 1992.

[Chu40] A. Church: *A Formulation of the Simple Theory of Types.* Journal of Symbolic Logic, 5:56–68, 1940.

[Cur95] R. Curien: *Outils pour la Preuve par Analogie.* Doktorarbeit, Universite Henri Poincare - Nancy, 1995.

[Deu71] Peter Deussen: *Halbgruppen und Automaten*, Heidelberger Taschenbücher, Sammlung Informatik-99. Springer Verlag, 1971.

[Dij59] E. W. Dijkstra: *A note on two problems in connexion with graphs.* Numerische Mathematik, S. 1:269–271, 1959.

[dlTC87] T. Boy de la Tour und R. Caferra: *Proof Analogy in Interactive Theorem Proving: A Method to Express and Use it via Second Order Pattern Matching.* In: *Proceedings of the AAAI-87*, S. 95–99, 1987.

[Dow94] Gilles Dowek: *Third order matching is decidable.* Annals of pure and applied mathematics, 69:135–155, 1994.

[FFG86] B. Falkenhainer, K.D. Forbus und D. Gentner: *The Structure-Mapping Engine.* In: *Proceedings of the AAAI-86*, Philadelphia, PA, 1986.

[FN71] R.E. Fikes und N.J. Nilsson: *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.* Artificial Intelligence, 2:189–208, 1971.

[Gen35] G. Gentzen: *Untersuchungen über das logische Schließen I& II.* Math. Zeitschrift, 39:176–210,572–595, 1935.

[GMW79] M. Gordon, R. Milner und C.P. Wadsworth: *Edinburgh LCF: A Mechanized Logic of Computation,* Lecture Notes in Computer Science-78. Springer, Berlin, 1979.

[Gol81] Warren D. Goldfarb: *The Undecidability of the Second-Order Unification Problem.* Theoretical Computer Science, 13:225–230, 1981.

[Had45] J. Hadamard: *The Psychology of Invention in the Mathematical Field.* Princeton Univ. Press, Princeton, 1945.

[HBS92] Jane Hesketh, Alan Bundy und Alan Smaill: *Using Middle-Out Reasoning to Control the Synthesis of Tail-Recursive Programs.* Lecture Notes in Computer Science, 607:310–??, 1992.

[HT89] K.J. Holyoak und P. Thagard: *Analogical Mapping by Constraint Satisfaction.* Cognitive Science, 13:295–355, 1989.

[Hue73] Gérard P. Huet: *The Undecidability of Unification in Third Order Logic.* Information and Control, 22(3):257–267, 1973.

[KF00] Michael Kohlhase und Andreas Franke: *MBase: Representing Knowledge and Context for the Integration of Mathematical Software Systems.* Journal of Symbolic Comutation; Special Issue on the Integration of Computer algebra and Deduction Systems, 2000. forthcoming.

[Kli71] R.E. Kling: *A Paradigm for Reasoning by Analogy.* Artificial Intelligence, 2:147–178, 1971.

[Koe96] J. Koehler: *Planning from Second Principles.* Artificial Intelligence, 87, 1996.

[Kol83] J.L. Kolodner: *Reconstructive Memory: A Computer Model.* Cognitive Science, 7:281–328, 1983.

[KW94] Th. Kolbe und Ch. Walther: *Reusing Proofs.* In: *Proceedings of 11th European Conference on Artificial Intelligence (ECAI-94)*, Amsterdam, 1994.

[KW95] Th. Kolbe und Ch. Walther: *Second-Order Matching modulo Evaluation – A Technique for Reusing Proofs.* In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, 1995. Morgan Kaufmann.

[Lei86] G. W. Leibniz: *Projet et Essais pour arriver à quelque certitude pour finir une bonne partie des disputes et pour avancer l'art d'inventer.* In: Karel Berka und Lothar Kreisler (Herausgeber): *Logik-Texte: Kommentierte Auswahl zur Geschichte der modernen Logik*, Kapitel I.1, S. 15–17. Akademie-Verlag, Berlin, Deutschland, 1686. Deutsche Übersetzung aus G. W. Leibniz, Fragmente zur Logik, Akademie-Verlag, Berlin, 1960.

[McC90] W.W. McCune: *Otter 2.0 users guide.* Technischer Bericht ANL-90/9, Argonne National Laboratory, Maths and CS Division, Argonne, Illinois, 1990.

[Mel93] Erica Melis: *Analogy between Proofs – A Case Study.* SEKI-Report SR-93-13, Fachbereich Informatik, Unversität des Saarlandes, 1993.

[Mel94] E. Melis: *How Mathematicians Prove Theorems.* In: *Proceedings of the Annual Conference of the Cognitive Science Society*, S. 624–628, Atlanta, Georgia U.S.A., 1994.

[Mel95] E. Melis: *A Model of Analogy-Driven Proof-Plan Construction.* In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, S. 182–189, Montreal, 1995.

[Mel98a] E. Melis: *The "Limit" Domain.* In: R. Simmons, M. Veloso und S. Smith (Herausgeber): *Proceedings of the Fourth International Conference on Artificial Intelligence in Planning Systems*, S. 199–206, 1998.

[Mel98b] E. Melis: *Proof Planning with Multiple Strategies.* In: B. Gramlich und F. Pfenning (Herausgeber): *CADE-15 workshop on Strategies in Automated Deduction*, S. 57–68, 1998.

[Mel98c] Erica Melis: *The Heine-Borel Challenge Problem. In Honor of Woody Bledsoe.* Journal of Automated Reasoning, 20:255–282, 1998.

[MM00] E. Melis und A. Meier: *Proof Planning with Multiple Strategies.* In: *First International Conference on Computational Logic*, 2000. eingereicht.

[MS99] E. Melis und J.H. Siekmann: *Knowledge-Based Proof Planning.* Artificial Intelligence, 115(1):65–105, November 1999.

[MU99a] E. Melis und C. Ullrich: *Flexibly Interleaving Processes.* In: K.-D. Althoff und R. Bergmann (Herausgeber): *International Conference on Case-Based Reasoning*, Lecture Notes in Artificial Intelligence-1650, S. 263–275. Springer, 1999.

[MU99b] E. Melis und C. Ullrich: *Interleaving Processes in Case-Based Planning.* In: *7th German Workshop on Case-Based Reasoning*, S. 20–29, 1999.

[Mun81] J.C. Munyer: *Analogy as a Means of Discovery in Problem Solving and Learning.* Doktorarbeit, University of California, Santa Cruz, 1981.

[MW99] E. Melis und J. Whittle: *Analogy in Inductive Theorem Proving.* Journal of Automated Reasoning, 22(2):117–147, 1999.

[Nes94] Dan Nesmith, editor: **KEIM**-*Manual. Version 1.2*, 1994. Universität des Saarlandes, Germany.

[NSS57] A. Newell, J.C. Shaw und H.A. Simon: *Empirical Exploration with the Logic Theory Machine.* In: *Proceedings of the Western Joint Computer Conference*, Band 15, S. 218–239, 1957.

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

[Pad96] V. Padovani: *Filtrage d'order supérieur.* Thése de doctorat, Université Paris VII, 1996.

[Pol73] G. Polya: *How to Solve it.* Princeton University Press, NJ, 1973.

[Pra65] D. Prawitz: *Natural Deduction - A proof theoretical study.* Almquist and Wiksell, Stockholm, 1965.

[Rob65] J.A. Robinson: *A Machine-Oriented Logic Based on the Resolution Principle.* JACM, 12, 1965.

[Rus08] Bertrand Russell: *Mathematical Logic as based on the Theory of Types.* American Jounal of Mathematics, XXX:222–262, 1908.

[Sch96] W. Schaarschmidt: *Analogy-driven Proof Plan Construction in Omega.* Diplomarbeit, Universität des Saarlandes, Saarbrücken, Germany, 1996.

[Sny91] Wayne Snyder: *A Proof Theory for General Unification,* Progress in Computer Science and Applied Logic. Progress in Computer Science and Applied Logic. Birkhäuser, 1991.

[Vel92] M.M. Veloso: *Learning by Analogical Reasoning in General Problem Solving.* Doktorarbeit, Carnegie Mellon University, CMU, Pittsburgh, USA, 1992. CMU-CS-92-174.

[Zer08] Ernst Zermelo: *Untersuchungen über die Grundlagen der Mengenlehre. I.* Mathematische Annalen, 65:261–281, 1908.

[Zim00] Jürgen Zimmer: *Constraintlösen für Beweisplanung.* Diplomarbeit, Fachbereich Informatik, Universität des Saarlandes, Mai 2000.