Analyse und Transformation von Gleichheitsbeweisen

Stephan Schulz

Projektarbeit
in
theoretischer Informatik

Januar 1993

durchgeführt am Fachbereich Informatik
der Universität Kaiserslautern
AG Avenhaus
Betreuer: Jörg Denzinger

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

0 Einleitung

Seit den Anfängen der künstlichen Intelligenz stellt die automatische Generierung von Beweisen eines ihrer Teilgebiete dar. Inzwischen haben automatische Beweiser eine erhebliche Leistungsfähigkeit. Trotzdem werden sie kaum außerhalb der KI eingesetzt. Dafür gibt es zweifellos verschiedene Gründe. Der Mensch ist dem Computer in vielen Fällen noch deutlich überlegen. Der Computer kann seine Fähigkeiten in der Regel nur auf einem eng begrenzten Teilgebiet ausspielen, ihm fehlt die Möglichkeit, ein geeigneteres Kalkül zu wählen. Trotz heuristischer Strategien fehlt vielfach die Fähigkeit eines gezielten Vorgehens, die einem menschlichen Experten auf Grund seiner Erfahrung oft zur Verfügung steht.

Ein anderer, entscheidender Grund für die geringe Verwendung von automatisch generierten Beweisen liegt in der Präsentation des Beweises. In der Regel ist bereits das dem Beweis zu Grunde liegende Kalkül für menschliche Gewohnheiten unnatürlich. Der Computer verwendet es zudem oft in Strategien, die den Beweis in vielen kleinen, nur selten direkt zusammenhängenden Schritten finden. Für prädikatenlogische Beweiser gibt es Ansätze, Beweise so umzustrukturieren, daß sie klarer hervortreten und durch sinnvolle Lemmata gegliedert werden (man vergleiche [Li90]). Auf dem Gebiet der Gleichheitslogik sind solche Versuche bisher nur spärlich und mit geringen praktischen Auswirkungen unternommen worden, obwohl die einzelnen Beweisschritte der menschlichen Gewohnheit viel eher entsprechen als etwa die Schritte des Resolutionskalküls.

Ziel dieser Projektarbeit ist es, in Gleichheitslogik mit vervollständigungsbasierten Beweisern gefundene Beweise zu strukturieren und in eine den menschlichen Gewohnheiten entsprechende Form zu bringen.

Dazu werden zunächst die Grundlagen der Vervollständigung erläutert und ein Algorithmus für einen Gleichheitsbeweiser vorgestellt. Im zweiten Kapitel wird dann die Protokollsprache PCL für Gleichheitsbeweise vorgestellt. Die Beweisprotokolle in PCL bilden die Grundlage für die Untersuchung von Gleichheitsbeweisen.

In Kapitel 3 wird ein Algorithmus vorgestellt, der aus PCL-Protokollen von tatsächlichen Beweisen die nicht benötigten Schritte entfernt. Viele Beispiele werden erst durch diese Verkürzung beherrschbar.

Das nächste Kapitel stellt verschiedene Methoden zur Strukturierung von Beweisprotokollen durch Lemmata vor. Dabei werden sowohl lokale als auch verschiedene globale Kriterien für die Bedeutung von Zwischenergebnissen entwickelt.

Kapitel 5 stellt einen Kalkül zur Begründung von Gleichungen vor. Die dabei erzeugten Begründungen ähneln von Menschen geführten Gleichheitsbeweisen. Sie können aus PCL-Protokollen gewonnen werden, ein Algorithmus für diese Transformation von PCL-Protokollen liegt ebenfalls vor.

Im Rahmen dieser Arbeit wurden auch Programme implementiert, die die oben angesprochenen Konzepte realisieren. Sie werden in Kapitel 6 näher erläutert.

Nach einer kurzen Zusammenfassung von Ergebnissen und offenen Fragen werden in den Anhängen verschiedene Beweisbeispiele untersucht.

1 Theoretische Grundlagen

Bevor man sich mit der Darstellung von Gleichheitsbeweisen beschäftigt, ist es notwendig, zu einem gewissen Verständnis über die Art der Beweise und die damit verbundenen Probleme zu kommen. In diesem Kapitel werden einige grundlegende Mechanismen vorgestellt, mit deren Hilfe automatische Theorembeweiser Beweise in Gleichheitslogik führen können. Die Darstellung orientiert sich an [Av91] und [Av90], dort finden sich auch die Beweise zu den meisten hier zitierten Sätzen.

1.1 Allgemeine Reduktionssysteme

Ein Beweis für `s=t` zeigt, daß `s` und `t` Elemente der gleichen =-Äquivalenzklasse sind. Auch wenn man von sehr abstrakten Elementen und Relationen ausgeht, kann man bereits eine Anzahl von wichtigen Definitionen und Ergebnissen formulieren. Dieses geschieht im folgenden.

Definition: Das Wortproblem

Sei $\mathcal{E}$ eine Menge, $\sim$ eine Äquivalenzrelation auf $\mathcal{E}$. Die Frage, ob für zwei Elemente `u` und `v` aus $\mathcal{E}$ `u` $\sim$ `v` gilt, bezeichnet man als *Wortproblem* zu $(\mathcal{E},\sim)$.

Das Wortproblem ist im allgemeinen unentscheidbar, es gibt jedoch Verfahren, die es in vielen interessanten Spezialfällen lösen können. Es können hinreichende Kriterien für $\sim$ angegeben werden, die die Entscheidbarkeit des Wortproblems garantieren. So ist das Wortproblem zu $(\mathcal{E},\sim)$ entscheidbar, wenn $\sim$ von einer Relation $\Longrightarrow$ aufgespannt wird, mit der innerhalb jeder $\sim$-Äquivalenzklasse eine eindeutige Normalform berechnet werden kann.

Definition: Reduktionssystem

Ein Reduktionssystem ist ein Tupel $(\mathcal{E},\Longrightarrow)$, wobei $\mathcal{E}$ eine Menge (die *Trägermenge*) und $\Longrightarrow$ eine zweistellige Relation auf $\mathcal{E}$ (die *Reduktionsrelation*) ist.

Wir definieren zu $\Longrightarrow$ noch folgende Relationen:

$\overset{*}{\Longrightarrow}$ als reflexiven, transitiven Abschluß von $\Longrightarrow$

$\Longleftrightarrow$ als symmetrischen Abschluß von $\Longrightarrow$

$\overset{*}{\Longleftrightarrow}$ als reflexiven, symmetrischen und transitiven Abschluß von $\Longrightarrow$

Damit ist $\overset{*}{\Longleftrightarrow}$ die von $\Longrightarrow$ erzeugte Äquivalenzrelation.

Wenn $\Longrightarrow$ bestimmten Kriterien genügt, so ist das Wortproblem zu $(\mathcal{E},\overset{*}{\Longleftrightarrow})$ entscheidbar. Gilt dann $\overset{*}{\Longleftrightarrow} =\sim$, so ist damit auch das Wortproblem zu $(\mathcal{E},\sim)$ entscheidbar. Die an $\Longrightarrow$ zu stellenden Anforderungen werden jetzt näher spezifiziert.

Definition: Reduzierbarkeit, Normalform

Sei $(\mathcal{E},\Longrightarrow)$ ein Reduktionssystem.

- $\mathtt{x}$ heißt *reduzibel*, falls es ein $\mathtt{y}$ gibt mit $\mathtt{x}\Longrightarrow\mathtt{y}$. Ist $\mathtt{x}$ nicht reduzibel, so ist $\mathtt{x}$ *irreduzibel.*
- $\mathtt{y}$ heißt eine *Normalform* zu $\mathtt{x}$, falls $\mathtt{x} \overset{*}{\Longrightarrow} \mathtt{y}$ und $\mathtt{y}$ irreduzibel ist. Hat $\mathtt{x}$ genau eine Normalform, so wird diese mit $\mathtt{x}\!\downarrow$ bezeichnet.
- $\mathtt{x}$ und $\mathtt{y}$ heißen *zusammenführbar*, falls ein $\mathtt{z}$ mit $\mathtt{x}\overset{*}{\Longrightarrow}\mathtt{z}$ und $\mathtt{y}\overset{*}{\Longrightarrow}\mathtt{z}$ existiert. Die Zusammenführbarkeit von $\mathtt{x}$ und $\mathtt{y}$ wird durch $\mathtt{x}\!\downarrow\!\mathtt{y}$ beschrieben.

Definition: Noethersch, Konfluent

Eine Relation $\Longrightarrow$ auf $\mathcal{E}$ heißt

- *Noethersch (terminierend)*, falls es keine unendliche Kette $\mathtt{u}_1 \Longrightarrow \mathtt{u}_2 \Longrightarrow \mathtt{u}_3 \ldots$ gibt.
- *konfluent*, falls aus $\mathtt{u} \overset{*}{\Longrightarrow} \mathtt{x}$ und $\mathtt{u} \overset{*}{\Longrightarrow} \mathtt{y}$ $\mathtt{x}\!\downarrow\!\mathtt{y}$ folgt.
- *lokal konfluent*, falls aus $\mathtt{u} \Longrightarrow \mathtt{x}$ und $\mathtt{u} \Longrightarrow \mathtt{y}$ $\mathtt{x}\!\downarrow\!\mathtt{y}$ folgt.
- *konvergent*, falls $\Longrightarrow$ konfluent und Noethersch ist.

Ein Reduktionssystem heißt Noethersch, (lokal) konfluent oder konvergent, wenn seine Reduktionsrelation Noethersch, (lokal) konfluent oder konvergent ist.

Im allgemeinen sind weder Konfluenz noch lokale Konfluenz entscheidbar, es ist jedoch häufig einfacher, hinreichende Kriterien für die lokale Konfluenz anzugeben. Für Noethersche Reduktionssysteme sind beide Begriffe äquivalent, zur Feststellung der Konfluenz genügt also der Nachweis der lokalen Konfluenz.

Satz: Lokale Konfluenz und Konfluenz

Sei $\Longrightarrow$ eine Noethersche Relation. Es ist $\Longrightarrow$ genau dann konfluent, wenn $\Longrightarrow$ lokal konfluent ist.

Ist ein Reduktionssystem Noethersch und konfluent, so existiert in jeder $\overset{*}{\Longleftrightarrow}$ -Äquivalenzklasse ein ausgezeichnetes Element, das Normalform für alle Elemente dieser Äquivalenzklasse ist. Es gilt folgender Satz:

Satz: Entscheidbarkeit des Wortproblems

Sei $(\mathcal{E},\Longrightarrow)$ ein Reduktionssystem. Ist $\Longrightarrow$ berechenbar, konfluent und Noethersch, so ist das Wortproblem zu $(\mathcal{E},\overset{*}{\Longleftrightarrow})$ entscheidbar.

Es gibt Verfahren, mit denen man zu vielen Äquivalenzrelationen Noethersche, konfluente und berechenbare Reduktionsrelationen konstruieren kann. Ein solches Verfahren für den Spezialfall der Termersetzungssysteme bildet eine Grundlage für automatische Gleichheitsbeweiser.

1.2 Terme und Termersetzung

Ein für die Praxis besonders wichtiger Spezialfall der Reduktionssysteme sind die Termersetzungssysteme. Hier besteht die Trägermenge $\mathcal{E}$ aus einer Menge von Termen über einer Signatur. Die Reduktionsrelation wird durch eine endliche Regelmenge beschrieben. Termersetzungssysteme dienen zum Rechnen in gleichungsdefinierten Strukturen. Die notwendigen Begriffe werden im folgenden definiert.

Definition: Signatur und Terme

- Eine *Signatur* ist ein Tripel $\mathtt{sig} = (\mathtt{S},\mathtt{F}, \tau)$. Dabei ist $\mathtt{S}$ eine endliche Menge von Sorten, $\mathtt{F}$ eine endliche Menge von Funktionssymbolen und $\tau : F \to S^+$ eine Funktion, die jedem Funktionssymbol eine Stelligkeit zuordnet.
- Ein Variablensystem $(\mathtt{V_s})_{\mathtt{s}\in\mathtt{S}}$ zu einer Signatur $\mathtt{sig}$ ist ein System von abzählbar unendlichen, paarweise disjunkten Mengen. Die Elemente von $\mathtt{V_s}$ heißen *Variablen* der Sorte $\mathtt{s}$ und τ wird erweitert durch $\tau(\mathtt{x}) = \mathtt{s}$ für alle $\mathtt{x} \in \mathtt{V_s}$. Sei $\mathtt{V} = \bigcup\limits_{\mathtt{s}\in\mathtt{S}} \mathtt{V_s}$.
- Die Menge $\mathtt{Term(F)_s}$ der *Grundterme* einer Sorte $\mathtt{s}$ zu einer Signatur $\mathtt{sig}$ ist definiert wie folgt:
  1) Ist $\mathtt{f} \in \mathtt{F}$ ein Konstantensymbol der Sorte $\mathtt{s}$, also $\tau(\mathtt{f}) = \mathtt{s}$, so ist $\mathtt{f} \in \mathtt{Term(F)_s}$.
  2) Ist $\tau(\mathtt{f}) = \mathtt{s_1}, \ldots, \mathtt{s_n}, \mathtt{s}$ und sind $\mathtt{t_i} \in \mathtt{Term(F)_{s_i}}$, so ist $\mathtt{f}(\mathtt{t_1}, \ldots, \mathtt{t_n}) \in \mathtt{Term(F)_s}$.

  Es ist $\mathtt{Term(F)} = \bigcup\limits_{\mathtt{s}\in\mathtt{S}} \mathtt{Term(F)_s}$ die Menge aller Grundterme zu $\mathtt{sig}$.
- Die Menge $\mathtt{Term(F,V)}$ aller *Terme* zu $\mathtt{sig}$ und $(\mathtt{V_s})_{\mathtt{s}\in\mathtt{S}}$ ist die Menge $\mathtt{Term}(\mathtt{F} \cup \mathtt{V})$.
- Sei $\mathtt{t} \in \mathtt{Term(F,V)}$ ein Term. $\mathtt{Var(t)}$ bezeichnet die Menge der in $\mathtt{t}$ vorkommenden Variablen.

Wir wollen uns im folgenden auf den Fall beschränken, daß $\mathtt{S}$ genau eine Sorte enthält. In diesem Fall ist $\mathtt{sig}$ häufig implizit durch die in den Termen auftretenden Funktionssymbole gegeben.

Ein Term entsteht nach dieser Definition rekursiv durch Verknüpfung eines Funktionssymboles mit einer Anzahl von Teiltermen. Diese Teilterme können durch Angabe einer Stelle im Term wieder bezeichnet werden.

Definition: Stellen in einem Term

Stellen in Termen sind Folgen von natürlichen Zahlen. Es stehen λ für die leere Folge, $\mathtt{t}|p$ für den Teilterm von $\mathtt{t}$ an der Stelle p und $\mathtt{O(t)}$ für die Menge aller Stellen in t.

- Ist $\mathtt{t} \in \mathtt{Term(F,V)}$ eine Konstante oder eine Variable, so ist $\mathtt{O(t)} = \{\lambda\}$, es ist $\mathtt{t}|_\lambda \equiv \mathtt{t}$.
- Ist $\mathtt{t} \equiv \mathtt{f}(\mathtt{t_1}, \ldots, \mathtt{t_n})$, so ist $\mathtt{O(t)} = \{\mathtt{i}.p | \mathtt{1} \leq \mathtt{i} \leq \mathtt{n}, p \in \mathtt{O}(\mathtt{t_i})\} \cup \{\lambda\}$. Es ist $\mathtt{t}|_{\mathtt{i}.p} \equiv \mathtt{t_i}|p$ und $\mathtt{t}|_\lambda \equiv \mathtt{t}$.

Eine Möglichkeit zur Manipulation von Termen ist das Ersetzen von Teiltermen durch andere Terme.

Definition: Termersetzung

Der Term $\mathtt{t}[p \leftarrow \mathtt{t'}]$ ist der Term, der entsteht, wenn man den Teilterm von `t` an der Stelle p durch `t'` ersetzt.

Terme können Variablen enthalten. Die Instanziierung dieser Variablen wird durch Substitutionen beschrieben. Eine Substitution ordnet (endlich vielen) Variablen Terme zu.

Definition: Substitutionen

- Eine *Substitution* ist eine Funktion $\sigma : \mathtt{V} \rightarrow \mathtt{Term(F,V)}$ mit $\mathtt{D}(\sigma) = \{\mathtt{x}|\sigma(\mathtt{x}) \neq \mathtt{x}\}$ ist endlich.
- Ist $\mathtt{D}(\sigma) = \{\mathtt{x_1}, \ldots, \mathtt{x_n}\}$ und $\sigma(\mathtt{x_i}) = \mathtt{t_i}$, so schreibt man $\sigma = \{\mathtt{x_1} \leftarrow \mathtt{t_1}, \ldots, \mathtt{x_n} \leftarrow \mathtt{t_n}\}$.
- Eine Substitution σ wird fortgesetzt zu einer Funktion $\sigma : \mathtt{Term(F,V)} \rightarrow \mathtt{Term(F,V)}$ durch $\sigma(\mathtt{f}(\mathtt{t_1}, \ldots, \mathtt{t_n})) = \mathtt{f}(\sigma(\mathtt{t_1}), \ldots, \sigma(\mathtt{t_n}))$.
- Die spezielle Substitution σ_{id} ist definiert durch $\sigma_{id}(\mathtt{x}) = \mathtt{x}$ für alle $\mathtt{x} \in \mathtt{V}$.

Mit Hilfe von Substitutionen kann man eine Partialordnung auf Termen definieren. Dabei ist ein *allgemeinerer* Term kleiner als ein spezieller Term.

Definition: Subsumptionsordnung auf Termen

Es ist genau dann $\mathtt{s} \gtrdot \mathtt{t}$, wenn eine Substitution σ mit $\mathtt{s} \equiv \sigma(\mathtt{t})$ existiert.

Analog definiert man eine Subsumptionsordnung auf den Substitutionen selbst. Damit kann man Klassen von Substitutionen durch einen allgemeinsten Repräsentanten beschreiben.

Definition: Subsumptionsordnung auf Substitutionen

Es ist genau dann $\tau \gtrdot \rho$, wenn eine Substitution σ mit $\tau \equiv \sigma \circ \rho$ existiert. τ heißt *allgemeiner* als ρ.

Die Anwendung von Substitutionen kann zwei verschiedene Terme syntaktisch gleich machen. Dabei kann eine Substitution entweder auf einen oder auf beide Terme angewendet werden. Die Berechnung von Substitutionen mit diesen Eigenschaftem bezeichnet man als *Matching* beziehungsweise *Unifikation*.

Definition: Match und Unifikator

- Ein *Match von* `s` *auf* `t` ist eine Substitution σ mit $\sigma(\mathtt{s}) \equiv \mathtt{t}$. Eine solche Substitution wird im folgenden mit `match(s,t)` bezeichnet.
- Ein *Unifikator von* `s` *und* `t` ist eine Substitution σ mit $\sigma(\mathtt{s}) \equiv \sigma(\mathtt{t})$. Existiert eine solche Substitution zu `s` und `t`, so heißen die beiden Terme *unifizierbar*.
- Ist σ ein Unifikator von `s` und `t` und gilt für alle Unifikatoren τ von `s` und `t` $\tau \gtrdot \sigma$, so heißt σ ein *allgemeinster Unifikator*. Im folgenden wird der allgemeinste Unifikator zu `s` und `t` mit `mgu(s,t)` bezeichnet.

Es gilt folgender Satz:

Satz: Match und Unifikation

Seien $\mathtt{s}$, $\mathtt{t}\in\mathtt{Term(F,V)}$.

- Gibt es einen Match von $\mathtt{s}$ auf $\mathtt{t}$, so ist dieser Match berechenbar und eindeutig.
- Sind $\mathtt{s}$ und $\mathtt{t}$ unifizierbar, so ist der allgemeinste Unifikator von $\mathtt{s}$ und $\mathtt{t}$ berechenbar und bis auf Variablenumbenennungen eindeutig.

Mit diesen Begriffen kann man viele Äquivalenzrelationen auf der Menge der Terme durch Angabe einer endlichen Menge von Gleichungen beschreiben. Durch Angabe einer Signatur und einer Menge von Gleichungen spezifiziert man so Strukturen.

Definition: Gleichungen, Spezifikationen

- Eine Gleichung ist ein Tupel $(\mathtt{s},\mathtt{t}) \in \mathtt{Term(F,V)} \times \mathtt{Term(F,V)}$. Statt $(\mathtt{s},\mathtt{t})$ schreibt man $\mathtt{s=t}$. Ein Gleichungssystem $\mathtt{E}$ ist eine Menge von Gleichungen.
- Eine Spezifikation ist ein Paar $\mathtt{spec} = (\mathtt{sig},\mathtt{E})$, bestehend aus einer Signatur und einem Gleichungssystem.

Eine Spezifikation definiert eine Menge von Termen und eine Äquivalenzrelation, die *E-Gleichheit* auf diesen Termen.

Definition: Operationale E-Gleichheit

Sei $\mathtt{spec} = (\mathtt{sig},\mathtt{E})$ eine Signatur, $\mathtt{Term(F,V)}$ die durch $\mathtt{sig}$ definierte Menge von Termen. Es sei eine zweistellige symmetrische Relation $\vdash\!\dashv_{\mathrm{E}}$ auf $\mathtt{Term(F,V)}$ definiert durch

$\mathtt{t}_1 \vdash\!\dashv_{\mathrm{E}} \mathtt{t}_2$ gdw es existiert eine Gleichung $\mathtt{l=r} \in \mathtt{E}$, eine Stelle $p \in \mathtt{O}(\mathtt{t}_1)$ und eine Substitution σ mit $\sigma(\mathtt{l}) \equiv \mathtt{t}_1|_p$ und $\mathtt{t}_2 \equiv \mathtt{t}_1[p \leftarrow \sigma(\mathtt{r})]$.

Dann ist $=_{\mathrm{E}}$ die reflexive und transitive Hülle zu $\vdash\!\dashv_{\mathrm{E}}$, also $\mathtt{t}_1 =_{\mathrm{E}} \mathtt{t}_2$ gdw $\mathtt{t}_1 \overset{*}{\vdash\!\dashv}_{\mathrm{E}} \mathtt{t}_2$.

Ziel ist es nun, zu einer so definierten Äquivalenzrelation eine konvergente berechenbare Reduktionsrelation zu finden. Man definiert diese Reduktionsrelation auf $\mathtt{Term(F,V)}$ durch Angabe einer Regelmenge.

Definition: Regel, Regelsystem

- Eine Regel ist ein Tupel $(\mathtt{l},\mathtt{r}) \in \mathtt{Term(F,V)} \times \mathtt{Term(F,V)}$ mit $\mathtt{Var(r)} \subseteq \mathtt{Var(l)}$. Statt $(\mathtt{l},\mathtt{r})$ schreibt man $\mathtt{l} \to \mathtt{r}$. Ein Regelsystem ist eine Menge von Regeln.
- Ein Regelsystem $\mathtt{R}$ beschreibt eine Reduktionsrelation $\Longrightarrow_R$ durch
  $\mathtt{t}_1 \Longrightarrow_R \mathtt{t}_2$ gdw es existiert eine Regel $\mathtt{l} \to \mathtt{r} \in \mathtt{R}$, eine Stelle $p \in \mathtt{O}(\mathtt{t}_1)$ und eine Substitution σ mit $\sigma(\mathtt{l}) \equiv \mathtt{t}_1|_p$ und $\mathtt{t}_2 \equiv \mathtt{t}_1[p \leftarrow \sigma(\mathtt{r})]$.
  Bei bekanntem $\mathtt{R}$ schreibt man häufig $\Longrightarrow$ statt $\Longrightarrow_R$. Ein Regelsystem heißt Noethersch, (lokal) konfluent oder konvergent, wenn $\Longrightarrow_R$ Noethersch, (lokal) konfluent oder konvergent ist.

Mit dieser Definition ist $(\mathtt{Term(F,V)}, \Longrightarrow_{\mathtt{R}})$ ein Reduktionssystem. Eine weitere Ordnung auf Termen ist die Umschließungsordnung.

Definition: Umschließungsordnung

Es ist genau dann $\mathtt{s} \trianglerighteq \mathtt{t}$, wenn eine Stelle $p \in \mathtt{O(s)}$ und eine Substitution σ existieren mit $\mathtt{s}|_p \equiv \sigma(\mathtt{t})$.

Man beachte, daß $\mathtt{s} \trianglerighteq \mathtt{l}$ impliziert, daß $\mathtt{s}$ mit der Regel $\mathtt{l} \to \mathtt{r}$ reduzierbar ist.

Die so definierte Reduktionsrelation ist verträglich mit Substitutionen und der Termstruktur. Betrachtet man $\mathtt{R}$ als Gleichungssystem, so ist $\overset{*}{\Longleftrightarrow}_{\mathrm{R}} = \mathtt{=}_{\mathrm{R}}$.

Satz: Verträglichkeit von $\Longrightarrow_R$

Sei $\mathtt{R}$ ein Regelsystem.

- Ist σ eine Substitution und gilt $\mathtt{t_1} \Longrightarrow_R \mathtt{t_2}$, so gilt auch $\sigma(\mathtt{t_1}) \Longrightarrow_{\mathtt{R}} \sigma(\mathtt{t_2})$.
- Ist $p \in \mathtt{O(t)}$ und gilt $\mathtt{t_1} \Longrightarrow_R \mathtt{t_2}$, so gilt auch $\mathtt{t}[p \leftarrow \mathtt{t_1}] \Longrightarrow_{\mathtt{R}} \mathtt{t}[p \leftarrow \mathtt{t_2}]$.
- Es ist $\overset{*}{\Longleftrightarrow}_{\mathrm{R}} = \mathtt{=}_{\mathrm{R}}$ die gröbste Kongruenzrelation $\sim$ auf $\mathtt{Term(F,V)}$ mit $\sigma(\mathtt{l}) \sim \sigma(\mathtt{r})$ für alle $\mathtt{l} \to \mathtt{r} \in \mathtt{R}$ und alle Substitutionen σ.

1.3 Termination und Konfluenz bei Termersetzungssystemen

Leider sind die Termination und Konfluenz eines Regelsystemes im allgemeinen unentscheidbar. Man kann jedoch hinreichende Kriterien für die Termination angeben, für ein terminierendes $\Longrightarrow_R$ ist die Konfluenz entscheidbar.

Insbesondere ist eine Reduktionsrelation Noethersch, wenn sie Teilmenge einer Noetherschen Partialordnung ist. Es lassen sich Partialordnungen konstruieren, mit denen sich diese Eigenschaft für viele Regelsysteme nachweisen läßt.

Definition: Reduktionsordnung

- Eine Reduktionsordnung $>$ ist eine Partialordnung auf $\mathtt{Term(F,V)}$ mit den folgenden Eigenschaften:
  - $>$ ist Noethersch.
  - $>$ ist verträglich mit Substitutionen, also $\mathtt{t_1} > \mathtt{t_2} \rightsquigarrow \sigma(\mathtt{t_1}) > \sigma(\mathtt{t_2})$.
  - $>$ ist verträglich mit der Termstruktur, also $\mathtt{t_1} > \mathtt{t_2} \rightsquigarrow \mathtt{t}[p \leftarrow \mathtt{t_1}] > \mathtt{t}[p \leftarrow \mathtt{t_2}]$.
- Ein Regelsystem $\mathtt{R}$ heißt verträglich mit einer Reduktionsordnung $>$, wenn für jedes $\mathtt{l} \to \mathtt{r} \in \mathtt{R}$ gilt: $\mathtt{l} > \mathtt{r}$.

Mit Reduktionsordnungen kann man ein einfaches Kriterium für die Termination von R angeben.

Satz: Termination von Regelsystemen

Ein Regelsystem $\mathtt{R}$ ist genau dann Noethersch, wenn es mit einer Reduktionsordnung verträglich ist.

Reduktionsordnungen auf $\mathtt{Term(F, V)}$ können im allgemeinen nicht total sein. Oft interessiert man sich jedoch nur für die Reduktion von Grundtermen. Auf der Menge der Grundterme lassen sich totale Reduktionsordnungen definieren.

Definition: Grundreduktionsordnung

Eine Reduktionsordnung $>$ heißt *Grundreduktionsordnung*, wenn für alle $\mathtt{s}, \mathtt{t} \in \mathtt{Term(F)}$ (mit $\mathtt{s} =_{\mathtt{E}} \mathtt{t}$) entweder $\mathtt{s} > \mathtt{t}$ oder $\mathtt{t} > \mathtt{s}$ oder $\mathtt{s} \equiv \mathtt{t}$ gilt.

Reduktionsordnungen erlauben es auch, Noethersche Reduktionsrelationen anzugeben, die direkt auf einer Menge von Gleichungen beruhen. Dies geschieht in folgender Definition.

Definition: Richtbare Instanzen

Sei $>$ eine Reduktionsordnung, sei $\mathtt{E}$ eine Menge von Gleichungen und sei $\mathtt{R}$ eine mit $>$ verträgliche Regelmenge.

- $\mathtt{R_E} = \{\sigma(\mathtt{u}) \to \sigma(\mathtt{v}) | \mathtt{u} \doteq \mathtt{v} \in \mathtt{E}, \sigma(\mathtt{u}) > \sigma(\mathtt{v}), \sigma \text{ Substitution}\}$ ist die Menge der *richtbaren Instanzen von Gleichungen in* $\mathtt{E}$.
- Es ist $\mathtt{R(E)} = \mathtt{R} \cup \mathtt{R_E}$.

$\mathtt{R_E}$ und $\mathtt{R(E)}$ definieren wie bisher zwei (terminierende) Reduktionsrelationen $\Longrightarrow_{\mathtt{R_E}}$ und $\Longrightarrow_{\mathtt{R(E)}}$.

Für terminierende Reduktionssysteme genügt ein Test auf lokale Konfluenz, um Konfluenz festzustellen. Dazu müssen zwar im allgemeinen unendlich viele lokale Divergenzen untersucht werden, diese können jedoch bei endlichen Regelsystemen durch endlich viele *kritische Paare* überdeckt werden.

Definition: Überlappung, kritische Paare

Seien $\mathtt{l_1} \to \mathtt{r_1}$ und $\mathtt{l_2} \to \mathtt{r_2}$ zwei Regeln, die keine gemeinsamen Variablen enthalten. Sei p eine Stelle in $\mathtt{l_1}$, $\mathtt{l_1}|_{\mathtt{p}} \notin \mathtt{V}$ und $\sigma = \mathtt{mgu}(\mathtt{l_1}|p, \mathtt{l_2})$ existiere.

- $\sigma(\mathtt{l_1})$ heißt *Überlappung* von $\mathtt{l_2} \to \mathtt{r_2}$ in $\mathtt{l_1} \to \mathtt{r_1}$.
- $\langle \sigma(\mathtt{r_1}), \sigma(\mathtt{l_1}), \sigma(\mathtt{l_1})[p \leftarrow \sigma(\mathtt{r_2})\rangle$ heißt ein *Überlappungstripel* und $\langle \sigma(\mathtt{r_1}), \sigma(\mathtt{l_1})[p \leftarrow \sigma(\mathtt{r_2})]\rangle$ heißt ein *kritisches Paar* zu $\mathtt{l_2} \to \mathtt{r_2}$ und $\mathtt{l_1} \to \mathtt{r_1}$.
- Ein *kritisches Paar zu* $\mathtt{E} \cup \mathtt{R}$ ist ein kritisches Paar $\langle \mathtt{u}, \mathtt{v}\rangle$ in einem Überlappungstripel $\langle \mathtt{u}, \mathtt{w}, \mathtt{v}\rangle$ zu den Regeln $\mathtt{l_1} \to \mathtt{r_1}$ und $\mathtt{l_2} \to \mathtt{r_2}$ mit $\mathtt{l_i} \to \mathtt{r_i}$ in $\mathtt{R}$ oder $\mathtt{l_i} \doteq \mathtt{r_i}$ in $\mathtt{E}$ mit $\mathtt{u} \not> \mathtt{w}$ und $\mathtt{v} \not> \mathtt{w}$.
- $\mathtt{CP(R)}$ ist die Menge aller kritischen Paare zu Regeln aus $\mathtt{R}$, $\mathtt{CP}(\mathtt{R} \cup \mathtt{E})$ ist die Menge aller kritischen Paare zu $\mathtt{E} \cup \mathtt{R}$.

Mit dieser Definition gilt der folgende Satz.

Satz: Lokale Divergenzen und kritische Paare

Sei $\mathtt{R}$ ein Regelsystem, sei $\mathtt{E}$ eine Gleichungsmenge und sei $>$ eine Reduktionsordnung.

- Aus $\mathtt{t}_1 \Longleftarrow_{\mathtt{R}} \mathtt{t} \Longrightarrow_{\mathtt{R}} \mathtt{t}_2$ folgt $\mathtt{t}_1 \downarrow_{\mathtt{R}} \mathtt{t}_2$ oder $\mathtt{t}_1 \Longleftrightarrow_{\mathtt{CP(R)}} \mathtt{t}_2$ (Knuth-Bendix-Lemma, siehe [KB70]).
- Aus $\mathtt{t}_1 \Longleftarrow_{\mathtt{R(E)}} \mathtt{t} \Longrightarrow_{\mathtt{R(E)}} \mathtt{t}_2$ folgt $\mathtt{t}_1 \downarrow_{\mathtt{R(E)}} \mathtt{t}_2$ oder $\mathtt{t}_1 \Longleftrightarrow_{\mathtt{CP(R \cup E)}} \mathtt{t}_2$.

Man beachte, daß der zweite Teil des obigen Satzes keine direkte Folgerung aus dem ersten Teil ist, da nicht die in der Regel unendliche Menge $\mathtt{CP(R_E)}$ sondern die bei endlichem $\mathtt{R}$ und $\mathtt{E}$ ebenfalls endliche Menge $\mathtt{CP(R \cup E)}$ auftaucht.

Ein Test auf Konfluenz muß also nur die Zusammenführbarkeit aller kritischen Paare betrachten.

Satz: Konfluenz von Regelsystemen

- Ein Regelsystem $\mathtt{R}$ ist genau dann lokal konfluent, wenn für alle $(\mathtt{s},\mathtt{t}) \in \mathtt{CP(R)}$ gilt $\mathtt{t}_1 \downarrow_{\mathtt{R}} \mathtt{t}_2$.
- Ein Noethersches Regelsystem $\mathtt{R}$ ist genau dann konfluent, wenn alle kritischen Paare in $\mathtt{R}$ zusammenführbar sind.
- Sei $>$ eine Reduktionsordnung, $\mathtt{R}$ ein mit $>$ verträgliches Regelsystem und $\mathtt{E}$ eine Menge von Gleichungen. $\Longrightarrow_{\mathtt{R(E)}}$ ist konfluent, wenn alle kritischen Paare in $\mathtt{CP(R \cup E)}$ mit $\Longrightarrow_{\mathtt{R(E)}}$ zusammenführbar sind.

Sind alle kritischen Paare zusammenführbar, so ist die Reduktionsrelation konfluent. Damit haben die beiden Seiten jedes kritischen Paares die gleiche eindeutige Normalform. Also ist die Zusammenführbarkeit der kritischen Paare entscheidbar. Damit ist für endliche, Noethersche Regelsysteme entscheidbar, ob sie konfluent sind. Für durch Regel- und Gleichungssysteme definierte Reduktionsrelationen existiert ein hinreichendes Kriterium für den Test auf Konfluenz.

1.4 Vervollständigung

Mit konfluenten und Noetherschen Regelsystemen kann man auf Termen effektiv rechnen, auch die R-Gleichheit ist für ein solches System einfach zu entscheiden. Ziel ist es also, zu einer Spezifikation $(\mathtt{sig},\mathtt{E}_0)$ ein konfluentes und Noethersches Regelsystem $\mathtt{R}$ zu finden, so daß $=_{\mathtt{E}_0} \,=\, =_{\mathtt{R}}$ ist. Ist dieses nicht möglich, so kann man häufig ein Regel- und Gleichungssystem $(\mathtt{E},\mathtt{R})$ bestimmen, so daß $=_{\mathtt{E}_0} \,=\, =_{\mathtt{E \cup R}}$ gilt und $\mathtt{R(E)}$ grundkonfluent ist. Auch in diesem Fall ist die $\mathtt{E}_0$-Gleichheit entscheidbar.

Das folgende Inferenzsystem arbeitet auf Paaren $(\mathtt{E},\mathtt{R})$. Es verwendet eine Grundreduktionsordnung und erzeugt bei geeigneten Randbedingungen ein Endsystem mit den gewünschten Eigenschaften. Allerdings kann dieses Endsystem unendlich sein und erst nach unendlich vielen Inferenzschritten entstehen, häufig entsteht jedoch bereits in endlicher Zeit ein endliches System.

Definition: Das Inferenzsystem U

Sei $>$ eine Grundreduktionsordnung.

(U1) Orientieren einer Gleichung

$$\frac{(\mathtt{E} \cup \{\mathtt{s} \doteq \mathtt{t}\}, \mathtt{R})}{(\mathtt{E}, \mathtt{R} \cup \{\mathtt{s} \to \mathtt{t}\})} \quad \text{falls } \mathtt{s} > \mathtt{t}$$

(U2) Erzeugen einer neuen Gleichung

$$\frac{(\mathtt{E}, \mathtt{R})}{(\mathtt{E} \cup \{\mathtt{s}{=}\mathtt{t}\}, \mathtt{R})} \quad \text{falls } \mathtt{s} \longmapsfrom_{\mathtt{E} \cup \mathtt{R}} \mathtt{u} \longmapsto_{\mathtt{E} \cup \mathtt{R}} \mathtt{t},\ \mathtt{s} \not> \mathtt{u},\ \mathtt{t} \not> \mathtt{u}$$

(U3) Vereinfachen einer Gleichung

a) $$\frac{(\mathtt{E} \cup \{\mathtt{s} \doteq \mathtt{t}\}, \mathtt{R})}{(\mathtt{E} \cup \{\mathtt{u}{=}\mathtt{t}\}, \mathtt{R})} \quad \text{falls } \mathtt{s} \Longrightarrow_{\mathtt{R}} \mathtt{u}$$

b) $$\frac{(\mathtt{E} \cup \{\mathtt{s} \doteq \mathtt{t}\}, \mathtt{R})}{(\mathtt{E} \cup \{\mathtt{u}{=}\mathtt{t}\}, \mathtt{R})} \quad \text{falls } \mathtt{s} \Longrightarrow_{\mathtt{R}_\mathtt{E}} \mathtt{u} \text{ mit } \mathtt{l} \to \mathtt{r} \text{ und } \mathtt{s} \triangleright \mathtt{l}$$

(U4) Löschen einer Gleichung

a) $$\frac{(\mathtt{E} \cup \{\mathtt{s}{=}\mathtt{s}\}, \mathtt{R})}{(\mathtt{E}, \mathtt{R})}$$

b) $$\frac{(\mathtt{E} \cup \{\mathtt{s} \doteq \mathtt{t}, \mathtt{u} \doteq \mathtt{v}\}, \mathtt{R})}{(\mathtt{E} \cup \{\mathtt{s}{=}\mathtt{t}\}, \mathtt{R})}$$ falls es $p \in \mathtt{O}(\mathtt{u})$ und σ gibt mit $\mathtt{u}|_p \equiv \sigma(\mathtt{s})$ und $\mathtt{v} \equiv \mathtt{u}[p \leftarrow \sigma(\mathtt{t})]$ und $\mathtt{u} \triangleright \mathtt{s}$ gilt.

(S1) Vereinfachen einer rechten Seite

$$\frac{(\mathtt{E}, \mathtt{R} \cup \{\mathtt{s} \to \mathtt{t}\})}{(\mathtt{E}, \mathtt{R} \cup \{\mathtt{s} \to \mathtt{u}\})} \quad \text{falls } \mathtt{t} \Longrightarrow_{\mathtt{R(E)}} \mathtt{u}$$

(S2) Vereinfachen einer linken Seite

$$\frac{(\mathtt{E}, \mathtt{R} \cup \{\mathtt{s} \to \mathtt{t}\})}{(\mathtt{E} \cup \{\mathtt{u}{=}\mathtt{t}\}, \mathtt{R})} \quad \text{falls } \mathtt{s} \Longrightarrow_{\mathtt{R(E)}} \mathtt{t} \text{ mit } \mathtt{l} \to \mathtt{r} \text{ und } \mathtt{s} \triangleright \mathtt{l}$$

Man schreibt $(\mathtt{E}, \mathtt{R}) \vdash_{\mathtt{U}} (\mathtt{E}', \mathtt{R}')$, falls sich $(\mathtt{E}, \mathtt{R})$ in einem Schritt mit **U** in $(\mathtt{E}', \mathtt{R}')$ transformieren läßt.

U ist korrekt, es gilt der folgende Satz.

Satz: Korrektheit von U

Es gelte $(\mathtt{E}, \mathtt{R}) \vdash_{\mathtt{U}} (\mathtt{E}', \mathtt{R}')$ und $\mathtt{R}$ sei verträglich mit $>$.

- $\mathtt{R}'$ ist verträglich mit $>$.
- Es gilt $=_{\mathtt{E} \cup \mathtt{R}} \; = \; =_{\mathtt{E}' \cup \mathtt{R}'}$.

Schritte mit **U** verändern die durch $\mathtt{E}$ und $\mathtt{R}$ definierte Gleichheit nicht. Um zu einem Endsystem mit den gewünschten Eigenschaften zu kommen, muß die Anwendung der Regeln in **U** in geeigneter Weise gesteuert werden. Dabei genügt es, bestimmte Randbedingungen einzuhalten.

Definition: U-Ableitung, U-Fairness

- Eine Folge $(\mathtt{E_i},\mathtt{R_i})_{\mathtt{i}\in\mathbb{N}}$ ist eine **U**-Ableitung, falls gilt $(\mathtt{E_i},\mathtt{R_i}) \vdash_{\mathtt{U}} (\mathtt{E_{i+1}},\mathtt{R_{i+1}})$. Sie definiert das Grenzsystem $(\mathtt{E}^\infty,\mathtt{R}^\infty)$ der *persistenten Regeln und Gleichungen* mit

$$\mathtt{E}^\infty = \bigcup_{\mathtt{i}\geq 0} \bigcap_{\mathtt{j}\geq \mathtt{i}} \mathtt{E_j} \text{ und } \mathtt{R}^\infty = \bigcup_{\mathtt{i}\geq 0} \bigcap_{\mathtt{j}\geq \mathtt{i}} \mathtt{R_j}$$

- Eine **U**-Ableitung $(\mathtt{E_i},\mathtt{R_i})_{\mathtt{i}\in\mathbb{N}}$ heißt fair, falls

$$\mathtt{CP}(\mathtt{E}^\infty,\mathtt{R}^\infty) \subseteq \bigcup_{\mathtt{i}\geq 0} \mathtt{E_i}$$

Eine **U**-Ableitung ist also fair, wenn alle kritischen Paare zu persistenten Regeln und Gleichungen gebildet und aufgenommen werden. Mit Hilfe von Beweisordnungen lassen sich die folgenden Aussagen über faire **U**-Ableitungen zeigen ([BDP89]).

Satz: Vollständigkeit fairer Ableitungen

Sei $(\mathtt{E_i},\mathtt{R_i})_{\mathtt{i}\in\mathbb{N}}$ eine faire **U**-Ableitung.

- $\mathtt{R}^\infty(\mathtt{E}^\infty)$ ist Noethersch und grundkonfluent. Ist $\mathtt{E}^\infty$ leer, so ist $\mathtt{R}^\infty$ konvergent.
- Es gilt $=_{\mathtt{E}^\infty\cup\mathtt{R}^\infty} \;=\; =_{\mathtt{E_0}\cup\mathtt{R_0}}$.
- Sind $\mathtt{s},\mathtt{t} \in \mathtt{Term(F)}$ und es gilt $\mathtt{s}=_{\mathrm{E}}\mathtt{t}$, so existiert ein $\mathtt{i}$ mit $\mathtt{s} \downarrow_{\mathtt{R_i(E_i)}} \mathtt{t}$.

Es gibt Algorithmen, die zu einer beliebigen Spezifikation eine faire **U**-Ableitung erzeugen.

Definition: U-Vervollständigung

Ein **U**-Vervollständigungsverfahren ist ein Verfahren, daß zu einer Spezifikation und einer Grundreduktionsordnung eine faire **U**-Ableitung erzeugt.

Ein solches Verfahren bildet die Grundlage zu dem im nächsten Abschnitt vorgestellten Algorithmus.

1.5 Ein Semi-Entscheidungsalgorithmus für E-Gleichheit

Mit den Ergebnissen aus 1.4 läßt sich ein Algorithmus zum Finden von Gleichheitsbeweisen angeben. Dabei sind zwei verschiedene Vorgehensweisen von Interesse. Zum ersten kann man einen Beweis zu einem festen Beweisziel suchen. Mit dem letzten Satz aus 1.4 ist sichergestellt, daß man einen solchen Beweis finden kann, falls er überhaupt existiert. Zum zweiten kann man versuchen, zu einer vorgegeben Gleichungsmenge $\mathtt{E}$ das Endsystem $(\mathtt{E}^\infty,\mathtt{R}^\infty)$ zu bestimmen, mit dem man effektiv auf den Termen der Signatur rechnen kann. Leider wird dieses System für viele Spezifikationen erst nach unendlich vielen Schritten erreicht, eine Termination kann daher nicht garantiert werden.

Beide Ziele kann man erreichen, in dem man die Inferenzschritte aus **U** so anwendet, daß eine faire Ableitungskette entsteht, die Verfahren unterscheiden sich im wesentlichen nur durch ihre Terminationsbedingungen. Der Beweiser bricht ab, sobald die Normalformen der beiden

Seiten des Beweiszieles übereinstimmen, im anderen Fall bricht der Algorithmus nur ab, wenn alle kritischen Paare zur aktuellen Regel- und Gleichungsmenge gebildet sind.

Der folgende Algorithmus implementiert ein **U**-Vervollständigungsverfahren. Um die Kontrolle über die bereits gebildeten kritischen Paare zu haben, verwendet der Algorithmus nicht zwei, sondern drei Mengen von Termpaaren. Die Menge der Termpaare, zwischen denen bereits kritische Paare gebildet sind, wird darüberhinaus interreduziert gehalten, die in ihr enthaltenen Regeln und Gleichungen können sich also gegenseitig nicht mehr reduzieren. Dies gewährleistet relativ kleine Mengen von bearbeiteten Regeln und Gleichungen und somit eine effiziente Implementierung. Konkrete Implementierungen verwenden in der Regel zusätzliche Mechanismen, um Duplizierungen zu vermeiden und bessere Strategien zu ermöglichen.

Eingabe:	`M`	Eine Menge von Gleichungen
	$>$	Eine Grundreduktionsordnung
	`PROOFMODE`	`TRUE`, falls ein einzelnes Beweisziel gezeigt werden soll, `FALSE` sonst
	`(gs,gt)`	Das zu zeigende Beweisziel (nur, falls `PROOFMODE` = `TRUE`)
Variable:	`R`	Menge von Regeln, mit denen bereits kritische Paare gebildet wurden
	`E`	Menge von Gleichungen , mit denen bereits kritische Paare gebildet wurden
	`CP`	Menge von Gleichungen , mit denen noch keine kritischen Paare gebildet wurden.
	`s,t`	Die Seiten des aktuellen Termpaares
	`l,r`	Die Seiten einer neu gebildeten Regel
	`u,v`	Die Seiten eines alten, erneut untersuchten Termpaares
	`u',v'`	Eventuell reduzierte Nachfolger von `u` und `v`
Funktionen:	`NOTEMPTY(list)`	FALSE, falls `list` leer, TRUE sonst
	`FIRST(list)`	Erstes Element von `list`
	`EXCEPTFIRST(list)`	`list` ohne erstes Element
	`NORMALFORM(t,R)`	Berechnet eine Normalform von `t` bezüglich der angegebenen Regelmenge
	`SUBSUM(e,E)`	Testet, ob die Gleichung `e` von einer der Gleichungen in `E` nach Regel (U4b) überdeckt wird oder ob eine zu `e` äquivalente Gleichung bereits in `E` enthalten ist
	`INSERT(list,e)`	Fügt das Termpaar so in die Liste ein, daß vor jedem Element in `list` maximal endlich viele andere Termpaare eingefügt werden. Dies gewährleistet die Fairness.
	`CPS(e,E)`	Berechnet alle kritischen Paare zwischen `e` und Termpaaren aus `E`

```
CP := M;
E := {};
R := {};
WHILE NOTEMPTY(CP)
  IF PROOFMODE = TRUE THEN
    gs := NORMALFORM(gs,R(E));
    gt := NORMALFORM(gt,R(E));
    IF gs = gt THEN END; Beweis ist gefunden
  ENDIF
  (s,t) := FIRST(CP);
  CP := EXCEPTFIRST(CP);
  s := NORMALFORM(s,R(E));
  t := NORMALFORM(t,R(E));
  IF s≠t AND NOT(SUBSUM(s=t,E)) THEN
    FOREACH (u,v)∈ E
      u' := NORMALFORM(u,R_{s=t});
      v' := NORMALFORM(v,R_{s=t});
      IF u≠u' OR v≠v' THEN
        E := E \ {u=v};
        CP:= INSERT(CP,u'=v');
      ENDIF
    ENDFOREACH
    FOREACH (u,v)∈ R
      v := NORMALFORM(v,R(E ∪ {s = t}));
      u' := NORMALFORM(u,R_{s=t});
      IF u'≠u THEN
        R := R\{(u,v)};
        CP := INSERT(CP,u'=v);
      ENDIF
    ENDFOREACH
    FOREACH (u,v) ∈ CPS(s=t,E ∪ R)
      CP := INSERT(CP,u=v);
    IF s>t THEN
      R := R ∪ {s → t};
    ELSE IF t>s THEN
      R := R ∪ {t → s};
    ELSE
      E:= E ∪ {s=t};
    ENDIF
  ENDIF
ENDWHILE
```

(E,R) bilden das Endsystem

2 Repräsentation automatisch generierter Beweise

Ein erstes Problem bei der Bearbeitung von Beweisen liegt bereits darin, daß nur sehr wenige Beweiser ihre Beweise in rekonstruierbarer Form zur Verfügung stellen. Fast immer beschränkt sich der Beweiser auf sparsame Ausgaben, die eine exakte Verfolgung des Beweises nicht ermöglichen. Die Struktur des Beweises wird nur durch die Datenstrukturen im Computer repräsentiert.

Um einen Beweis zu bearbeiten ist also entweder ein Eingriff in die internen Abläufe des Beweisers oder eine vollständige Ausgabe des Beweislaufes in einer geeigneten Form notwendig. Die Ausgabe des Beweislaufes bietet dabei verschiedene Vorteile:

- Die Eingriffe in den Beweiser beschränken sich auf ein Minimum. Nach Einbindung geeigneter Ausgaberoutinen können Beweiser und beweisverarbeitende Programme unabhängig gewartet werden.
- Der Übergang auf einen neuen Beweiser erfolgt ohne Schwierigkeiten, eine Neuimplementierung der bearbeitenden Programme ist nicht notwendig. Die Einbindung von Ausgaberoutinen in einen neuen Beweiser erfordert nur geringen Aufwand.
- Der Beweiser wird nicht durch zusätzliche Datenstrukturen belastet, seine Mächtigkeit bleibt im wesentlichen erhalten.
- Viele Operationen lassen sich bereits auf der Ebene der Ablaufbeschreibung durchführen, diese lassen sich teilweise auf andere Kalküle übertragen.
- Aus dem Ablaufprotokol lassen sich Vorgehensweise und Probleme des Beweisers wesentlich genauer erkennen als aus den üblichen Kurzausgaben. Die verwendeten Heuristiken des Beweisers lassen sich so optimieren.

Aufgrund dieser Überlegungen entstand die Idee einer allgemeinen Beschreibungssprache für auf Vervollständigung basierende Beweise.

2.1 Die Protokollsprache PCL

Eine solche Sprache zur Beschreibung von Beweisen ist PCL (*Proof Comunication Language*). Sie erlaubt die Darstellung von Beweisen in einer konsistenten Form, die einerseits die Vorgehensweise des Beweisers transparent macht, andererseits aber auch alle Informationen für eine manuelle oder maschinelle Weiterbearbeitung des Beweises enthält.

Eine Ablaufbeschreibung in PCL besteht aus einer Reihe von Schritten, die Eingabegleichungen und Zwischenergebnisse der Vervollständigung repräsentieren. Die Verbindung zwischen den einzelnen Schritten wird durch Ausdrücke hergestellt, die die für die Herleitung verwendeten Inferenzschritte eindeutig kennzeichnen.

```
<pcl-list> ::= <pcl-step>*;;
```

Ein PCL-Schritt besteht aus einem eindeutigen Identifier, einem Typbezeichner, einem Faktum, das das Ergebnis des Schrittes angibt, und einem Ausdruck, der die Herkunft des Faktums eindeutig beschreibt.

```
<pcl-step> ::= <pcl-id> ':' <step-type> ':'
               <fact> : <pcl-initexpr>;;
```

PCL-Identifier sind Listen von ganzen Zahlen. Diese Wahl erlaubt einerseits, technische Informationen, wie etwa die Herkunft des Beweisschrittes bei verteilten Beweisern oder die verwendete Heuristik durch eine geeignete Wahl der Namensräume auszudrücken, andererseits können einzelne Schritte lokal zerlegt und und die entstehenden Teilschritte in die Beschreibung eingefügt werden, ohne eine globale Umbenennung zu erfordern.

```
<pcl-id> ::= <int>['.'<int>]*;;

<int> ::= <digit>+;;

<digit> ::= '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9';;
```

Um eine effizientere Bearbeitung der PCL-Beschreibung zu ermöglichen, müssen PCL-Identifier in aufsteigender Reihenfolge vergeben werden. Ein Schritt, der in der Beschreibung später auftaucht, muß also einen im lexikographischen Vergleich größeren Identifier haben als alle früheren Schritte.

Die Typangabe erfolgt aus zwei Gründen. Zum einen können Informationen über die interne Repräsentation des PCL-Schrittes angegeben werden, zum anderen lassen sich Informationen über die Bedeutung des Schrittes für den Beweis hier kodieren. So können zum Beispiel wichtige Teilergebnisse bei einer parallelen Vervollständigung als Lemmata ausgezeichnet und allen Prozessen zugänglich gemacht werden.

```
<step-type> ::= 'tes-rule'|'tes-eqn'|'tes-goal'|
                'tes-lemma'|'tes-final';;
```

Die verschiedenen Typen bezeichen:

`tes-rule`	Eine Regel, die im Vervollständigungsprozess entstanden ist
`tes-eqn`	Eine aus der Vervollständigung entstandene Gleichung
`tes-goal`	Eine der Hypothesen oder ein aus einer Hypothese durch Termersetzung entstandenes Ziel
`tes-lemma`	Eine durch die Vervollständigung entstandene Regel oder Gleichung, die als Lemma eingestuft wird
`tes-final`	Ein Schritt, durch den ein Beweis abgeschlossen wird

Jeder PCL-Schritt repräsentiert ein während des Beweislaufes gefundenes Faktum. Bei einem vervollständigungsbasierten Beweiser handelt es sich um Regeln und Gleichungen, die entweder aus den Axiomen folgen oder durch Reduktion von Zielen entstanden sind. Gleichungen bestehen aus zwei durch ein Gleichheitszeichen verbundenen Termen, Regeln aus zwei durch einen Pfeil verbundenen Termen.

```
<fact> ::= <rule>|<eqn>;;

<rule> ::= <term> '->' <term>;;

<eqn> ::= <term> '=' <term> ;;

<term> ::= <ident>|<ident>'(' <arglist> ')';;

<arglist> ::= <term> [',' <term>]* ;;
```

Letzter Teil eines PCL-Schrittes ist der Ausdruck, der die Herleitung des Faktums beschreibt. Dabei sind initiale Fakten, also Axiome und Beweisziele, und während des Beweislaufes gewonnenes Wissen zu unterscheiden. Initiale Fakten benötigen keine zusätzliche Rechtfertigung, später entstandene Fakten werden durch die Angabe der zu ihrer Entstehung führenden Inferenzschritte begründet.

Ein PCL-Schritt mit Begründung initial repräsentiert ein Axiom, ein Schritt mit Begründung hypothesis eine Hypothese, also ein Beweisziel. Diesen Begründungen wird als Wert das von ihnen begründete Faktum zugewiesen.

```
<pcl-initexpr> ::= 'initial'|'hypothesis'|<pcl-expr>;;

<pcl-expr> ::= <quote-expr>|<orient-expr>|<cp-expr>|
               <tes-red-expr>|<instance-expr>;;
```

Die einfachste Rechtfertigung eines neuen Faktums ist die Übernahme von bekanntem Wissen. Wert einer <quote-expr> ist das Faktum des durch einen Identifier referenzierten PCL-Schrittes. Dabei dürfen nur bereits vorhandene Schritte referenziert werden.

```
<quote-expr> ::= <pcl-id>;;
```

Während einer Vervollständigung werden Gleichungen mit einer Reduktionsordnung zu Regeln gerichtet. In PCL wird dieser Vorgang durch eine <orient-expr> beschrieben. Argumente des Orient-Ausdrucks sind eine <pcl-expr> und eine Richtungsangabe. Mögliche Richtungsangaben sind u und x. Wert einer <orient-expr> mit Richtungsangabe u ist die Regel, die durch eine Orientierung des ersten Argumentes ohne Vertauschung der Seiten entsteht, Wert einer <orient-expr> mit Richtungsangabe x ist die Regel, die bei Vertauschung der Seiten gebildet wird.

```
<orient-expr> ::= 'orient(' <pcl-expr> ',' ['u'|'x'] ')';;
```

Neue Gleichungen entstehen aus kritischen Paaren. Dabei wird eine Seite einer Regel oder Gleichung in eine Seite einer anderen Regel oder Gleichung (oder einer anderen Instanz der selben Regel oder Gleichung) überlappt. Eine <cp-expr> beschreibt diesen Vorgang durch Angabe der beider Termpaare, der Seiten und der Stellen der Überlappung. Der Wert einer <cp-expr> ist das kritische Paar, das aus der Überlappung der durch die erste Stellenangabe definierten Stelle der ersten <pcl-expr> mit der durch die zweite Stellenangabe definierten Seite der zweiten <pcl-expr> entsteht.

```
<cp-expr> ::= 'cp('<pcl-expr>','<place>','<pcl-expr>','<side>')';;
```

Eine Stellenangabe besteht dabei aus zwei Teilen, einem Seitenbezeichner (L oder R),der den linken beziehungsweise rechten Term eines Termpaares repräsentiert, und einer Liste von Zahlen, die eine Stelle in diesem Term angibt.

```
<place> ::= <side> [.<int>]*;;

<side> ::= 'L'|'R';;
```

Neue Fakten werden auch durch Reduktion vorhandener Termpaare gewonnen. Diese Operation wird durch eine `<tes-red-expr>` beschrieben. Die Argumente einer `<tes-red-expr>` sind zwei Termpaare, eine Stellen- und eine Seitenangabe. Der Wert des Ausdruckes ist das Termpaar, das entsteht, wenn man den durch die erste Stellenangabe bezeichneten Teilterm des ersten Termpaares mit der durch die zweite Stellenangabe bezeichneten Seite des zweiten Termpaares reduziert.

```
<tes-red-expr> ::= 'tes-red('<pcl-expr>','<place>','
                            <pcl-expr>','<side>')';;
```

Eine andere Methode zur Erzeugung von gültigen Gleichungen ist die Instanziierung einer vorhandenen Gleichung. Dieser Schritt tritt zwar während einer Vervollständigung nicht auf, ein auf Vervollständigung basierender Beweiser kann jedoch oft ein Beweisziel mit einer hergeleiteten Gleichung überdecken und so beweisen. Dieses wird von einer `<instance-expr>` beschrieben. Argumente einer `<instance-expr>` sind zwei Termpaare. Das erste beschreibt die überdeckte Regel oder Gleichung (in der Regel ein Beweisziel), das zweite das instanziierte Termpaar.

```
<instance-expr> ::= 'instance(' <pcl-expr> ',' <pcl-expr> ')';;
```

Oft will man neben dem Protokoll des Beweises zusätzliche Anmerkungen einfügen. PCL unterstützt zwei verschiedene Notationen für Kommentare. Zum einen können Kommentare an beliebige, also auch an leere Zeilen angehängt werden, zum anderen können Kommentare an beliebiger Stelle zwischen zwei syntaktischen Elementen in den Text eingefügt werden. Ein angehängter Kommentar wird durch ein `#` eingeleitet und durch das Zeilenende terminiert. Eingeschobene Kommentare werden durch `/*` und `*/` begrenzt.

```
<linecomment> ::= '#' <comment>;;

<insertcomment> ::= '/*' <comment> '*/';;

<comment> ::= beliebiger Text ohne Terminationszeichen;;
```

Hier ein kurzes Beispiel für ein Beweisprotokol in PCL:

```
0 : tes-eqn : f(e(),x) = x : initial
    # e() ist linksneutrales Element
1 : tes-eqn : f(x,g(x)) = e() : initial
    # g(x) ist rechtsinverses Element zu x
2 : tes-eqn : f(f(x,y),z) = f(x,f(y,z)) : initial
    # f ist assoziativ
3 : tes-goal : g(f(e(),x)) = f(e(),g(x)) : hypothesis
    # Beweisziel
4 : tes-rule : f(e(),x) -> x : orient(0,u)
    # Orientierung von 0 ohne Vertauschen der Seiten
5 : tes-goal : g(f(e(),x)) = g(x) : tes-red(3,R,4,L)
    # Reduktion der rechten Seite des Zieles mit der linken Seite von 4
```

```
6 : tes-final : g(x) = g(x) : tes-red(5,L.1,4,L)
    # Reduktion der Stelle 1 der linken Seite des Zieles mit der linken
    # Seite von Regel 4, damit ist das Ziel auf eine triviale
    # Gleichung reduziert und bewiesen.
```

2.2 Funktionen zu PCL-Protokollen

Um die Formulierung von Algorithmen zu PCL-Protokollen zu erleichtern, werden einige Hilfsfunktionen auf PCL-Schritten definiert. Eine erste Gruppe dieser Funktionen erlaubt den Zugriff auf die einzelnen Teile eines PCL-Schrittes.

Definition: Die Projektionsfunktionen ID, TYPE, FACT und EXPR, die Funktion STEP

Sei <step> := <pcl-id> : <step-type> : <fact> : <pcl-initexpr> ein PCL-Schritt.

- ID(<step>) = <pcl-id>.
- TYPE(<step>) = <step-type>.
- FACT(<step>) = <fact>.
- EXPR(<step>) = <pcl-initexpr>.

Umgekehrt ordnet die Funktion STEP einem Identifier den von ihm bezeichneten Schritt zu, also STEP(<pcl-id>) = <step>.

Als verkürzende Notation steht in der Regel nur ID(step) statt step.

Die folgenden Funktionen haben Multimengen über der Menge der PCL-Schritte als Ergebnis. Die wichtigsten Unterschiede zwischen Multimengen und einfachen Mengen bestehen darin, daß Multimengen zum einen immer endlich sind und zum anderen identische Elemente mehrfach enthalten können. Genauere Definitionen zu Multimengen finden sich in [Av91, Seite 21f].

Zusätzlich wird ein weiterer Operator auf Multimengen verwendet, der dem \-Operator (*ohne*) auf Mengen entspricht. Auch die *Mächtigkeit* einer Multimenge wird später benötigt.

Definition: Der \\-Operator und die Mächtigkeit von Multimengen

- Seien A und B Multimengen über einer Menge M. Es ist

$$(\mathtt{A}\setminus\!\!\setminus\mathtt{B})(\mathtt{m}) = \begin{cases} \mathtt{0} & \text{falls } \mathtt{B(m)} \neq \mathtt{0} \\ \mathtt{A(m)} & \text{sonst} \end{cases} \quad \text{für alle } \mathtt{m} \in \mathtt{M}$$

- Die Mächtigkeit einer Multimenge A über M ist

$$|\mathtt{A}| = \sum_{\mathtt{m}\in\mathtt{M}} \mathtt{A(m)}$$

Das Konzept der Multimengen ermöglicht eine kompakte Beschreibung der Zusammenhänge zwischen einzelnen PCL-Schritten.

Definition: Die Funktionen $\mathtt{PARENTS}$, $\mathtt{PARENTS}^*$ und $\mathtt{CHILDREN}$

Sei `step` ein PCL-Schritt.

- $\mathtt{PARENTS(step)}$ ist die Multimenge der *direkten* Vorgänger von `step`, also die Menge der in $\mathtt{EXPR(step)}$ zitierten Schritte.
- $\mathtt{PARENTS}^*\mathtt{(step)}$ ist die Mulitmenge *aller* Vorgänger von `step`, also

$$\mathtt{PARENTS}^*\mathtt{(step)} = \bigcup_{\mathtt{s} \in \mathtt{PARENTS(step)}} \mathtt{PARENTS}^*\mathtt{(s)} \cup \mathtt{PARENTS(step)}$$

- $\mathtt{CHILDREN(step)}$ ist die Multimenge aller Schritte, die `step` zitieren, also

$$\mathtt{CHILDREN(step)} = \{\mathtt{s} | \mathtt{step} \in \mathtt{PARENTS(s)}\}$$

Für die Untersuchung des Beweises ist es oft günstig, Lemmata und Axiome ähnlich zu behandeln, also die Vorfahren von Lemmata zu ignorieren. Daher werden zu `PARENTS` und $\mathtt{PARENTS}^*$ analoge Funktionen definiert, die dieses berücksichtigen.

Definition: Die Funktionen $\mathtt{LPARENTS}$, $\mathtt{LPARENTS}^*$ und $\mathtt{LCHILDREN}$

Sei `step` ein PCL-Schritt.

- Ist `step` kein Lemma, so ist $\mathtt{LPARENTS(step)}$ die Multimenge der Schritte, die in $\mathtt{EXPR(step)}$ zitiert werden, andernfalls ist $\mathtt{LPARENTS(step)}$ die leere Multimenge. Damit gilt

$$\mathtt{LPARENTS(step)} = \begin{cases} \{\} & \text{falls } \mathtt{TYPE(step)} = \mathtt{tes\text{-}lemma} \\ \mathtt{PARENTS(step)} & \text{sonst} \end{cases}$$

- $\mathtt{LPARENTS}^*\mathtt{(step)}$ ist die Multimenge der Vorgänger von `step`, die in einem Beweis zu `step` verwendet werden. Damit ist

$$\mathtt{LPARENTS}^*\mathtt{(step)} = \bigcup_{\mathtt{s} \in \mathtt{LPARENTS(step)}} \mathtt{LPARENTS}^*\mathtt{(s)} \cup \mathtt{LPARENTS(step)}$$

- $\mathtt{LCHILDREN(step)}$ ist die Multimenge aller Schritte, die `step` zitieren, wenn `step` weder Lemma noch Axiom ist, sonst die leere Multimenge. Es ist also

$$\mathtt{LCHILDREN(step)} = \begin{cases} \{\} & \text{falls } \mathtt{TYPE(step)} = \mathtt{tes\text{-}lemma} \\ \{\} & \text{falls } \mathtt{EXPR(step)} \in \{\mathtt{initial}, \mathtt{hypothesis}\} \\ \mathtt{CHILDREN(step)} & \text{sonst} \end{cases}$$

Beispiel: Man betrachte den folgenden Ausschnitt aus einem PCL-Protokoll zum Beweis des dritten Problems aus [LO82] (Das vollständige Protokoll mit einer anderen Auswahl von Lemmata befindet sich in A.1).

```
   0 : tes-eqn : j(0(),x) = x : initial
   1 : tes-eqn : j(x,0()) = x : initial
 ...
   3 : tes-eqn : j(x,g(x)) = 0() : initial
   4 : tes-eqn : j(j(x,y),z) = j(x,j(y,z)) : initial
 ...
  11 : tes-rule : j(0(),x) -> x : orient(0,u)
  12 : tes-rule : j(x,0()) -> x : orient(1,u)
  16 : tes-rule : j(x,g(x)) -> 0() : orient(3,u)
  18 : tes-rule : j(j(x,y),z) -> j(x,j(y,z)) : orient(4,u)
  28 : tes-eqn : j(x,j(g(x),y)) = j(0(),y) : cp(18,L.1,16,L)
  29 : tes-eqn : j(x,j(g(x),y)) = y : tes-red(28,R,11,L)
 ...
 102 : tes-lemma : j(x,j(g(x),y)) -> y : orient(29,u)
 108 : tes-eqn : g(g(x)) = j(x,0()) : cp(102,L.2,16,L)
 109 : tes-eqn : g(g(x)) = x : tes-red(108,R,12,L)
 ...
 124 : tes-eqn : x = j(y,j(x,g(y))) : cp(102,L.2,5,L)
 126 : tes-lemma : g(g(x)) -> x : orient(109,u)
 ...
 130 : tes-eqn : x = j(g(y),j(y,x)) : cp(102,L.2.1,126,L)
 ...
1173 : tes-eqn : f(g(x),x) = j(x,0()) : cp(102,L.2,1156,L)
 ...
1699 : tes-eqn : j(x,j(x,y)) = y : tes-red(102,L.2.1,1690,L)
 ...
```

Es ist $\texttt{PARENTS}(102) = \{29\}$ und $\texttt{PARENTS}^*(102) = \{29, 28, 18, 16, 11, 4, 3, 0\}$. Die Menge der Nachfolger von Schritt 102 ist $\texttt{CHILDREN}(102) = \{108, 124, 126, 130, 1173, 1699\}$. Da der Schritt 102 ein Lemma ist, gilt $\texttt{LCHILDREN}(\texttt{step}) = \{\}$.

Weiterhin gilt $\texttt{LPARENTS}(130) = \texttt{PARENTS}(130)$, aber es ist $\texttt{LPARENTS}^*(130) = \{102, 126\}$ und $\texttt{PARENTS}^*(130) = \{126, 109, 108, 102, 29, 28, 18, 16, 12, 11, 4, 3, 1, 0\}$. ist

3 Extraktion der Beweisstruktur

Ein Beweisprotokoll in PCL enthält in der Regel eine vollständige Beschreibung aller vom Beweiser ausgeführten Inferenzschritte. Dies ermöglicht zwar eine exakte Analyse der verwendeten Strategie, führt aber zu sehr großen und unübersichtlichen Protokollen. Für Beweise im Grenzbereich der Leistungsfähigkeit heutiger Beweiser umfaßt ein typisches Protokoll mehrere hundertausend Schritte.

Es hat sich gezeigt, daß eine Analyse, die sich nur auf die für den Beweis tatsächlich verwendeten Schritte stützt, wesentlich einfacher und ergiebiger ist. Selbst bei kurzen Beweisen schrumpft die Datenmenge auf einen Bruchteil, bei längeren Beispielen typischerweise auf weniger als ein Promille. Auch die weitere Bearbeitung und Transformation des Beweises ist so mit einem Bruchteil der Resourcen möglich.

Am Beweis beteiligt sind alle Fakten, mit denen ein Ziel reduziert oder überdeckt wird, außerdem alle Fakten, die für die Herleitung solcher Fakten gebraucht wurden. Die Extraktion dieser Schritte aus einem PCL-Protokoll ist leicht möglich.

3.1 Ein Algorithmus zur Extraktion

Folgender Algorithmus erzeugt aus einer PCL-Eingabe eine PCL-Beschreibung der geführten Beweise, in der nur noch die tatsächlich verwendeten Schritte auftauchen:

Eingabe:	`in`	Eine Liste von PCL-Schritten
Ausgabe:	`out`	Ebenso
Variable:	`used`	Eine Menge von PCL-Schritten
	`step`	Ein einzelner PCL-Schritt
Funktionen:	`NOTEMPTY(list)`	FALSE, falls `list` leer, TRUE sonst
	`LAST(list)`	Letztes Element von `list`
	`EXCEPTLAST(list)`	`list` ohne letztes Element
	`TYPE(step)`	Typ von `step` (siehe 2.2)
	`PARENTS(step)`	Menge der direkten Vorgänger von `step` (siehe 2.2)
	`CONS(step,list)`	Liste, die durch Einfügen von `step` als erstes Element in `list` entsteht

```
used := {};
WHILE NOTEMPTY(in)
  step := LAST(in);
  in := EXCEPTLAST(in);
  IF TYPE(step) = tes-final OR step ∈ used THEN
    used := used\\{step};
    used := used ∪ PARENTS(step);
    out := CONS(step,out);
  ENDIF
ENDWHILE
```

Beispiel	**Komplett**	**Extrahiert**	**Bemerkungen**
Lusk Overbeck 2	54	16	In jeder Gruppe gilt $g(g(x)) = x$. Es handelt sich um ein sehr kurzes Beispiel, das später genauer betrachtet wird.
Self Inverse	1703	69	In einem Ring mit $x^2 = x$ ist jedes Element selbstinvers.
Lusk Overbeck 3	5009	83	In einem Ring mit $x^2 = x$ ist die multiplikative Verknüpfung kommutativ. Näheres in A.1.
$6x = 0$	167992	199	In einem Ring mit $x^3 = x$ gilt $x + x + x + x + x + x = 0$.
DeMorgan	238706	151	Beweis eines der DeMorgan'schen Gesetze.
Lusk Overbeck 6	387273	190	In einem Ring mit $x^3 = x$ ist die multiplikative Verknüpfung kommutativ. Dieses Beispiel liegt an der Grenze der Leistungsfähigkeit heutiger Beweiser, es wird in A.4 näher untersucht.
Gruppe	414	44	Vervollständigung der Gruppenaxiome. Siehe A.2.
Fibgroup	1610	137	Vervollständigung einer zyklischen Gruppe.
Z22	18254	711	Größereres Beispiel einer Vervollständigung.

Tabelle 1: PCL-Schritte in vollständigen und extrahierten Protokollen

3.2 Ergebnisse der Extraktion

Ein direkter Vergleich von kompletten und extrahierten PCL-Beweisen ist wegen der großen Länge der Beweise hier nicht möglich. In Anhang A.1 befinden sich Protokolle in den verschiedenen Bearbeitungsstufen für ein Beispiel mittlerer Komplexität. Tabelle 1 zeigt die Anzahl der Schritte in verschiedenen PCL-Protokollen vor und nach der Entfernung der nicht verwendeten Schritte. Genauere Spezifikationen der einzelnen Beispiele befinden sich, soweit nicht anders vermerkt, in Anhang A.5.

Auch bei einer großen Anzahl von weiteren Beispielen ergeben sich ähnliche Ergebnisse. Dabei können zwei Beobachtungen gemacht werden. Zum ersten zeigt sich, daß mit zunehmender Größe der Beispiele die relative Anzahl der benötigten Schritte stark abnimmt. Dieses ist darauf zurückzuführen, daß selbst sehr starke Beweiser mit mächtigen Heuristiken in den immer größer werdenden Suchräumen viele unnötige Schritte machen.

Ein zweites Ergebnis ist, daß Vervollständigungen weit mehr Schritte als einfache Beweise benötigen. Dieser Effekt kann im wesentlichen mit der Tatsache erklärt werden, daß bei Vervollständigungen alle Regeln und Gleichungen des vervollständigten Systems gezeigt werden, während bei den vorliegenden Beweisen nur ein einziges Ziel zu zeigen ist.

3.3 Ein konkretes Beispiel

Nahezu alle interessanten Beispiele sind so umfangreich, daß sie selbst im Anhang nur in extrahierter Form wiedergegeben werden können. Eines der kürzesten Beispiele stammt aus [LO82]. Es wird bewiesen, daß in jeder Gruppe $g(g(x)) = x$ gilt. Das folgende ist die PCL-Beschreibung des Beweises, den das DISCOUNT-System (siehe [Pi92]) in einem seriellen Lauf gefunden hat.

```
 0 : tes-eqn : f(e(),x) = x : initial
 1 : tes-eqn : f(x,e()) = x : initial
 2 : tes-eqn : f(g(x),x) = e() : initial
 3 : tes-eqn : f(x,g(x)) = e() : initial
 4 : tes-eqn : f(f(x,y),z) = f(x,f(y,z)) : initial
 5 : tes-goal : g(g(a())) = a() : hypothesis
 6 : tes-rule : f(e(),x) -> x : orient(0,u)
 7 : tes-rule : f(x,e()) -> x : orient(1,u)
 8 : tes-eqn : e() = e() : cp(7,L,6,L)
 9 : tes-rule : f(g(x),x) -> e() : orient(2,u)
10 : tes-eqn : e() = g(e()) : cp(9,L,7,L)
11 : tes-rule : g(e()) -> e() : orient(10,x)
12 : tes-eqn : e() = f(e(),e()) : cp(9,L.1,11,L)
13 : tes-eqn : e() = e() : tes-red(12,R,6,L)
14 : tes-rule : f(x,g(x)) -> e() : orient(3,u)
15 : tes-eqn : e() = f(e(),e()) : cp(14,L.2,11,L)
16 : tes-eqn : e() = e() : tes-red(15,R,6,L)
17 : tes-eqn : e() = g(e()) : cp(14,L,6,L)
18 : tes-eqn : e() = e() : tes-red(17,R,11,L)
19 : tes-rule : f(f(x,y),z) -> f(x,f(y,z)) : orient(4,u)
20 : tes-eqn : f(e(),f(x,y)) = f(x,y) : cp(19,L.1,6,L)
21 : tes-eqn : f(x,y) = f(x,y) : tes-red(20,L,6,L)
22 : tes-eqn : f(x,f(y,e())) = f(x,y) : cp(19,L,7,L)
23 : tes-eqn : f(x,y) = f(x,y) : tes-red(22,L.2,7,L)
24 : tes-eqn : f(x,f(e(),y)) = f(x,y) : cp(19,L.1,7,L)
25 : tes-eqn : f(x,y) = f(x,y) : tes-red(24,L.2,6,L)
26 : tes-eqn : f(g(x),f(x,y)) = f(e(),y) : cp(19,L.1,9,L)
27 : tes-eqn : f(g(x),f(x,y)) = y : tes-red(26,R,6,L)
28 : tes-eqn : f(x,f(y,g(f(x,y)))) = e() : cp(19,L,14,L)
29 : tes-eqn : f(x,f(g(x),y)) = f(e(),y) : cp(19,L.1,14,L)
30 : tes-eqn : f(x,f(g(x),y)) = y : tes-red(29,R,6,L)
31 : tes-eqn : f(f(x,y),f(z,u)) = f(f(x,f(y,z)),u) : cp(19,L.1,19,L)
32 : tes-eqn : f(x,f(y,f(z,u))) = f(f(x,f(y,z)),u) : tes-red(31,L,19,L)
33 : tes-eqn : f(x,f(y,f(z,u))) = f(x,f(f(y,z),u)) : tes-red(32,R,19,L)
34 : tes-eqn : f(x,f(y,f(z,u))) = f(x,f(y,f(z,u))) : tes-red(33,R.2,19,L)
35 : tes-rule : f(g(x),f(x,y)) -> y : orient(27,u)
36 : tes-eqn : x = f(e(),f(e(),x)) : cp(35,L.1,11,L)
37 : tes-eqn : x = f(e(),x) : tes-red(36,R.2,6,L)
```

```
38 : tes-eqn : x = x : tes-red(37,R,6,L)
39 : tes-eqn : x = f(g(e()),x) : cp(35,L.2,6,L)
40 : tes-eqn : x = f(e(),x) : tes-red(39,R.1,11,L)
41 : tes-eqn : x = x : tes-red(40,R,6,L)
42 : tes-eqn : e() = f(g(x),x) : cp(35,L.2,7,L)
43 : tes-eqn : e() = e() : tes-red(42,R,9,L)
44 : tes-eqn : x = f(g(g(x)),e()) : cp(35,L.2,9,L)
45 : tes-eqn : x = g(g(x)) : tes-red(44,R,7,L)
46 : tes-eqn : g(x) = f(g(x),e()) : cp(35,L.2,14,L)
47 : tes-eqn : g(x) = g(x) : tes-red(46,R,7,L)
48 : tes-eqn : x = f(g(f(y,z)),f(y,f(z,x))) : cp(35,L.2,19,L)
49 : tes-eqn : f(g(x),f(f(x,y),z)) = f(y,z) : cp(19,L.1,35,L)
50 : tes-eqn : f(g(x),f(x,f(y,z))) = f(y,z) : tes-red(49,L.2,19,L)
51 : tes-eqn : f(x,y) = f(x,y) : tes-red(50,L,35,L)
52 : tes-eqn : f(x,y) = f(g(g(x)),y) : cp(35,L.2,35,L)
53 : tes-rule : g(g(x)) -> x : orient(45,x)
54 : tes-final : a() = a() : tes-red(5,L,53,L)
```

Selbst bei diesem sehr einfachen Beispiel ist ein Verfolgen des Beweises nur mit Mühe möglich. Die tatsächlich zum Beweis führenden Schritte sind nicht ohne weiteres zu erkennen. In der extrahierten Beschreibung ist der Beweis wesentlich übersichtlicher, insbesondere fällt sofort ins Auge, daß sogar eines der Axiome nicht verwendet wurde.

```
 0 : tes-eqn : f(e(),x) = x : initial
 1 : tes-eqn : f(x,e()) = x : initial
 2 : tes-eqn : f(g(x),x) = e() : initial
 4 : tes-eqn : f(f(x,y),z) = f(x,f(y,z)) : initial
 5 : tes-goal : g(g(a())) = a() : hypothesis
 6 : tes-rule : f(e(),x) -> x : orient(0,u)
 7 : tes-rule : f(x,e()) -> x : orient(1,u)
 9 : tes-rule : f(g(x),x) -> e() : orient(2,u)
19 : tes-rule : f(f(x,y),z) -> f(x,f(y,z)) : orient(4,u)
26 : tes-eqn : f(g(x),f(x,y)) = f(e(),y) : cp(19,L.1,9,L)
27 : tes-eqn : f(g(x),f(x,y)) = y : tes-red(26,R,6,L)
35 : tes-rule : f(g(x),f(x,y)) -> y : orient(27,u)
44 : tes-eqn : x = f(g(g(x)),e()) : cp(35,L.2,9,L)
45 : tes-eqn : x = g(g(x)) : tes-red(44,R,7,L)
53 : tes-rule : g(g(x)) -> x : orient(45,x)
54 : tes-final : a() = a() : tes-red(5,L,53,L)
```

4 Gliederung von Beweisen

Selbst relativ einfache Beweise werden unabhängig vom verwendeten Kalkül und der sonstigen Darstellung sehr schnell unübersichtlich, wenn sie ohne eine zusätzliche Gliederung präsentiert werden. Eine solche Gliederung entsteht, indem man geeignete Zwischenergebnisse als Lemmata formuliert und beweist. Da in Beweisen späterer Lemmata die bereits bewiesenen verwendet werden, ergibt sich eine hierarchische Struktur. Bei geeigneter Wahl der Lemmata entsteht außerdem eine thematische Gliederung des Beweises.

Leider geben Menschen für die Auswahl ihrer Lemmata nur selten konkrete Gründe an. Sehr häufig werden Lemmata gewählt, weil die Aussage "wichtig" oder "schön" ist - objektive Kriterien, die für eine automatische Beweisaufbereitung hilfreich sein könnten, werden kaum genannt. Im folgenden wird versucht, solche Kriterien zu finden.

Ein Teil dieser Kriterien verwendet relativ große Ausschnitte eines Beweisprotokolles. Dieses ist verständlich, da eine Gliederung durch Lemmata erst bei Beweisen einer bestimmten Mindestgröße sinnvoll wird. Eine Folge dieser Tatsache ist leider, daß die angeführten Beispiele relativ umfangreich und unübersichtlich sind.

4.1 Häufig verwendete Zwischenergebnisse

Ein naheliegendes Kriterium für die Auswahl von Lemmata ist die Häufigkeit, mit der das Faktum im weiteren Beweis verwendet wird. Ein Faktum, das überdurchschnittlich häufig verwendet wird, ist aus mehreren Gründen ein geeigneter Kandidat für ein Lemma. Zum ersten wird der Beweis kürzer und damit übersichtlicher, da ein Lemma nur noch beim ersten Auftreten und nicht bei jeder Anwendung bewiesen wird. Zum zweiten ist die häufige Verwendung ein starkes Indiz für die Bedeutung des Ergebnisses, während nur einmal auftretende Fakten in der Regel als reine Zwischenergebnisse gelten können.

Nach diesem Kriterium ist ein Schritt `step` ein Lemma, falls

$$|\texttt{CHILDREN(step)}| \geq \texttt{MINUSED}$$

gilt. `MINUSED` ist dabei eine Konstante, die angibt, wie oft ein Schritt verwendet werden muß, um ein Lemma zu werden.

Beispiel:

Man betrachte folgenden Auszug aus dem Beweis zum dritten Problem aus [LO82]. Genauere Angaben zu diesem Problem befinden sich inA.1.

```
  ...
   65 : tes-lemma : j(x,j(g(x),y)) -> y : orient(54,u)
   71 : tes-eqn : g(g(x)) = j(x,0()) : cp(65,L.2,20,L)
  ...
   87 : tes-eqn : x = j(y,j(x,g(y))) : cp(65,L.2,5,L)
  ...
   93 : tes-eqn : x = j(g(y),j(y,x)) : cp(65,L.2.1,89,L)
 2721 : tes-eqn : f(g(x),0()) = j(x,g(x)) : cp(65,L.2,2707,L)
  ...
 2961 : tes-eqn : f(g(x),x) = j(x,0()) : cp(65,L.2,2944,L)
```

```
 ...
3610 : tes-lemma : f(j(x,y),x) -> j(x,f(y,x)) : orient(3465,u)
3630 : tes-eqn : j(x,f(0(),x)) = f(x,x) : cp(3610,L.1,13,L)
 ...
3632 : tes-eqn : j(x,f(g(x),x)) = f(0(),x) : cp(3610,L.1,20,L)
 ...
4132 : tes-eqn : j(x,j(x,y)) = y : tes-red(65,L.2.1,4117,L)
 ...
4259 : tes-lemma : j(x,j(x,y)) -> y : orient(4132,u)
4263 : tes-eqn : j(j(x,y),f(j(x,y),x)) = f(j(x,y),y) : cp(2859,L.2,4259,L)
4264 : tes-eqn : j(j(x,y),j(x,f(y,x))) = f(j(x,y),y) : tes-red(4263,L.2,3610,
 ...
4268 : tes-eqn : j(x,f(x,y)) = f(j(y,x),x) : tes-red(4267,L,4259,L)
 ...
4835 : tes-eqn : j(x,j(x,f(y,x))) = j(x,j(x,f(x,y))) : tes-red(4834,L.2,3610,L)
4836 : tes-lemma : f(x,y) = j(y,j(y,f(y,x))) : tes-red(4835,L,4259,L)
4837 : tes-eqn : f(x,y) = f(y,x) : tes-red(4836,R,4259,L)
 ...
5008 : tes-final : f(a(),b()) = f(b(),a()) : instance(10,4837)
```

Es ist hier $\mathtt{CHILDREN}(65) = \{\mathtt{71,87,93,2721,2961,4132}\}$, $|\mathtt{CHILDREN}(65)| = \mathtt{6}$. Schritt 65 wird also 6 Mal im weiteren Beweis verwendet und ist damit ein attraktives Lemma. Die Schritte 3610 und 4259 werden je 4 Mal zitiert und sind ebenfalls als Lemmata geeignet. Dagegen wird Schritt 4835 auch in dem kompletten Beweisprotokoll nur einmal zitiert und ist deswegen nach diesem Lemma-Kriterium kein geeigneter Kandidat.

4.2 Wichtige Teilergebnisse

Ein anderes Maß für die Bedeutung eines PCL-Schrittes ist die Zahl der durch Verwendung des Schrittes repräsentierten Anwendungen von Axiomen oder Lemmata. Diese ergibt sich als Produkt der Häufigkeit, mit der ein Schritt angewendet wird, und der Länge der Beweiskette für diesen Schritt.

Definition: Die Funktion CHAINLEN

Sei `step` ein PCL-Schritt. `CHAINLEN(step)` gibt an, wie viele Umformungen mit Axiomen oder Lemmata notwendig waren, um `step` zu beweisen. Damit ist

$$\mathtt{CHAINLEN}(\mathtt{step}) = \left| \left\{ \mathtt{s} \in \mathtt{LPARENTS}^*(\mathtt{step}) \,\middle|\, \begin{array}{l} \mathtt{TYPE}(\mathtt{s}) = \mathtt{tes\text{-}lemma} \text{ oder} \\ \mathtt{EXPR}(\mathtt{s}) \in \{\mathtt{initial}, \mathtt{hypothesis}\} \end{array} \right\} \right|$$

Der Wert von `CHAINLEN(step)` läßt sich in der Praxis durch Rekursion über die Argumente von `EXPR(step)` effizient berechnen.

Satz: Rekursive Berechnung von CHAINLEN

Es gilt

$$\mathtt{CHAINLEN(step)} = \begin{cases} 1 & \text{falls } \mathtt{EXPR(step)} \in \{\mathtt{initial}, \mathtt{hypothesis}\} \\ 1 & \text{falls } \mathtt{TYPE(step)} = \mathtt{tes\text{-}lemma} \\ \displaystyle\sum_{\mathtt{s} \in \mathtt{LPARENTS(step)}} \mathtt{CHAINLEN(s)} & \text{sonst} \end{cases}$$

Damit ist ein Schritt nach diesem Kriterium ein Lemma, falls

$$|\mathtt{CHILDREN(step)}| \times \mathtt{CHAINLEN(step)} \geq \mathtt{MINWEIGHT}$$

ist. `MINWEIGHT` ist dabei wieder eine Konstante, die die Mindestanforderungen an ein Lemma definiert.

Beispiel: Man betrachte wieder einen Auszug aus dem Beweisprotokoll zum Beweis des Problems aus A.1:

```
...
   1 : tes-eqn : j(0(),x) = x : initial
...
   4 : tes-eqn : j(x,g(x)) = 0() : initial
...
   6 : tes-eqn : j(j(x,y),z) = j(x,j(y,z)) : initial
...
  12 : tes-rule : j(0(),x) -> x : orient(1,u)
...
  20 : tes-rule : j(x,g(x)) -> 0() : orient(4,u)
...
  45 : tes-rule : j(j(x,y),z) -> j(x,j(y,z)) : orient(6,u)
  53 : tes-eqn : j(x,j(g(x),y)) = j(0(),y) : cp(45,L.1,20,L)
  54 : tes-eqn : j(x,j(g(x),y)) = y : tes-red(53,R,12,L)
...
  65 : tes-rule : j(x,j(g(x),y)) -> y : orient(54,u)
  71 : tes-eqn : g(g(x)) = j(x,0()) : cp(65,L.2,20,L)
...
  87 : tes-eqn : x = j(y,j(x,g(y))) : cp(65,L.2,5,L)
...
  93 : tes-eqn : x = j(g(y),j(y,x)) : cp(65,L.2.1,89,L)
2721 : tes-eqn : f(g(x),0()) = j(x,g(x)) : cp(65,L.2,2707,L)
 ...
2961 : tes-eqn : f(g(x),x) = j(x,0()) : cp(65,L.2,2944,L)
 ...
4132 : tes-eqn : j(x,j(x,y)) = y : tes-red(65,L.2.1,4117,L)
 ...
5008 : tes-final : f(a(),b()) = f(b(),a()) : instance(10,4837)
```

ID(step)	CHAINLEN(step)	**Herleitung**
1	1	initial
4	1	initial
6	1	initial
12	1	CHAINLEN(1)
20	1	CHAINLEN(4)
45	1	CHAINLEN(6)
53	2	CHAINLEN(20) + CHAINLEN(45)
54	3	CHAINLEN(12) + CHAINLEN(53)
65	3	CHAINLEN(54)

Tabelle 2: Berechnung von CHAINLEN(65)

Bei der rekursiven Berechnung von CHAINLEN(65) ergeben sich die in Tabelle 2 wiedergegebenen Werte.

Es ist also CHAINLEN(65) $=$ 3. Wie bereits in 4.1 gezeigt wurde, ist |CHILDREN(65)| $=$ 6, damit ist CHAINLEN(65) $\times$ |CHILDREN(65)| $=$ 18. Anwendungen von Schritt 65 repräsentieren also insgesamt 18 Schritte mit den Axiomen 1, 4 und 6. Wählt man MINWEIGHT $\leq$ 18, so ist Schritt 65 nach dem hier gewählten Kriterium ein gutes Lemma.

4.3 Isolierte Teilergebnisse

Ein anderes Kriterium für die Bedeutung eines Ergebnisses ergibt sich aus der Struktur des Beweisbaumes. Ergebnisse, die große, ansonsten relativ isolierte Teile des Beweisbaumes zusammenfassen, sind gute Kandidaten für Lemmata. Sie repräsentieren Ergebnisse, die aus einer abgeschlossenen Teilmenge von Fakten folgen. So realisieren sie die bereits oben erwähnte thematische Gliederung. Die in [Li90] entwickelten Verfahren zur Strukturierung von prädikatenlogischen Beweisen verwenden ebenfalls isolierte Teilgraphen als Kriterium. In [LP90] wird vorgeschlagen, diese Verfahren auf Gleichheitsbeweise zu übertragen. Allerdings wird dort von einer Darstellung des Beweises als *Decomposition Graph* beziehungsweise *Equation Solution Graph* ausgegangen, eine direkte Übertragung auf die hier verwendete Darstellung ist also nicht möglich.

Ein einfaches Kriterium für den Grad der Isolierung eines Teilbeweises ist die Anzahl der in ihm vorkommenden Fakten, die auch außerhalb des Teilbeweises Verwendung finden. Eine genauere Differenzierung wird möglich, wenn man auch die Bedeutung dieser Fakten für den Gesamtbeweis berücksichtigt. So wird die Zahl der redundanten Teilbeweise minimiert und damit die Länge des Gesamtbeweises minimiert.

Realisiert wird dieses Kriterium, indem man das wie in 4.2 ermittelte Gewicht eines potentiellen Lemmas step mit der Summe der Gewichte der echten Zwischenergebnisse im Teilbeweis zu step vergleicht. Für diese Zwischenergebnisse wird dabei nur die Verwendung außerhalb des Beweises zu step berücksichtigt. Die folgende Funktion berechnet zu einem Beweisschritt step die Multimenge der Schritte, die zum Beweis von `step` dienende Zwischenergebnisse außerhalb des Beweises zu `step` zitieren.

Definition: Die Funktion `EXITS`

`EXITS(step)` ist die Menge der PCL-Schritte, die ein Zwischenergebnis im Beweis zu `step` zitieren, selbst aber nicht in diesem Beweis verwendet werden. Damit ist

$$\mathtt{EXITS}(\mathtt{step}) = \left(\bigcup_{\mathtt{s} \in \mathtt{LPARENTS}^*(\mathtt{step})} \mathtt{LCHILDREN}(\mathtt{s}) \right) \setminus (\mathtt{LPARENTS}^*(\mathtt{step}) \cup \{\mathtt{step}\})$$

Auch diese Funktion läßt sich effizienter durch Rekursion über die Argumente der PCL-Ausdrücke berechnen.

Satz: Rekursive Berechnung von `EXITS`

Es gilt

$$\mathtt{EXITS}(\mathtt{step}) = \left(\bigcup_{\mathtt{s} \in \mathtt{LPARENTS}(\mathtt{step})} (\mathtt{LCHILDREN}(\mathtt{s}) \cup \mathtt{EXITS}(\mathtt{s})) \right) \setminus (\mathtt{LPARENTS}^*(\mathtt{step}) \cup \{\mathtt{step}\})$$

Bei dieser Rekursion wird in der Praxis zu jedem Ausgang auch die Beweislänge des Schrittes gespeichert, der von dem Ausgang zitiert wird. Damit läßt sich das folgende Kriterium für die Lemmasuche nach isolierten Teilbeweisen mit einer einzelnen Rekursion entscheiden. Ein Schritt ist nach diesem Kriterium ein Lemma, falls

$$\left(\sum_{\mathtt{s} \in \mathtt{LPARENTS}^*(\mathtt{step})} |\mathtt{CHAINLEN}(\mathtt{s})| \times |\mathtt{CHILDREN}(\mathtt{s}) \cap \mathtt{EXITS}(\mathtt{step})| \right) \times \mathtt{WEIGHTFACTOR} + \mathtt{OFFSET}$$
$$< |\mathtt{CHILDREN}(\mathtt{step})| \times \mathtt{CHAINLEN}(\mathtt{step})$$

Die beiden Parameter `WEIGHTFACTOR` und `OFFSET` erlauben die Steuerung der Suche nach Lemmata mit diesem Kriterium.

Beispiel: Man betrachte auch hier einen Auszug aus dem Beweis aus A.1.

```
...
  1 : tes-eqn : j(0(),x) = x : initial
...
  4 : tes-eqn : j(x,g(x)) = 0() : initial
...
  6 : tes-eqn : j(j(x,y),z) = j(x,j(y,z)) : initial
...
 12 : tes-rule : j(0(),x) -> x : orient(1,u)
...
 20 : tes-rule : j(x,g(x)) -> 0() : orient(4,u)
 45 : tes-rule : j(j(x,y),z) -> j(x,j(y,z)) : orient(6,u)
```

```
  53 : tes-eqn : j(x,j(g(x),y)) = j(0(),y) : cp(45,L.1,20,L)
  54 : tes-eqn : j(x,j(g(x),y)) = y : tes-red(53,R,12,L)
  59 : tes-eqn : j(x,j(y,z)) = j(y,j(z,x)) : cp(5,L,45,L)
  65 : tes-rule : j(x,j(g(x),y)) -> y : orient(54,u)
  71 : tes-eqn : g(g(x)) = j(x,0()) : cp(65,L.2,20,L)
 ...
  87 : tes-eqn : x = j(y,j(x,g(y))) : cp(65,L.2,5,L)
 ...
  93 : tes-eqn : x = j(g(y),j(y,x)) : cp(65,L.2.1,89,L)
 ...
2721 : tes-eqn : f(g(x),0()) = j(x,g(x)) : cp(65,L.2,2707,L)
2722 : tes-eqn : f(g(x),0()) = 0() : tes-red(2721,R,20,L)
 ...
2883 : tes-eqn : j(g(x),f(g(x),x)) = f(g(x),0()) : cp(2859,L.2,20,L)
 ...
2961 : tes-eqn : f(g(x),x) = j(x,0()) : cp(65,L.2,2944,L)
 ...
3632 : tes-eqn : j(x,f(g(x),x)) = f(0(),x) : cp(3610,L.1,20,L)
 ...
3986 : tes-eqn : g(x) = j(x,0()) : tes-red(3985,R.2,20,L)
 ...
4132 : tes-eqn : j(x,j(x,y)) = y : tes-red(65,L.2.1,4117,L)
 ...
4267 : tes-eqn : j(x,j(x,j(y,f(y,x)))) = f(j(x,y),y) : tes-red(4266,L.2,45,L)
 ...
```

Stellt man diesen Beweisausschnitt als Graph da, so ergibt sich Abbildung 1.

Man erkennt, daß der Schritt mit der Nummer 65 insgesamt 6 Mal verwendet wird. Es gilt $\texttt{CHAINLEN}(65) = 3$, damit werden durch diesen Schritt 18 Anwendungen der Gleichungen aus 1, 4 und 6 repräsentiert. Es werden die Schritte 20 und 45 insgesamt 7 Mal außerhalb des Beweises zu Schritt 65 zitiert, jedes dieser Zitate steht jedoch nur für eine einzelne Gleichheitsumformung. Damit ist der Teilbeweis zu Schritt 65 relativ isoliert, Schritt 65 also ein brauchbares Lemma.

4.4 Syntaktische Kriterien

Ein Mensch nimmt die Baumstruktur eines Beweises nur selten bewußt wahr. Auch läßt sich die Bedeutung eines Zwischenergebnisses für einen konkreten Beweis in der Regel erst nach Abschluß des Beweises feststellen. Die Entscheidung, ob ein Ergebnis zum Lemma gemacht wird, ist also weitgehend von lokalen Kriterien abhängig. Es liegt nahe, diesen Prozeß an Hand von syntaktischen Kriterien nachzuvollziehen.

Dabei bieten sich selbst bei Beschränkung auf einfache Kriterien viele verschiedenen Möglichkeiten. Naheliegend ist es etwa, Zwischenergebnisse mit kleinen Termen als Lemmata auszuzeichnen, da sie für Menschen relativ gut verständliche und allgemeine Konzepte repräsentieren. Ein Maß für die Größe eines Termes ist dabei die Zahl der in ihm vorkommenden Funktionssymbole und Variablen.

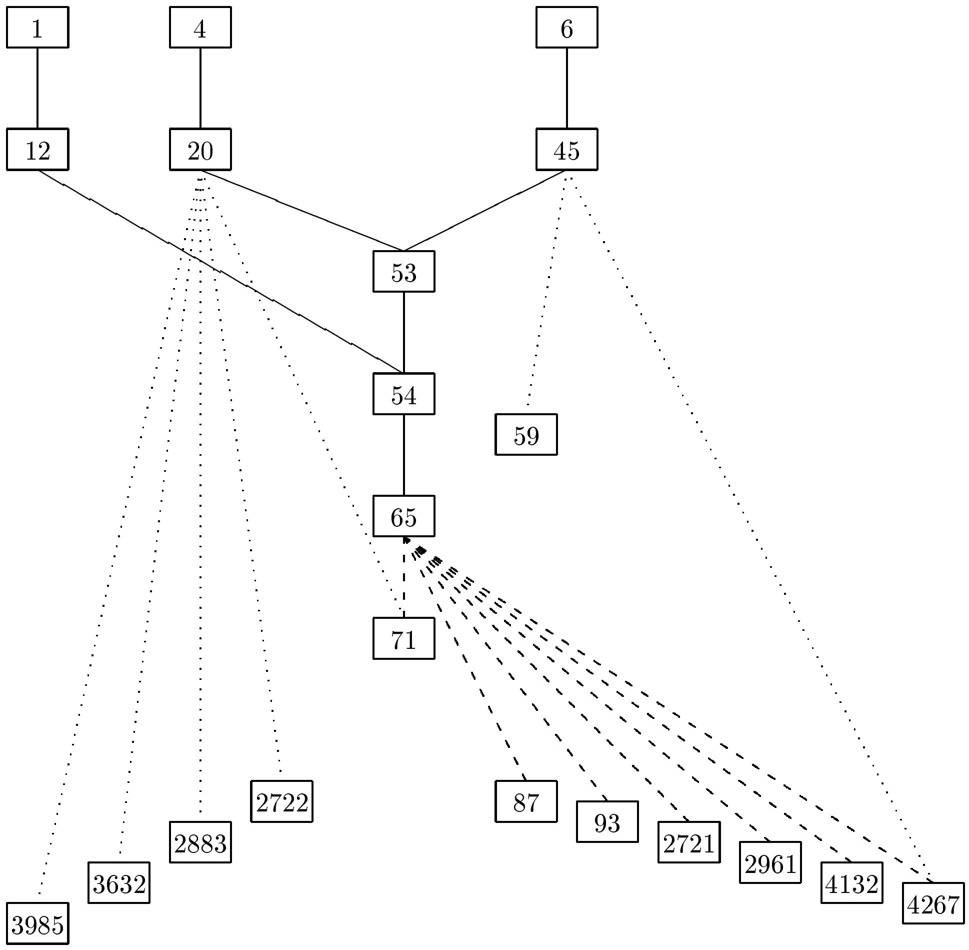

Abbildung 1: Beweisgraph zur Herleitung und Verwendung von Schritt 65

Definition: Länge eines Termes

Die Länge eines Termes ist die Anzahl der in ihm vorkommenden Funktionssymbole und Variablen. Es gilt

$$|\mathtt{t}| = \begin{cases} 1 & \text{falls } \mathtt{t} \in \mathtt{V} \\ 1 + \sum\limits_{1 \leq \mathtt{i} \leq \mathtt{n}} |\mathtt{t_i}| & \text{falls } \mathtt{t} \equiv \mathtt{f}(\mathtt{t_1}, \ldots, \mathtt{t_n}) \end{cases}$$

Sei `step` ein PCL-Schritt mit $\mathtt{FACT(step)} = \mathtt{s{=}t}$ oder $\mathtt{FACT(step)} = \mathtt{s} \rightarrow \mathtt{t}$. Wählt man die maximale Termgröße als Kriterium, so werden Schritte mit

$$\mathtt{max}(|\mathtt{s}|, |\mathtt{t}|) \leq \mathtt{MAXSIZE}$$

Lemmata. Ebenso kann man aber auch die durchschnittliche Termgröße heranziehen. Dann lautet die Forderung an Lemmata

$$\frac{|\mathtt{t_1}| + |\mathtt{t_2}|}{2} \leq \mathtt{AVERAGESIZE}$$

Beispiel:

Beispiele für syntaktische Lemmata findet man in fast allen Beweisprotokollen ohne Schwierigkeiten. Verwendet man wieder das Protokoll aus A.1, so findet man unter anderem die folgenden potentiellen Lemmata:

```
  ...
   72 : tes-eqn : g(g(x)) = x : tes-red(71,R,13,L)
  ...
 3987 : tes-eqn : g(x) = x : tes-red(3986,R,13,L)
  ...
```

Bei Schritt 72 ist die maximale Termgröße $|\mathtt{g(g(x))}| = 3$, die durchschnittliche Termgröße

$$\frac{|\mathtt{g(g(x))}| + |\mathtt{x}|}{2} = \frac{3+1}{2} = 2$$

Damit sind die Terme relativ klein und der Schritt ist als Lemma geeignet. Die Terme im Faktum zu Schritt 3987 sind beide klein, insbesondere ist hier aber auch der größere Term mit $|\mathtt{g(x)}| = 2$ ungewöhnlich klein.

Berücksichtigt man, daß die hier untersuchten PCL-Protokolle Beweise beschreiben, die von einem vervollständigungsbasierten Beweiser stammen, so bieten sich andere Kriterien an. Fakten, die die Größe des Termes stark verändern, sind interessante Lemma-Kandidaten, da sie oft wichtige Vereinfachungen der Terme ermöglichen. Die herkömmlichen Reduktionsordnungen bieten in der Regel kein Maß für den "Abstand" in der Größe zweier Terme, außerdem fehlen Informationen über die Reduktionsordnung in PCL. Aber auch hier bietet ein Vergleich der Termgrößen eine einfache und intuitiv naheliegende Möglichkeit, den Größenunterschied zwischen den beiden Seiten des Faktums in einer zumindest heuristisch günstigen Weise zu vergleichen. Für ein mögliches Lemma sollte also nach diesem Kriterium

$$\frac{\mathtt{max}(|\mathtt{s}|,|\mathtt{t}|)}{\mathtt{min}(|\mathtt{s}|,|\mathtt{t}|)} \geq \mathtt{MINFAK}$$

gelten. In seltenen Fällen kann es auch interessant sein, einen geringen Größenunterschied als Lemma-Kriterium zu wählen. Man beachte etwa, daß sowohl für Assoziativität als auch für Kommutativität die beiden Seiten der beschreibenden Gleichungen die gleiche Länge haben. Die Forderung an ein Lemma lautet nach diesem Kriterium

$$\frac{\mathtt{max}(|\mathtt{s}|,|\mathtt{t}|)}{\mathtt{min}(|\mathtt{s}|,|\mathtt{t}|)} \leq \mathtt{MAXFAK}$$

Beispiel: Auch für diese Kriterien finden sich in dem selben Beweisprotokoll genügend Beispiele.

```
  ...
  54 : tes-eqn : j(x,j(g(x),y)) = y : tes-red(53,R,12,L)
  ...
4837 : tes-eqn : f(x,y) = f(y,x) : tes-red(4836,R,4259,L)
  ...
```

Für Schritt 54 ist

$$\frac{|\mathtt{j(x,j(g(x),y))}|}{|\mathtt{y}|} = \frac{6}{1} = 6,$$

also ist das Faktum zu Schritt 54 potentiell zu "starken" Reduktionen fähig. Schließlich sind die Terme von Schritt 4837 beide von gleicher Größe, dabei aber außergewöhnlich klein. Auch diese Schritt wäre also als Lemma geeignet.

4.5 Analyse der verwendeten Inferenzschritte

Die für die Herleitung eines Faktums verwendeten Inferenzschritte können ebenfalls Grundlage eines Kriteriums zur Lemmasuche sein. Ein solches Kriterium ist weniger an menschlichen Gewohnheiten als an dem Wissen über den Ablauf einer Vervollständigung orientiert. So ist etwa die Bildung eines neuen kritischen Paares ein konstruktiver Vorgang, während die Orientierung einer Gleichung zu einer Regel nur einen rein technischen Vorgang darstellt, der kein neues Wissen über die Gleichungstheorie ergibt.

Ordnet man den unterschiedlichen Inferenzschritten verschiedene Gewichte zu und berechnet die Summe dieser Gewichte für die zur Herleitung eines Faktums notwendigen Schritte, so kann man diese Summe als Ausgangspunkt für die Suche nach Lemmata verwenden.

Konstante	Wert
INITW	1
HYPOW	0
QUOTEW	0
ORIENTW	0
CPW	3
REDW	2
INSW	0
MINWEIGHT	15

Tabelle 3: Belegung der Konstanten für die Berechnung von `IWEIGHT`

Definition: Die Funktion `IWEIGHT`

Die Funktion `IWEIGHT` ist auf PCL-Schritten und Ausdrücken definiert und ordnet diesen ein Gewicht zu. Sei `step` ein PCL-Schritt und seien `INITW`, `HYPOW`, `QUOTEW`, `ORIENTW`, `CPW`, `REDW` und `INSW` Konstanten.

- $\texttt{IWEIGHT}(\texttt{step}) = \begin{cases} \texttt{INITW} & \text{falls } \texttt{TYPE}(\texttt{step}) = \texttt{tes-lemma} \\ \texttt{IWEIGHT}(\texttt{EXPR}(\texttt{step})) & \text{sonst} \end{cases}$
- $\texttt{IWEIGHT}(\texttt{expr})$

$$= \begin{cases} \texttt{INITW} & \text{falls } \texttt{expr} = \texttt{initial} \\ \texttt{HYPOW} & \text{falls } \texttt{expr} = \texttt{hypothesis} \\ \texttt{QUOTEW} + \texttt{IWEIGHT}(\texttt{STEP}(\texttt{<id>})) & \text{falls } \texttt{expr} = \texttt{<id>} \\ \texttt{ORIENTW} + \texttt{IWEIGHT}(\texttt{expr}_1) & \text{falls } \texttt{expr} = \texttt{orient}(\texttt{expr}_1, \texttt{d}) \\ \texttt{CPW} + \texttt{IWEIGHT}(\texttt{expr}_1) + \texttt{IWEIGHT}(\texttt{expr}_2) & \text{falls } \texttt{expr} = \texttt{cp}(\texttt{expr}_1, p, \texttt{expr}_2, s) \\ \texttt{REDW} + \texttt{IWEIGHT}(\texttt{expr}_1) + \texttt{IWEIGHT}(\texttt{expr}_2) & \text{falls } \texttt{expr} = \texttt{tes-red}(\texttt{expr}_1, p, \texttt{expr}_2, s) \\ \texttt{INSW} + \texttt{IWEIGHT}(\texttt{expr}_1) + \texttt{IWEIGHT}(\texttt{expr}_2) & \text{falls } \texttt{expr} = \texttt{instance}(\texttt{expr}_1, \texttt{expr}_2) \end{cases}$$

Diese Definition erlaubt es, die Zahl der verwendeten Inferenzschritten durch eine gewichtete Summe zu repräsentieren. Der Vergleich dieser Summe mit einem Grenzwert entscheidet, ob ein Schritt zum Lemma wird. Gefordert wird also

$$\texttt{IWEIGHT}(\texttt{step}) \geq \texttt{MINWEIGHT}$$

für eine Konstante `MINWEIGHT`, die die Minimalanforderungen an ein Lemma definiert.

Beispiel: Für ein konkretes Beispiel müssen die Konstanten in der Definition für die Funktion `IWEIGHT` belegt werden. Die im Beispiel verwendete Belegung gibt Tabelle 3 wieder.

In diesem Beispiel wurden ausschließlich die Gewichte der PCL-Ausdrücke für die Lemma-Generierung in dem Protokoll aus A.1 verwendet.

ID(step)	IWEIGHT(step)	**Herleitung**
1	1	INITW
2	1	INITW
4	1	INITW
6	1	INITW
12	1	ORIENTW + QUOTEW + IWEIGHT(1)
13	1	ORIENTW + QUOTEW + IWEIGHT(2)
20	1	ORIENTW + QUOTEW + IWEIGHT(4)
45	1	ORIENTW + QUOTEW + IWEIGHT(6)
53	5	CPW + QUOTEW + IWEIGHT(45) + QUOTEW + IWEIGHT(20)
54	8	REDW + QUOTEW + IWEIGHT(53) + QUOTEW + IWEIGHT(12)
65	8	ORIENTW + QUOTEW + IWEIGHT(54)
71	12	CPW + QUOTEW + IWEIGHT(65) + QUOTEW + IWEIGHT(20)
72	15	REDW + QUOTEW + IWEIGHT(71) + QUOTEW + IWEIGHT(13)

Tabelle 4: Berechnung von IWEIGHT(72)

```
 ...
   1 : tes-eqn : j(0(),x) = x : initial
   2 : tes-eqn : j(x,0()) = x : initial
   4 : tes-eqn : j(x,g(x)) = 0() : initial
 ...
   6 : tes-eqn : j(j(x,y),z) = j(x,j(y,z)) : initial
 ...
  12 : tes-rule : j(0(),x) -> x : orient(1,u)
  13 : tes-rule : j(x,0()) -> x : orient(2,u)
  20 : tes-rule : j(x,g(x)) -> 0() : orient(4,u)
  45 : tes-rule : j(j(x,y),z) -> j(x,j(y,z)) : orient(6,u)
  53 : tes-eqn : j(x,j(g(x),y)) = j(0(),y) : cp(45,L.1,20,L)
  54 : tes-eqn : j(x,j(g(x),y)) = y : tes-red(53,R,12,L)
 ...
  65 : tes-rule : j(x,j(g(x),y)) -> y : orient(54,u)
  71 : tes-eqn : g(g(x)) = j(x,0()) : cp(65,L.2,20,L)
  72 : tes-eqn : g(g(x)) = x : tes-red(71,R,13,L)
 ...
 115 : tes-eqn : x = j(g(y),j(x,y)) : cp(97,L.2.2,89,L)
 ...
 181 : tes-eqn : g(x) = j(g(j(x,y)),y) : cp(160,L.2,126,L)
 ...
2721 : tes-eqn : f(g(x),0()) = j(x,g(x)) : cp(65,L.2,2707,L)
 ...
```

Bei der rekursiven Berechnung von IWEIGHT(72) ergeben sich die in Tabelle 4 angegebenen Werte. Damit gilt $\texttt{IWEIGHT(72)} = 15 \geq \texttt{MINWEIGHT} = 15$, Schritt 72 ist also nach diesem Kriterium ein Lemma.

4.6 Unterteilung langer Beweise

Auch ein Beweis, in dem kein Zwischenergebnis von herausragender Bedeutung ist, sollte bei Erreichen einer bestimmten Maximallänge durch Lemmata gegliedert werden. Da sich die Länge des Beweises zu einem Faktum aus der PCL-Beschreibung des Beweises einfach und exakt berechnen läßt, genügt der Vergleich der tatsächlichen Länge eines Beweises mit einem Grenzwert, um ein hinreichendes Kriterium für ein Lemma zu haben. Damit wird ein Schritt zum Lemma, falls

$$\mathtt{CHAINLEN}(\mathtt{step}) > \mathtt{MAXLEN}$$

gilt. Bei der Wahl von `MAXLEN` ist zu beachten, daß bei der Kombination von zwei Beweisketten, etwa bei der Reduktion oder bei der Bildung von kritischen Paaren, die Länge der entstehenden Beweiskette die Summe der Länge der Einzelketten ist. Damit können Beweislängen bis zu $2 \times \mathtt{MAXLEN}$ entstehen.

Beispiel: Die Häufigkeit der Lemmata nach diesem Kriterium hängt natürlich direkt von der Konstante `MAXLEN` ab. Um ein überschaubares Beispiel wählen zu können, wird hier `MAXLEN` auf den Wert 5 gesetzt. Betrachtet man wieder den Beweis zum dritten Beispiel aus [LO82] ohne weitere Lemmata, so findet man folgende Situation:

```
...
  1 : tes-eqn : j(0(),x) = x : initial
  2 : tes-eqn : j(x,0()) = x : initial
  4 : tes-eqn : j(x,g(x)) = 0() : initial
...
  6 : tes-eqn : j(j(x,y),z) = j(x,j(y,z)) : initial
...
 12 : tes-rule : j(0(),x) -> x : orient(1,u)
 13 : tes-rule : j(x,0()) -> x : orient(2,u)
 20 : tes-rule : j(x,g(x)) -> 0() : orient(4,u)
 45 : tes-rule : j(j(x,y),z) -> j(x,j(y,z)) : orient(6,u)
 53 : tes-eqn : j(x,j(g(x),y)) = j(0(),y) : cp(45,L.1,20,L)
 54 : tes-eqn : j(x,j(g(x),y)) = y : tes-red(53,R,12,L)
...
 65 : tes-rule : j(x,j(g(x),y)) -> y : orient(54,u)
 71 : tes-eqn : g(g(x)) = j(x,0()) : cp(65,L.2,20,L)
 72 : tes-eqn : g(g(x)) = x : tes-red(71,R,13,L)
...
  89 : tes-rule : g(g(x)) -> x : orient(72,u)
  93 : tes-eqn : x = j(g(y),j(y,x)) : cp(65,L.2.1,89,L)
...
```

Die Berechnung der Beweislängen für die einzelnen Schritte findet man in Tabelle 5.

Es ist $\mathtt{CHAINLEN}(93) > \mathtt{MAXLEN} = 5$, damit ist der Schritt 93 ein nach diesem Kriterium geeignetes Lemma. Schritt 93 ist auch der erste Schritt, der dieses Kriterium erfüllt. Wäre etwa Schritt 89 ein Lemma, so ergäbe sich $\mathtt{CHAINLEN}(93) = 4$ und der Schritt würde nicht als Lemma gewählt.

ID(step)	CHAINLEN(step)	**Herleitung**
1	1	initial
2	1	initial
4	1	initial
6	1	initial
12	1	CHAINLEN(1)
13	1	CHAINLEN(2)
20	1	CHAINLEN(4)
45	1	CHAINLEN(6)
53	2	CHAINLEN(20) + CHAINLEN(45)
54	3	CHAINLEN(12) + CHAINLEN(53)
65	3	CHAINLEN(54)
71	4	CHAINLEN(65) + CHAINLEN(20)
72	5	CHAINLEN(71) + CHAINLEN(13)
89	5	CHAINLEN(72)
93	8	CHAINLEN(65) + CHAINLEN(89)

Tabelle 5: Berechnung von CHAINLEN(93)

Ein anderer Ansatz ergibt sich aus dem Versuch, solche Beweise in Teile gleicher Länge zu zerlegen. Ein solcher Versuch ist jedoch mit großen Schwierigkeiten verbunden, da er eine Generierung der Lemmata in der Reihenfolge des Auftretens der Ergebnisse unmöglich macht. Statt dessen macht er eine Analyse zukünftiger Schritte notwendig. Nur so kann der relevante Teilbeweis erkannt werden. Da die Beweiskette nicht durch Anwendung einzelner Axiome oder Lemmata, sondern durch Verknüpfung von Fakten, die ihrerseits größere Beweisketten repräsentieren, erfolgt, kann eine Unterteilung in Ketten gleicher Länge auch so nicht garantiert werden. Den so gefundenen Lemmata kommt nur eine relativ geringe Bedeutung zu. Deshalb ist der mit einer solchen Suche verbundene Aufwand meiner Ansicht nach nicht zu vertreten.

4.7 Negative Kriterien für Lemmata

Neben hinreichenden Bedingungen kann man auch notwendige Bedingungen für Lemmata definieren. Ein PCL-Schritt, der einem aus solchen Bedingungen gewonnenes Kriterium nicht genügt, kann auch dann kein Lemma werden, wenn er andere Bedingungen für ein Lemma erfüllt.

In der gegenwärtigen Implementierung von PCL können etwa Schritte vom Typ `tes-goal` niemals Lemmata werden. Sie repräsentieren zwar Zwischenergebnisse auf dem Weg zum Beweisziel, werden aber nicht konstruktiv, sondern destruktiv gewonnen. Die vorgestellten Lemma-Kriterien sind nicht geeignet, Schritte dieser Art zu bewerten. Außerdem bringen Lemmata vom Typ `tes-goal` Probleme bei der weiteren Bearbeitung des Beweises mit sich. Bei den bisher untersuchten Beweisen hat sich gezeigt, daß diese Einschränkung keine wesentliche ist, da nur sehr wenige Schritte dieses Typs im endgültigen Beweis auftreten.

Neben diesen technischen Gründen sind die Kriterien aus 4.1 und 4.6 geeignet, Kriterien zu formulieren, die gegen einen Schritt sprechen. Auch diese können zu notwendigen Bedingungen werden.

Ein Lemma sollte in einem Beweis mindestens einmal verwendet werden. Es kann jedoch auch sinnvoll sein, mehrere Verwendungen zu verlangen. Damit muß ein Lemma die Bedingung

$$|\texttt{CHILDREN}(\texttt{step})| \geq \texttt{MINUSED}$$

erfüllen. Man beachte, daß die Bedingung zwar die selbe Form wie in 4.1 hat, hier aber eine notwendige und keine hinreichende Bedingung beschreibt.

Analog kann man für Lemmata eine bestimmte Mindestbeweislänge fordern. Dieses ist besonders sinnvoll, da etwa Schritte mit Beweislänge 1 nur Zitate von bereits bekannten Lemmata und Axiomen darstellen. Die Bedingung hier lautet

$$\texttt{CHAINLEN}(\texttt{step}) \geq \texttt{MINLEN}$$

Beispiel: Wir betrachten wieder einen Ausschnitt aus dem Protokoll aus A.1.

```
 ...
   4 : tes-eqn : j(x,g(x)) = 0() : initial
 ...
  20 : tes-rule : j(x,g(x)) -> 0() : orient(4,u)
 ...
  53 : tes-eqn : j(x,j(g(x),y)) = j(0(),y) : cp(45,L.1,20,L)
 ...
  71 : tes-eqn : g(g(x)) = j(x,0()) : cp(65,L.2,20,L)
 ...
2722 : tes-eqn : f(g(x),0()) = 0() : tes-red(2721,R,20,L)
 ...
2883 : tes-eqn : j(g(x),f(g(x),x)) = f(g(x),0()) : cp(2859,L.2,20,L)
 ...
3632 : tes-eqn : j(x,f(g(x),x)) = f(0(),x) : cp(3610,L.1,20,L)
 ...
3986 : tes-eqn : g(x) = j(x,0()) : tes-red(3985,R.2,20,L)
3987 : tes-eqn : g(x) = x : tes-red(3986,R,13,L)
4117 : tes-rule : g(x) -> x : orient(3987,u)
 ...
```

Es ist $\texttt{CHILDREN}(20) = \{53, 71, 2722, 2883, 3632, 3986\}$, Schritt 20 wird also 6 Mal im weiteren Beweis verwendet. Allerdings ist $\texttt{CHAINLEN}(20) = 1$, denn Schritt 20 ist nur die gerichtete Version der initialen Gleichung 4. Schritt 20 ist also kein geeignetes Lemma.

Auch ist Schritt 3987 nach den syntaktischen Kriterien aus 4.4 ein potentielles Lemma. Dieser Schritt wird jedoch nur noch ein Mal verwendet und ist deshalb nur bedingt geeignet.

4.8 Kombination verschiedener Kriterien

Für das Erkennen von geeigneten Lemmata in der Praxis bietet sich eine Kombination verschiedener Kriterien an. Einige Möglichkeiten, die verschiedenen Methoden zu kombinieren, werden im folgenden mit ihren wichtigsten Vor- und Nachteilen vorgestellt. Viele dieser Kombinationen erfordern eine getrennte Behandlung der negativen Kriterien aus 4.7. In den hier vorgestellten Verfahren wird darauf in der Regel hingewiesen.

(1) Für jeden Schritt wird geprüft, ob er nach einem der zu verwendenden Kriterien als Lemma geeignet ist. Ist die der Fall, so wird der Schritt zum Lemma. Dieses einfache Vorgehen hat verschiedene Vorteile:

   - Das Verfahren ist einfach zu implementieren.
   - Bei Bewertung eines Schrittes sind alle vorherigen Lemmata bekannt und können für die Entscheidung benutzt werden.
   - Das Verfahren ist effizient, da das PCL-Protokoll nur ein einziges Mal durchlaufen werden muß.

   Diesen Vorteilen stehen auch einige Nachteile gegenüber:

   - Alle Kriterien sind von gleicher Bedeutung. Insbesondere kann durch die Auswahl eines Lemmas nach einem relativ schwachen Kriterium - etwa der Analyse der Beweislänge aus 4.6 - die Auswahl eines weit besser geeigneten Lemmas verhindert werden.
   - Auch eine Unterscheidung, ob ein Schritt nach nur einem einzigen Kriterium als Lemma geeignet ist, oder ob mehrere Gründe für den Schritt sprechen, ist nicht möglich.
   - Die negativen Kriterien aus 4.7 können nur als notwendige Bedingungen berücksichtigt werden, eine gewichtete Entscheidung ist nicht möglich.

(2) Die Lemma-Kriterien werden nach "Qualität" geordnet, der gesamte Beweis dann mit jeweils einem Kriterium nach Lemmata durchsucht. Eine wichtige Gemeinsamkeit dieses Vorgehens mit dem Verfahren aus (1) ist, daß auch hier nur ein Kriterium erfüllt sein muß, um einen Schritt zu einem Lemma zu machen. Vorteile dieses Vorgehens sind:

   - Wichtige Kriterien haben Priorität, die Wahrscheinlichkeit, daß die besten Lemmata gefunden werden, ist höher als bei (1).
   - Der Implementierungsaufwand ist vertretbar.

   Allerdings werden diese Vorteile mit einigen neuen Nachteilen erkauft:

   - Die Lemmata werden nicht alle in der Reihenfolge ihres Auftretens erzeugt. Da die vorgestellten Kriterien nur frühere Schritte berücksichtigen, können Lemmata erzeugt werden, die bereits erzeugte Lemmata nachträglich überflüssig machen.
   - Das PCL-Protokoll muß für jedes Kriterium einmal durchlaufen werden, der Rechenaufwand ist also um ein mehrfaches höher als bei (1). Die bereitete allerdings bei den untersuchten Beweisen keine Probleme.
   - Nach wie vor können die negativen Kriterien nur als notwendige Bedingungen verwendet werden.

(3) Den verschiedenen Kriterien werden Gewichte zugeordnet, die gewichtete Summe aller Kriterien wird mit einem Grenzwert verglichen. Auch dieser Ansatz erlaubt eine Abstufung der Lemma-Kriterien, hier können sogar prinzipiell alle Kriterien gewichtet werden. Dabei kombinieren sich ein großer Teil der Vorteile aus (1) und (2):

- Der Implementierungsaufwand ist vertretbar.
- Bei Bewertung eines Schrittes sind alle vorherigen Lemmata bekannt und können für die Entscheidung benutzt werden.
- Das Verfahren ist relativ effizient, das PCL-Protokoll muß nur ein einziges Mal durchlaufen werden.
- Auch die negativen Kriterien können mit einem Gewicht berücksichtigt werden, damit können ansonsten sehr gute Lemmata auch berücksichtigt werden, wenn (schwächere) Gründe dagegen sprechen.
- Auf der Grundlage dieses Kriteriums lassen sich andere, allerdings extrem aufwendige, Verfahren konstruieren, die etwa eine globale Maximierung der Summe der Lemmagewichte anstreben.

Auch dieses Verfahren hat einige Nachteile:

- Wie bei (1) ist auch hier normalerweise nicht gewährleistet, daß die global besten Lemmata gefunden werden.
- Wie bereits zu Beginn des Kapitels erwähnt wurde, gibt es keine exakten Kriterien für die Qualität der Lemmata. Die bisherigen Erfahrungen genügen nicht, eine vernünftige quantitative Bewertung der Kriterien vorzunehmen.

(4) Betrachtet man die vorgestellten Lemma-Kriterien, so stellt man fest, daß die meisten auf dem Vergleich nummerischer Werte beruhen. Verrechnet man diese Werte, so lassen sich statt der bisherigen zweiwertigen Empfehlung mehrwertige Gewichte direkt errechnen. Die Summe dieser Gewichte wird wie in (3) für die Entscheidung verwendet. Die Vorteile von (3) bleiben zum größten Teil erhalten, zusätzliche Vorteile sind:

- Das Konzept erlaubt flexiblere und potentiell mächtigere Verfahren.
- Die negativen Kriterien aus 4.7 können mit den Kriterien aus 4.1 und 4.6 zusammengefaßt und einheitlich behandelt werden.

Dem stehen allerdings 2 gewichtige Nachteile entgegen:

- In weit stärkerem Maße wie bei (3) macht sich hier die geringe Erfahrung mit den Kriterien bemerkbar. Wie die ermittelten Werte sinnvoll auf eine mehrwertige Skala abgebildet werden können, ist im Rahmen dieser Arbeit noch nicht untersucht worden.
- Der Implementierungsaufwand ist, zumindest von der gegenwärtigen Implementierung ausgehend, relativ hoch.

(5) Eine letzte Möglichkeit ist schließlich eine allgemeine Sprache zum Beschreiben von Lemma-Kriterien, mit Zugriff auf die in diesem Kapitel vorgestellten Funktionen und eventuell auf das Beweisprotokoll selbst. Der Vorteil dieses Ansatzes ist offensichtlich:

- Man erhält maximale Flexibilität und die Möglichkeit, auch die potentiell stärksten Kombinationen von Kriterien zu formulieren.

Leider stehen dem auch entsprechende Nachteile gegenüber:

- Der Implementierungsaufwand ist sehr hoch.
- Die Effizienz ist deutlich geringer als bei den anderen Methoden, da eine eigene Beschreibungssprache interpretiert werden muß.
- Die Anforderungen an einen Benutzer, der die zur Verfügung gestellten Möglichkeiten nutzen will, sind sehr hoch.

Implementiert wurden im Rahmen dieser Arbeit die Methoden (1) und (2). Dabei hat sich gezeigt, daß selbst einzelne Kriterien in der Regel für eine gute Strukturierung des Beweises genügen. Die Ergebnisse der Kombinationen nach (1) und (2) erscheinen ebenfalls bisher hinreichend.

Interessant für die Zukunft sind meiner Ansicht nach vor allem die Methoden (3) und (4), während Methode (5) weniger vielversprechend scheint. Für die wenigen Spezialfälle, in denen die anderen Verfahren keine ausreichende Gliederung erreichen, erscheint es einfacher, diese von Hand vorzunehmen oder die benötigten Verfahren direkt in den C-Source-Code der Programme zu integrieren.

4.9 Ein Algorithmus zum Erkennen von Lemmata in PCL-Beschreibungen

Hier soll die wesentliche Struktur eines Algorithmus dargestellt werden, der die vorgestellten Lemma-Kriterien zur Suche nach Lemmata verwendet. Er verwendet alle vorgestellten Kriterien und die Kombination verschiedener Kriterien nach den Verfahren (1) und (2) aus 4.8.

Ein- und Ausgabe für den Algorithmus sind je ein (extrahiertes) PCL-Protokoll. In der Ausgabe sind dabei eventuelle Lemmata markiert.

Eingabe:	`in`	Eine Liste von PCL-Schritten
	`criteria`	Eine Liste von Funktionen, die einen Schritt bewerten.
	`iterate`	TRUE, falls die Kombination der Kriterien aus `criteria` nach 4.8 (2) erfolgen soll, FALSE, falls Methode (1) verwendet werden soll.
Ausgabe:	`out`	Eine Liste von PCL-Schritten
Variable:	`store`	Ebenso
	`step`	Ein einzelner PCL-Schritt
	`IsLemma(step)`	Die aktuelle Lemma-Funktion
Funktionen:	`NOTEMPTY(list)`	FALSE, falls `list` leer, TRUE sonst
	`FIRST(list)`	Erstes Element von `list`
	`EXCEPTFIRST(list)`	`list` ohne letztes Element
	`TYPE(step)`	Typ von `step`(siehe 2.2)
	`APPEND(list,step)`	Liste, die durch Einfügen von `step` als letztes Element in `list` entsteht
	`IsOftenLemma(step)`	Realisiert das Kriterium aus 4.1
	`IsImportantLemma(step)`	Realisiert das Kriterium aus 4.2
	`IsTreeLemma(step)`	Realisiert die Suche nach Lemmata mit isolierten Teilbeweisen (4.3)
	`IsSyntaxLemma(step)`	Realisiert die syntaktischen Kriterien aus 4.4
	`IsCompletionLemma(step)`	Realisiert die Bewertung nach gewichteten Inferenzschritten (4.5)
	`IsPartLemma(step)`	Realisiert die Unterteilung langer Beweisketten aus 4.6
	`IsNoLemma(step)`	Realisiert die negativen Kriterien aus 4.7

Anmerkung: Es ist `IsNoLemma(step)` = TRUE, wenn der Schritt nicht als Lemma geeignet ist, sonst FALSE. Die anderen Lemma-Funktionen sind jeweils TRUE, wenn der Schritt als Lemma geeignet ist, FALSE sonst. Die für die Funktionen notwendigen Konstanten sind nicht extra als Parameter des Algorithmus aufgeführt. Man beachte, daß sich durch geeignete Wahl dieser Konstanten erreichen läßt, daß `IsNoLemma(step)` = FALSE immer erfüllt ist.

```
IF iterate = TRUE THEN
  store := in;
  WHILE NOTEMPTY(criteria)
    Islemma := FIRST(criteria);
    criteria := EXCEPTFIRST(criteria);
    WHILE NOTEMPTY(store)
      step := FIRST(store);
      store := EXCEPTFIRST(store);
      IF NOT(IsNoLemma(step)) THEN
        IF IsLemma(step) THEN TYPE(step) := tes-lemma;
      out := APPEND(out,step);
    ENDWHILE
    store := out;
  ENDWHILE
ELSE
  WHILE NOTEMPTY(in)
    step := FIRST(in);
    in := EXCEPTFIRST(in);
    IF NOT(IsNoLemma(step)) THEN
      FOREACH IsLemma ∈ criteria
        IF IsLemma(step) THEN TYPE(step) := tes-lemma;
    ENDIF
    out := APPEND(out,step);
  ENDWHILE
ENDIF
```

4.10 Bewertung der verschiedenen Kriterien für Lemmata

Bei der Bewertung der verschiedenen Kriterien für Lemmata wirkt sich der Mangel an objektiven Bewertungsmaßtäben in gleichem Maße wie bei der Suche nach Lemmata aus. Diese Bewertung muß also notwendigerweise relativ subjektiv bleiben. PCL-Protokolle eignen sich nur sehr begrenzt zur Untersuchung der Lemmata durch Menschen. Deshalb wurden für die Untersuchung die nach Kapitel 5 transformierten Beweise verwendet. Dabei handelt es sich um eine Darstellung der Beweise, die sich im wesentlichen an der menschlichen Gewohnheit orientiert.

Neben dem rein subjektiven Eindruck wurde die Länge der Teilbeweise und die Größe der in diesen Teilbeweisen vorkommenden Terme für die Bewertung verwendet. Diese Kriterien liefern allerdings nur sehr schwache Aussagen über die Qualität des Beweises.

Die Auszeichnung von häufig verwendeten Schritten als Lemmata (4.1) liefert Lemmata, die für den Beweis von relativ hoher Bedeutung sind. Oft kann mit diesen Lemmata auch eine anschauliche Bedeutung verbunden werden. Allerdings treten bei diesem Kriterium extreme Unterschiede in den Längen der Teilbeweise auf. Während die Terme in den Lemmata selbst im allgemeinen relativ kurz sind, treten in den Beweisketten Terme auf, die nicht mehr überschaubar sind. Dieses Kriterium ist als einzelnes nicht für die Suche nach Lemmata geeignet, zusammen mit einem anderen Kriterium, daß die Länge der Teilbeweise begrenzt, liefert es brauchbare Ergebnisse.

Die Suche nach "wichtigen" Schritten (4.2) stellt eines der stärksten Einzelkriterien für die Suche nach Lemmata dar. Die gefundenen Lemmata sind oft von guter Qualität, die Längen der Teilbeweise schwanken zwar stark, es treten jedoch keine überlangen Ketten auf. Mit diesem Kriterium ist es auch ohne die Verwendung anderer Kriterien möglich, den Beweis in verständlicher Weise zu gliedern.

Isolierte Teilbeweise nach 4.3 liefern hervorragende Lemmata. Dabei treten jedoch Beweisketten in jeder Länge auf, insbesondere folgen oft mehrere kurze Lemmata aufeinander, während auf der anderen Seite sehr lange Teilbeweise auftreten können. In Kombination mit Kriterien, die die Beweislänge begrenzen (etwa die Kriterien aus 4.2 oder 4.6) und eventuell Anforderungen an die Mindestlänge der Beweise stellen, ergibt sich die intuitiv "beste" Gliederung, die mit den vorliegenden Werkzeugen bisher erzielt wurde.

Es zeigte sich im Verlauf dieser Arbeit, daß zumindest die "einfachen" syntaktischen Kriterien aus 4.4, die die Länge und damit die Komplexität der beiden Seiten des Faktums berücksichtigen und auf verschiedene Arten vergleichen, keine brauchbaren Ergebnisse liefern. Auch in Kombination mit globalen Kriterien bringen sie keine sichtbare Verbesserung, in der Regel gehen sogar "intuitiv wichtige" Lemmata verloren und werden durch weniger wichtige ersetzt. Die Lemmata selbst wirken zwar gut lesbar, tragen in der Regel aber nicht überdurchschnittlich zum Beweis bei. Insbesondere gegen Ende des Beweises nehmen die Längen der Teilbeweise und der in diesen vorkommenden Terme stark zu, eine vernünftige Gliederung ist nur noch selten zu entdecken.

Die Lemmasuche nach Analyse der verwendeten Inferenzschritte liefert Lemmata mittlerer Qualität. Dabei halten sich aber sowohl Schwankungen in der Länge der Teilbeweise wie in der Größe der in diesen vorkommenden Terme in engen Grenzen. Insgesamt ist dieses Kriterium noch geeignet, den Beweis zu gliedern. Eine interessante Weiterentwicklung dürfte die Verwendung von `IWEIGHT(step)` anstelle von `CHAINLEN(step)` sein. Die auf diesen Funktionen basierenden Kriterien verhalten sich im wesentlichen ähnlich, dabei wirken die Ergebnisse nach 4.5 aber insgesammt etwas ausgewogener.

Lemmata, die nur zur Vermeidung überlanger Beweisketten dienen, eignen sich nur in geringen Ausmaß für eine Gliederung des Beweises. Durch sie unterteilte Beweise sind zwar besser lesbar als monolithische Beweise, insgesamt handelt es sich hier aber um das wohl schwächste globale Kriterium. Die Längen der Teilbeweise unterliegt natürlich nur geringen Schwankungen, dafür kommt es bei den in der Größe der Terme in diesen Teilbeweisen zu relativ großen Schwankungen. Dieses Kriterium ist also nicht besonders geeignet, Beweise zu unterteilen, es läßt sich aber leicht mit anderen Kriterien kombinieren. In Kombination mit der Suche nach isolierten Teilbeweisen oder häufig verwendeten Fakten ergeben sich gute Ergebnisse.

Bei der Untersuchungen von Menschen unterteilter PCL-Beweise zeigte sich, daß die von den leistungsfähigeren unter den vorgestellten Kriterien vorgenommene Unterteilung in der Regel zu besseren Ergebnissen führt. Damit scheinen die untersuchten Kriterien insgesamt ausreichend stark zu sein.

5 Beschreibung von Gleichheitsbeweisen

Im Gegensatz zu den durch Anwendung von Inferenzschritten gewonnenen Mengen von Gleichungen, die letztlich den von einem vervollständigungsbasierten Beweiser gefundenen Gleichheitsbeweis beschreiben, verwenden Menschen nur wenige neue Gleichungen in einem Beweis. Statt dessen bauen sie Gleichheitsketten, also Ketten von Termen, die durch die Anwendung von Gleichungen ineinander überführt werden können. Schon die Prinzipien unterscheiden sich also erheblich: Während von Menschen geführte Beweise auf einer Umformung von Termen basieren, arbeiten automatische Beweiser im wesentlichen auf Gleichungen.

Der vom automatischen Beweiser gefundene Beweis wird dabei über sehr viele kleine Teilziele geführt, die in keinem erkennbaren Zusammenhang stehen und in weitgehend zufälliger Reihenfolge auftauchen. Obwohl die einzelnen Schritte leicht als korrekt erkennbar sind, geht die Übersicht über die Struktur des Beweises verloren. Die Axiome tauchen nur selten auf und ihre Anwendung auf die Terme ist nicht mehr zu erkennen.

Um automatisch generierte Beweise in einer der menschlichen Gewohnheit angepaßten Form darzustellen, ist es nötig, eine Darstellung für Gleichheitsbeweise zu finden, die dieser Gewohnheit weit näher steht und aus der sich übersichtliche und verständliche Beweise erzeugen lassen. Hierzu dient das Konzept der *begründeten Gleichung.*

5.1 Begründete Gleichungen

Eine Begründung für eine Gleichung beschreibt deren Herleitung durch die Anwendung anderer Gleichungen auf einen Term. Eine Gleichung wird also nicht als Konsequenz einer Menge von Gleichungen betrachtet, sondern als Ergebnis der *Anwendung* dieser Gleichungen auf einen Term. Obwohl der zu Grunde liegende Kalkül unverändert bleibt, ändert sich die Sichtweise.

Definition: Begründung für eine Gleichung

1) Ein Tupel $(\mathtt{u}, (\mathtt{s{=}t}, p, \sigma), \mathtt{v})$ heißt Begründung für die Gleichung $\mathtt{u{=}v}$, falls $\mathtt{u}|_p = \sigma(\mathtt{s})$ und $\mathtt{u}[p \leftarrow \sigma(\mathtt{t})] = \mathtt{v}$.
2) Ein Tupel $(\mathtt{u}, (\overline{\mathtt{s{=}t}}, p, \sigma), \mathtt{v})$ heißt Begründung für eine Gleichung $\mathtt{u{=}v}$, falls $\mathtt{u}|_p = \sigma(\mathtt{t})$ und $\mathtt{u}[p \leftarrow \sigma(\mathtt{s})] = \mathtt{v}$.
3) Ist $(\mathtt{s}, B_1, \mathtt{t})$ eine Begründung für $\mathtt{s{=}t}$ und $(\mathtt{t}, B_2, \mathtt{u})$ eine Begründung für $\mathtt{t{=}u}$, dann ist $(\mathtt{s}, B_1, \mathtt{t})\bullet(\mathtt{t}, B_2, \mathtt{u}) \equiv (\mathtt{s}, B_1, \mathtt{t}, B_2, \mathtt{u})$ eine Begründung für $\mathtt{s{=}u}$.

Eine Begründung ist damit im allgemeinen eine Kette der Form

$(\mathtt{u_0}, (\mathtt{s_1} \doteq \mathtt{t_1}, p_1, \sigma_1), \mathtt{u_1}, (\mathtt{s_2} \doteq \mathtt{t_2}, p_2, \sigma_2), \mathtt{u_2}, \ldots, \mathtt{u_{n-1}}, (\mathtt{s_n} \doteq \mathtt{t_n}, p_n, \sigma_n), \mathtt{u_n})$.

Dabei steht $\mathtt{s} \doteq \mathtt{t}$ für $\mathtt{s{=}t}$ oder $\overline{\mathtt{s{=}t}}$ und es gilt $\overline{\overline{\mathtt{s{=}t}}} = \mathtt{s{=}t}$.

Auf den so erklärten Begründungen lassen sich neben der direkten Verkettung weitere Operationen angeben, mit deren Hilfe sich aus vorhandenen Begründungen neue erzeugen lassen. So kann man sich die Symmetrie der Gleichheit zunutze machen. Sowohl eine Gleichung wie die Anwendung einer Gleichung sind symmetrisch, aus einer Begründung für $\mathtt{s{=}t}$ läßt sich also sehr einfach eine Begründung für $\mathtt{t{=}s}$ gewinnen. Die folgende Definition formalisiert diese Aussage.

Definition: Die Symetrieoperation auf Begründungen

Der Symmetrieoperator $\overline{}$ ist wie folgt definiert:

1) $\overline{(\mathtt{u}, (\mathtt{s} \doteq \mathtt{t}, p, \sigma), \mathtt{v})} = (\mathtt{v},\ (\overline{\mathtt{s} \doteq \mathtt{t}},\ p,\ \sigma),\ \mathtt{u})$
2) $\overline{(\mathtt{u}, B_1, \mathtt{t}, B_2, \mathtt{v})} = \overline{(\mathtt{t}, B_2, \mathtt{v})} \bullet \overline{(\mathtt{u}, B_1, \mathtt{t})}$

Wiederholte Anwendung von 1) und 2) liefert

$$\overline{(\mathtt{u_0}, (\mathtt{s_1} \doteq \mathtt{t_1}, p_1, \sigma_1), \mathtt{u_1}, (\mathtt{s_2} \doteq \mathtt{t_2}, p_2, \sigma_2), \mathtt{u_2}, \ldots, \mathtt{u_{n-1}}, (\mathtt{s_n} \doteq \mathtt{t_n}, p_n, \sigma_n), \mathtt{u_n})} =$$

$$(\mathtt{u_n}, (\overline{\mathtt{s_n} \doteq \mathtt{t_n}}, p_n, \sigma_n), \mathtt{u_{n-1}}, \ldots, \mathtt{u_2}, (\overline{\mathtt{s_2} \doteq \mathtt{t_2}}, p_2, \sigma_2), \mathtt{u_1}, (\overline{\mathtt{s_1} \doteq \mathtt{t_1}}, p_1, \sigma_1), \mathtt{u_0}).$$

Mit diesem Operator können Begründungen und damit auch die durch sie begründeten Gleichungen umgedreht werden. Die Korrektheit der Transformation garantiert der folgende Satz.

Satz: Symmetrie von Begründungen

Ist B eine Begründung für `s=t`, so ist $\overline{B}$ eine Begründung für `t=s`.

Wird eine begründete Gleichung auf einen Term angewandt, entsteht eine neue begründete Gleichung. Dabei kann man das Auftreten der ursprünglichen Gleichung in der neuen Begründung vermeiden, da die durch sie ausgedrückte Gleichheit ja bereits durch ihre Begründung beschrieben wird. Man kann also die Anwendung der Gleichung durch die Anwendung der in ihrer Begründung auftretenden Gleichungen ersetzen. Die folgende Definition beschreibt diesen Vorgang.

Definition: Der Verflachungsoperator S

Der Verflachungsoperator S ist definiert wie folgt:

1) $\mathrm{S}(\mathtt{w},\ q,\ \tau,\ (\mathtt{u},\ (\mathtt{s} \doteq \mathtt{t},\ p,\ \sigma),\ \mathtt{v})) = (\mathtt{w}[q{\leftarrow}\tau(\mathrm{u})],\ (\mathtt{s} \doteq \mathtt{t},\ q.p,\ \tau\circ\sigma),\ \mathtt{w}[q{\leftarrow}\tau(\mathrm{v})])$
2) $\mathrm{S}(\mathtt{w},\ q,\ \tau,\ (\mathtt{s},\ B_1,\ \mathtt{t},\ B_2,\ \mathtt{u})) = \mathrm{S}(\mathtt{w},\ q,\ \tau,\ (\mathtt{s},\ B_1,\ \mathtt{t}))\bullet\ \mathrm{S}(\mathtt{w},\ q,\ \tau,\ (\mathtt{t},\ B_2,\ \mathtt{u}))$

Das sich mit Hilfe dieses Operators tatsächlich begründete Gleichungen durch ihre Begründungen ersetzen lassen, zeigt der nächste Satz.

Satz: Schachtelung von Begründungen

1) Ist
   $(\mathtt{u},\ B_1,\ \mathtt{s'},\ (\mathtt{s{=}t},\ p,\ \sigma),\ \mathtt{t'},\ B_2,\ \mathtt{v})$
   eine Begründung für die Gleichung `u=v` und ist B eine Begründung für `s=t`, so ist
   $(\mathtt{u},\ B_1,\ \mathrm{S}(\mathtt{s'},\ p,\ \sigma,\ B),\ B_2,\ \mathtt{v})$
   eine andere Begründung für `u=v`.
2) Ist
   $(\mathtt{u},\ B_1,\ \mathtt{s'},\ (\overline{\mathtt{s{=}t}},\ p,\ \sigma),\ \mathtt{t'},\ B_2,\ \mathtt{v})$
   eine Begründung für die Gleichung `u=v` und ist B eine Begründung für `s=t`, so ist
   $(\mathtt{u},\ B_1,\ \mathrm{S}(\mathtt{s'},\ p,\ \sigma,\ \overline{B}),\ B_2,\ \mathtt{v})$
   eine andere Begründung für `u=v`.

Das folgende Beispiel erläutert diesen Satz.

Beispiel:

Sei $(\mathtt{g(a,b)}, (\mathtt{g(x,b)=g(x,c)}, \lambda, \{\mathtt{x} \leftarrow \mathtt{a}\}), \mathtt{g(a,c)})$ eine Begründung für `g(a,b)=g(a,c)` und sei $(\mathtt{g(x,b)}, (\mathtt{b=c}, \mathit{2}, \{\}), \mathtt{g(x,c)})$ eine Begründung für `g(x,b)=g(x,c)`.
Dann ist
$\mathtt{S}(\mathtt{g(a,b)}, \lambda, \{\mathtt{x} \leftarrow \mathtt{a}\}, (\mathtt{g(x,b)}, (\mathtt{b=d}, \mathit{2}, \{\}), \mathtt{g(x,c)}))$
$= (\mathtt{g(a,b)}[\lambda \leftarrow \{\mathtt{x} \leftarrow \mathtt{a}\}(\mathtt{g(x,b)})], (\mathtt{b=d}, \lambda.2, \{\mathtt{x} \leftarrow \mathtt{a}\} \circ \{\}), \mathtt{g(a,b)}[\lambda \leftarrow \{\mathtt{x} \leftarrow \mathtt{a}\}(\mathtt{g(x,c)})])$
$\equiv (\mathtt{g(a,b)}, (\mathtt{b=c}, \mathit{2}, \{\mathtt{x} \leftarrow \mathtt{a}\}), \mathtt{g(a,c)})$
eine alternative Begründung für `g(a,b)=g(a,c)`. Es wird also die Anwendung der Gleichung `g(x,b)=g(x,c)` zurückgeführt auf die Anwendung der Gleichung `b=c`.

5.2 Axiomatisierte Gleichungsmengen und Begründungen

Wir beschäftigen uns im allgemeinen mit Gleichungen, die unter Annahme eines Axiomensystems gelten. Eine Gleichung ist dann gültig, wenn ihre beiden Seiten durch Anwendung der Axiome ineinander überführt werden können. Die oben definierten Begründungen beschreiben die Anwendung von Gleichungen, also ist eine Gleichung, in deren Begründung nur Axiome und bereits bewiesene Gleichungen auftauchen, ebenfalls gültig.

Definition: In E begründete Gleichung

Sei **E** eine Menge von Gleichungen, **A** eine Menge von Gleichungen mit $=_{\mathrm{A}} \; = \; =_{\mathrm{E}}$ (Die Elemente von **A** heißen *Axiome* von E, in der Praxis gilt im allgemeinen **A**=**E**). `s=t` heißt *in* **E** *begründete Gleichung*, falls eine der folgenden Bedingungen zutrifft.

1) $\mathtt{s=t} \in \mathbf{A}$. $B \equiv (\mathtt{s},\ (\mathtt{s=t},\ \lambda,\ \sigma_{id}),\ \mathtt{t})$ heißt *Begründung* für `s=t`.
2) Es existiert eine Begründung $(\mathtt{u_0},\ (\mathtt{s_1} \doteq \mathtt{t_1},\ p_1,\ \sigma_1),\ \mathtt{u_1},\ (\mathtt{s_2} \doteq \mathtt{t_2},\ p_2,\ \sigma_2),\ \mathtt{u_2},\ \ldots,\ \mathtt{u_{n-1}},\ (\mathtt{s_n} \doteq \mathtt{t_n},\ p_n,\ \sigma_n),\ \mathtt{u_n})$ mit den Eigenschaften, daß alle $\mathtt{s_i=t_i}$ *in* **E** *begründete Gleichungen* sind und daß $\mathtt{u_0} \equiv \mathtt{s}$ und $\mathtt{u_n} \equiv \mathtt{t}$ sind.

In obiger Definition werden beliebige in **E** begründete Gleichungen zur Herleitung einer neuen Gleichung zugelassen. In Gleichheitsbeweisen wird man jedoch nur die Axiome und eventuell einige strukturierende Lemmata verwenden.

Definition: Flache Begründungen

Sei **E** und **A** wie oben, sei $\mathbf{L} \subset \mathbf{E}$ die Menge der Lemmata. **L** enthalte nur in **E** begründete Gleichungen. Eine Begründung $(\mathtt{u_0},\ (\mathtt{s_1} \doteq \mathtt{t_1},\ p_1,\ \sigma_1),\ \mathtt{u_1},\ (\mathtt{s_2} \doteq \mathtt{t_2},\ p_2,\ \sigma_2),\ \mathtt{u_2},\ \ldots,\ \mathtt{u_{n-1}},\ (\mathtt{s_n} \doteq \mathtt{t_n},\ p_n,\ \sigma_n),\ \mathtt{u_n})$ heißt *flache Begründung zu* **A** *und* **L** *in* **E**, falls alle $\mathtt{s_i=t_i} \in (\mathbf{L} \cup \mathbf{A})$ sind.

Diese flachen Begründungen reichen aus, um alle in **E** begründeten Gleichungen zu beschreiben.

Satz: Existenz einer flachen Begründung

Seien **E** und **A** wie oben. Sei `s=t` eine in **E** begründete Gleichung. Zu jedem festen **L** existiert eine flache Begründung für `s=t`.

Man beachte, daß obiger Satz auch für leeres **L** gilt.

Die Berechtigung, Beweise durch begründete Gleichungen zu beschreiben, zeigt folgender Satz:

Satz: Begründete Gleichungen und E-Gleichheit

Sei **E** eine Menge von Gleichungen. Es gilt genau dann $\texttt{s=}_{\mathrm{E}}\texttt{t}$, wenn `s=t` eine in **E** begründete Gleichung ist.

5.3 Begründete Gleichungen und PCL-Schritte

Begründungen für eine Gleichung beschreiben ihre Gültigkeit mit den selben Methoden, die auch Menschen für Gleichheitsbeweise verwenden. Die verbleibenden Unterschiede bestehen im wesentlichen aus der Vollständigkeit der Begründung (Menschen abstrahieren häufig von Stellenangaben und Substitutionen) und der Darstellung des Beweises. Diese Unterschiede sind aber ohne Schwierigkeiten durch Projektion der erwünschten Begründungsteile und geeignete Ausgabe der Begründung zu beseitigen.

Leider liegen die von automatischen Beweisern geführten Beweise aber nicht in Form von begründeten Gleichungen, sondern bestenfalls als PCL-Beschreibung vor. In diesem Abschnitt wird daher ein Verfahren angegeben, mit dem aus der PCL-Beschreibung eines Beweises begründete Gleichungen erzeugt werden können.

Folgende Transformationsvorschrift berechnet zu einem PCL-Ausdruck, zu dessen Argumenten bereits flache Begründungen existieren, ebenfalls eine flache Begründung.

Definition: Die Transformationsvorschrift BG *(Begründete Gleichungen)*

Im folgenden stehe $\texttt{s} \simeq \texttt{t}$ für $\texttt{s} \to \texttt{t}$ oder `s=t`, die Funktion `eval` ordne einem PCL-Ausdruck seinen Wert zu (siehe 2.1). Es sei $\texttt{eval(<expr}_1\texttt{>)} = \texttt{s}_1 \simeq \texttt{t}_1$, $\texttt{B}_1$ eine (flache) Begründung zu $\texttt{s}_1 \simeq \texttt{t}_1$, $\texttt{eval(<expr}_2\texttt{>)} = \texttt{s}_2 \simeq \texttt{t}_2$, $\texttt{B}_2$ eine (flache) Begründung zu $\texttt{s}_2 \simeq \texttt{t}_2$. Eingabe für einen Transformationsschritt ist ein PCL-Ausdruck mit Wert $\texttt{s} \simeq \texttt{t}$, Ausgabe ist eine (flache) Begründung für diesen Schritt.

(1) Axiome

$$\frac{\texttt{initial}}{(\texttt{s}, (\texttt{s=t}, \lambda, \sigma_{id}), \texttt{t})}$$

Man vergleiche die Definition von in **E** begründeten Gleichungen (5.2).

(2) Zitate

$$\frac{\texttt{<expr}_1\texttt{>}}{\texttt{B}_1}$$

<expr$_1$> ist hier nur ein Bezeichner, der auf einen andere Schritt verweist.

(3) Orientieren

$$\frac{\texttt{orient(<expr}_1\texttt{>,u)}}{\texttt{B}_1}$$

$$\frac{\texttt{orient(<expr}_1\texttt{>,x)}}{\overline{\texttt{B}_1}}$$

(4) Kritische Paare

$$\frac{\texttt{cp(<expr}_1\texttt{>,L}.p\texttt{,<expr}_2\texttt{>,L)}}{\texttt{S}(\sigma(\texttt{s}_1),p,\sigma,\overline{\texttt{B}_2})\bullet\texttt{S}(\sigma(\texttt{s}_1),\lambda,\sigma,\texttt{B}_1)} \quad \text{mit } \sigma=\texttt{mgu}(\texttt{s}_1|p,\texttt{s}_2).$$

$$\frac{\texttt{cp(<expr}_1\texttt{>,L}.p\texttt{,<expr}_2\texttt{>,R)}}{\texttt{S}(\sigma(\texttt{s}_1),p,\sigma,\texttt{B}_2)\bullet\texttt{S}(\sigma(\texttt{s}_1),\lambda,\sigma,\texttt{B}_1)} \quad \text{mit } \sigma=\texttt{mgu}(\texttt{s}_1|p,\texttt{t}_2).$$

$$\frac{\texttt{cp(<expr}_1\texttt{>,R}.p\texttt{,<expr}_2\texttt{>,L)}}{\texttt{S}(\sigma(\texttt{t}_1),p,\sigma,\overline{\texttt{B}_2})\bullet\texttt{S}(\sigma(\texttt{t}_1),\lambda,\sigma,\overline{\texttt{B}_1})} \quad \text{mit } \sigma=\texttt{mgu}(\texttt{t}_1|p,\texttt{s}_2).$$

$$\frac{\texttt{cp(<expr}_1\texttt{>,R}.p\texttt{,<expr}_2\texttt{>,R)}}{\texttt{S}(\sigma(\texttt{t}_1),p,\sigma,\texttt{B}_2)\bullet\texttt{S}(\sigma(\texttt{t}_1),\lambda,\sigma,\overline{\texttt{B}_1})} \quad \text{mit } \sigma=\texttt{mgu}(\texttt{t}_1|p,\texttt{t}_2).$$

(5) Reduktionen

$$\frac{\texttt{tes-red(<expr}_1\texttt{>,L}.p\texttt{,<expr}_2\texttt{>,L)}}{\texttt{S}(\texttt{s}_1,p,\sigma,\overline{\texttt{B}_2})\bullet\texttt{B}_1} \quad \text{mit } \sigma=\texttt{match}(\texttt{s}_2,\texttt{s}_1|p).$$

$$\frac{\texttt{tes-red(<expr}_1\texttt{>,L}.p\texttt{,<expr}_2\texttt{>,R)}}{\texttt{S}(\texttt{s}_1,p,\sigma,\texttt{B}_2)\bullet\texttt{B}_1} \quad \text{mit } \sigma=\texttt{match}(\texttt{t}_2,\texttt{s}_1|p).$$

$$\frac{\texttt{tes-red(<expr}_1\texttt{>,R}.p\texttt{,<expr}_2\texttt{>,L)}}{\texttt{B}_1\bullet\texttt{S}(\texttt{t}_1,p,\sigma,\texttt{B}_2)} \quad \text{mit } \sigma=\texttt{match}(\texttt{s}_2,\texttt{t}_1|p).$$

$$\frac{\texttt{tes-red(<expr}_1\texttt{>,R}.p\texttt{,<expr}_2\texttt{>,R)}}{\texttt{B}_1\bullet\texttt{S}(\texttt{t}_1,p,\sigma,\overline{\texttt{B}_2})} \quad \text{mit } \sigma=\texttt{match}(\texttt{t}_2,\texttt{t}_1|p).$$

(6) Instanziierungen

$$\frac{\texttt{instance(<expr}_1\texttt{>,<expr}_2\texttt{>)}}{\texttt{S}(\texttt{s}_1,\lambda,\sigma,\texttt{B}_2)} \quad \text{mit } \sigma(\texttt{s}_2)=\texttt{s} \text{ und } \sigma(\texttt{t}_2)=\texttt{t}.$$

Das Transformationssystem **BG** erlaubt es also, zu den im Laufe einer Vervollständigung entstehenden Gleichungen Begründungen anzugeben.

Satz: Korrektheit von BG

Die Transformationsvorschrift **BG** ist korrekt.

Bisher können mit **BG** nur konstruktiv hergeleitete Gleichungen behandelt werden. Häufig werden gerade die Beweisziele aber destruktiv, also durch Reduktion der beiden Seiten auf eine gemeinsame Normalform, bewiesen. Der folgende Satz erlaubt die Angabe von Begründungen für solche Gleichungen. Dabei ist in der praktischen Anwendung `u=v` das zu beweisende Ziel.

Satz: Begründungen für destruktiv bewiesene Gleichungen

- Sei `B` eine Begründung für die triviale Gleichung `s=s`, in der die Gleichung `u=v` genau ein Mal mit der leeren Substitution und an der leeren Stelle auftaucht, also $\mathtt{B} \equiv (\mathtt{s}, \mathtt{B}_1, \mathtt{u}, (\mathtt{u=v}, \lambda, \sigma_{\mathtt{id}}), \mathtt{v}, \mathtt{B}_2, \mathtt{s})$. Dann ist $\overline{(\mathtt{s}, \mathtt{B}_1, \mathtt{u})} \bullet \overline{(\mathtt{v}, \mathtt{B}_2, \mathtt{s})}$ eine Begründung für `u=v`.
- Sei `B` eine Begründung für eine Gleichung `s=t`, in der `u=v` nicht auftaucht, sei `B'` eine andere Begründung für `s=t`, in der die Gleichung `u=v` genau ein Mal mit der leeren Substitution und an der leeren Stelle auftaucht, also $\mathtt{B'} \equiv (\mathtt{s}, \mathtt{B}_1, \mathtt{u}, (\mathtt{u=v}, \lambda, \sigma_{\mathtt{id}}), \mathtt{v}, \mathtt{B}_2, \mathtt{t})$. Dann ist $\overline{(\mathtt{s}, \mathtt{B}_1, \mathtt{u})} \bullet \mathtt{B} \bullet \overline{(\mathtt{v}, \mathtt{B}_2, \mathtt{t})}$ eine Begründung für `u=v`.

Wenn man die zu beweisenden Ziele also wie Axiome behandelt, kann man bei geeigneten Strategien auch Begründungen für destruktiv bewiesene oder überdeckte Gleichungen berechnen. Man beachte, daß in diesem Fall Begründungen entstehen, die keine korrekten Begründungen in E sind. Erst nachträglich werden die korrekten Begründungen ermittelt. Um dieses zu ermöglichen wird die Transformationsvorschrift **BG** um eine Regel erweitert.

Definition: Die Transformationsvorschrift BG'

Seien die Voraussetzungen wie in der Definition von **BG**. Das System **BG'** besteht aus den Regeln in **BG** und der folgenden neuen Regel.

(1') Hypothesen

$$\frac{\mathtt{hypothesis}}{(\mathtt{s}, (\mathtt{s=t}, \lambda, \sigma_{id}), \mathtt{t})}$$

Mit dem System **BG'** lassen sich Begründungen für alle in einer PCL-Beschreibung vorkommenden Fakten angeben. Die Methoden eines vervollständigungsbasierten Beweisers (1.5) garantieren, daß nur Begründungen in E und Begründungen, die die Bedingungen aus dem letzten Satz erfüllen, erzeugt werden. Dies illustriert das folgende Beispiel. Es verwendet das kurze PCL-Protokoll von Seite 19.

Beispiel: In diesem kurzen PCL-Protokoll wird ein Beweisziel in zwei Schritten auf eine triviale Gleichung reduziert und damit bewiesen. Hier werden die PCL-Schritte durch die Ergebnisse der Anwendung von **BG'**-Regeln ergänzt.

```
0 : tes-eqn : f(e(),x) = x : initial
```

$\mathtt{B}_0 \equiv (\mathrm{f(e(),x)}, (\mathtt{f(e(),x)=x}, \lambda, \{\}), \mathrm{x})$ ist eine Begründung für Schritt 0 nach Regel (1).

```
3 : tes-goal : g(f(e(),x)) = f(e(),g(x)) : hypothesis
```

$\mathtt{B}_3 \equiv (\mathtt{g}(\mathtt{f}(\mathtt{e}(),\mathtt{x})),(\mathtt{g(f(e(),x))=f(e(),g(x))},\lambda,\{\}),\mathtt{f}(\mathtt{e}(),\mathtt{g}(\mathtt{x})))$ nach Regel (1').

```
4 : tes-rule : f(e(),x) -> x : orient(0,u)
```

$\mathtt{B}_4 = \mathtt{B}_0 \equiv (\mathtt{f}(\mathtt{e}(),\mathtt{x}),(\mathtt{f(e(),x)=x},\lambda,\{\}),\mathtt{x})$ nach Regel (2), 1.Fall.

```
5 : tes-goal : g(f(e(),x)) = g(x) : tes-red(3,R,4,L)
```

$\mathtt{B}_5 = \mathtt{B}_3 \bullet \mathtt{S}(\mathtt{f}(\mathtt{e}(),\mathtt{g}(\mathtt{x})),\lambda,\{\mathtt{x} \leftarrow \mathtt{g}(\mathtt{x})\},\mathtt{B}_4)$
$\equiv (\mathtt{g}(\mathtt{f}(\mathtt{e}(),\mathtt{x})),(\mathtt{g(f(e(),x))=f(e(),g(x))},\lambda,\{\}),\mathtt{f}(\mathtt{e}(),\mathtt{g}(\mathtt{x})),(\mathtt{f(e(),x)=x},\lambda,\{\mathtt{x} \leftarrow \mathtt{g}(\mathtt{x})\}),\mathtt{g}(\mathtt{x}))$
nach Regel (5), 3. Fall

```
6 : tes-final : g(x) = g(x) : tes-red(5,L.1,4,L)
```

$\mathtt{B}_6 = \mathtt{S}(\mathtt{g}(\mathtt{f}(\mathtt{e}(),\mathtt{x})),1,\{\},\overline{\mathtt{B}_4}) \bullet \mathtt{B}_5$
$\equiv (\mathtt{g}(\mathtt{x}),(\overline{\mathtt{f(e(),x)=x}},1,\{\}),\mathtt{g}(\mathtt{f}(\mathtt{e}(),\mathtt{x})))$
$\bullet(\mathtt{g}(\mathtt{f}(\mathtt{e}(),\mathtt{x})),(\mathtt{g(f(e(),x))=f(e(),g(x))},\lambda,\{\}),\mathtt{f}(\mathtt{e}(),\mathtt{g}(\mathtt{x})),(\mathtt{f(e(),x)=x},\lambda,\{\mathtt{x} \leftarrow \mathtt{g}(\mathtt{x})\}),\mathtt{g}(\mathtt{x}))$
nach Regel (5), 1. Fall

Damit ist $\mathtt{B}_6$ eine Begründung für die triviale Gleichung `g(x)=g(x)`, in der das ursprüngliche Beweisziel `g(f(e(),x))=f(e(),g(x))` an der leeren Stelle und mit der leeren Substitution vorkommt.

Diese Begründung erfüllt die Bedingung aus dem letzten Satz, man kann also aus ihr eine Begründung für `g(f(e(),x))=f(e(),g(x))` gewinnen. Es ist
$\overline{(\mathtt{g}(\mathtt{x}),(\overline{\mathtt{f(e(),x)=x}},1,\{\}),\mathtt{g}(\mathtt{f}(\mathtt{e}(),\mathtt{x})))} \bullet \overline{(\mathtt{f}(\mathtt{e}(),\mathtt{g}(\mathtt{x})),(\mathtt{f(e(),x)=x},\lambda,\{\mathtt{x} \leftarrow \mathtt{g}(\mathtt{x})\}),\mathtt{g}(\mathtt{x}))}$
$\equiv (\mathtt{g}(\mathtt{f}(\mathtt{e}(),\mathtt{x})),(\mathtt{f(e(),x)=x},1,\{\}),\mathtt{g}(\mathtt{x}),(\overline{\mathtt{f(e(),x)=x}},\lambda,\{\}),\mathtt{f}(\mathtt{e}(),\mathtt{g}(\mathtt{x})))$
eine Begründung für dieses Ziel.

5.4 Erzeugung von Begründungen zu Beweisen in PCL

Der im folgenden vorgestellte Algorithmus berechnet zu den Beweiszielen und Lemmata eines PCL-Protokolls gültige Begründungen, in denen nur die Axiome und bewiesene Lemmata auftauchen. Dabei können konstruktiv erzeugte Fakten ebenso als Beweisziele auftreten wie initiale Hypothesen, es können also auch reine Vervollständigungen ohne vorgegebene Beweisziele bearbeitet werden.

Zuerst werden zwei rekursive Funktion angeben, die zu einem beliebigen PCL-Schritt oder -Ausdruck unter Verwendung von **BG'** eine Begründung berechnet.

Definition: Die Funktionen BEG und LBEG

Sei step ein PCL-Schritt und expr ein PCL-Ausdruck.

- Die Funktion BEG ist definiert wie folgt:
  - $\mathtt{BEG(step)} = \mathtt{BEG(EXPR(step))}$
  - $\mathtt{BEG(expr)}$ berechnet sich nach folgener Vorschrift:
    Setze $\mathtt{B_i} = \mathtt{BEG(expr_i)}$ und $(\mathtt{s_i}, \mathtt{t_i}) = \mathtt{eval(expr_i)}$ für alle Ausdrücke, die Argumente von expr sind. Verwende die $\mathtt{expr_i}$ und $(\mathtt{s_i}, \mathtt{t_i})$ als Eingabe für den zu expr gehörenden Transformationsschritt aus **BG'**. Sei B das Ergebnis dieses Transformationsschrittes, dann ist $\mathtt{BEG(expr)} = \mathtt{B}$.
- Die Funktion LBEG ist im wesentlichen analog definiert, verwendet jedoch eventuelle Lemmata, um die erzeugte Begründung abzukürzen. Es gilt:
  - $$\mathtt{LBEG(step)} = \begin{cases} \mathtt{LBEG(EXPR(step))} & \text{falls } \mathtt{TYPE(step)} \neq \mathtt{tes\text{-}lemma} \\ (\mathtt{s}, (\mathtt{s} = \mathtt{t}, \lambda, \sigma_{\mathtt{id}}), \mathtt{t}) & \text{sonst} \end{cases}$$
    mit $\mathtt{FACT(step)} = \mathtt{s} \simeq \mathtt{t}$.
  - $\mathtt{LBEG(expr)}$ berechnet sich nach folgener Vorschrift:
    Setze $\mathtt{B_i} = \mathtt{LBEG(expr_i)}$ und $(\mathtt{s_i}, \mathtt{t_i}) = \mathtt{eval(expr_i)}$ für alle Ausdrücke, die Argumente von expr sind. Verwende die $\mathtt{expr_i}$ und $(\mathtt{s_i}, \mathtt{t_i})$ als Eingabe für den zu expr gehörenden Transformationsschrittes aus **BG'**. Sei B das Ergebnis dieses Transformationsschrittes, dann ist $\mathtt{LBEG(expr)} = \mathtt{B}$.

Neben BEG und LBEG benötigt man zwei weitere Hilfsfunktionen, die die Bearbeitung von destruktiv bewiesenen Gleichungen ermöglichen. Sie orientieren sich an dem Satz über Beweise zu solchen Gleichungen (Seite 52).

Definition: Die Funktionen UsesHypothesis und SplitChain

Die Funktion UsesHypothesis stellt fest, ob in einer Begründung eine Hypothese an der leeren Stelle und mit der leeren Substitution vorkommt. Ist dies der Fall, so zerlegt die Funktion SplitChain die Begründung. Genauer:

- $\mathtt{UsesHypothesis(B, list)} = \mathtt{TRUE}$ gilt genau dann, wenn $\mathtt{FACT(step)}$ in B an der leeren Stelle mit der leeren Substitution vorkommt für ein step aus list mit $\mathtt{EXPR(step)} = \mathtt{hypothesis}$, sonst gilt $\mathtt{UsesHypothesis(B, list)} = \mathtt{FALSE}$.
- Ist $\mathtt{UsesHypothesis(B, list)} = \mathtt{TRUE}$, so hat B die folgende Form:
  $(\mathtt{s}, \mathtt{B_1}, \mathtt{u}, (\mathtt{u} = \mathtt{v}, \lambda, \sigma_{\mathtt{id}}), \mathtt{v}, \mathtt{B_2}, \mathtt{t})$
  Dabei ist u=v Fakt eines PCL-Schrittes mit $\mathtt{EXPR(step)} = \mathtt{hypothesis}$. In diesem Fall gilt $\mathtt{SplitChain(B, list)} = (\mathtt{u{=}v}, (\mathtt{s}, \mathtt{B_1}, \mathtt{u}), (\mathtt{v}, \mathtt{B_2}, \mathtt{t}))$, sonst ist $\mathtt{SplitChain(B, list)}$ undefiniert.

Mit diesen Funktionen läßt sich ein Algorithmus formulieren, der zu einem PCL-Protokoll eine Liste von begründeten Gleichungen berechnet. Diese Liste hat folgende Eigenschaften:

- Alle Begründungen aus der Liste verwenden nur frühere Gleichungen und Fakten von PCL-Schritten vom Typ `initial`. Setzt man also $\mathbf{A} = \{\texttt{FACT}(\texttt{step})|\texttt{EXPR}(\texttt{step}) = \texttt{initial}\}$ und $\mathbf{L} = \{\texttt{FACT}(\texttt{step})|\texttt{TYPE}(\texttt{step}) = \texttt{tes-lemma}\}$, so sind die Begründungen flache Begründungen zu **A** und **L**.
- Zu jedem PCL-Schritt `step` mit $\texttt{TYPE}(\texttt{step}) = \texttt{tes-lemma}$ ist $\texttt{FACT}(\texttt{step})$ mit einer Begründung in der Liste enthalten.
- Zu jedem PCL-Schritt `step` mit $\texttt{TYPE}(\texttt{step}) = \texttt{tes-final}$, der den Beweis eines ursprünglichen Zieles abschließt, ist dieses Ziel mit einer Begründung in der Liste vertreten.
- Jeder andere PCL-Schritt `step` mit $\texttt{TYPE}(\texttt{step}) = \texttt{tes-final}$ ist mit seinem Fakt und einer Begründung für dieses Fakt in der Liste vertreten.

Damit repräsentiert die erzeugte Liste einen gegliederten Beweis für die Hypothesen und eventuelle neue Ergebnisse.

Eingabe:	`in`	Eine Liste von PCL-Schritten
Ausgabe:	`out`	Eine Liste von begründeten Gleichungen
Variable	`store`	Eine Liste von PCL-Schritten
	`step`	Ein einzelner PCL-Schritt
	`u, v`	Terme
	$\texttt{B}_1, \texttt{B}_2$	Begründungen
	$\texttt{expr}_1, \texttt{expr}_2$	PCL-Ausdrücke
Funktionen:	`NOTEMPTY(list)`	FALSE, falls `list` leer, TRUE sonst
	`FIRST(list)`	Erstes Element von `list`
	`EXCEPTFIRST(list)`	`list` ohne erstes Element
	`APPEND(list,step)`	Liste, die durch Einfügen von `step` als letztes Element in `list` entsteht
	`TYPE(step)`	Typ von `step`
	`TOP(expr)`	Top-Symbol eines PCL-Ausdruckes
	`SUBEXPRS(expr)`	Die Argumente von `expr`, die PCL-Ausdrücke sind

```
store := in;
WHILE NOTEMPTY(in)
  step := FIRST(in);
  in := EXCEPTFIRST(in);
  IF TYPE(step) = tes-lemma THEN
    out := APPEND(out,(FACT(step),LBEG(step)));
  IF TYPE(step) = tes-final THEN
    IF TOP(EXPR(step)) = instance THEN
      Ein (Teil-) Ziel wird durch Überdecken bewiesen
      (expr$_1$,expr$_2$) := SUBEXPRS(EXPR(step));
      (u=v,B$_1$, B$_1$) := SplitChain(LBEG(expr$_1$),store);
      out := APPEND(out,(u=v, $\overline{B_1} \bullet$ LBEG(step) $\bullet \overline{B_2}$));
    ELSE IF NOT(UsesHypothesis(LBEG(step),store)) THEN
      Hier wird ein konstruktives Ergebnis nachträglich zum Theorem
      out := APPEND(out,(FACT(step),LBEG(step)));
    ELSE
      Ein Ziel wird destruktiv bewiesen
      (u=v,B$_1$, B$_1$) := SplitChain(LBEG(step),store);
      out := APPEND(out,(u=v,$\overline{B_1} \bullet \overline{B_2}$);
    ENDIF
  ENDIF
ENDWHILE
```

5.5 Ein kurzer Beweis mit begründeten Gleichungen

Um einen besseren Eindruck von einem solchen Beweis in Form von begründeten Gleichungen zu vermitteln, soll hier für das kurze Beispiel aus 3.3 ein solcher Beweis wiedergegeben werden. Dieser Beweis verwendet zwei Lemmata und präsentiert die Begründungen in einer übersichtlichen Form. Insbesondere werden die in den Begründungen verwendeten Gleichungen nur symbolisch angegeben und die Begründung als Gleichheitskette dargestellt.

Dieser Beweis entstand durch Anwendung des Programmes `proof` auf das extrahierte PCL-Protokoll aus 3.3. `proof` unterstützt auch Darstellungen, die von einigen Informationen abstrahieren, näheres findet sich in 6.5. Die verwendeten Lemmata sind die Schritte mit den Nummern 45 und 27.

```
Consider the following set of axioms:

  Axiom 1: f(e(),x) = x
  Axiom 2: f(x,e()) = x
  Axiom 3: f(g(x),x) = e()
  Axiom 4: f(f(x,y),z) = f(x,f(y,z))

This theorem holds true:

  Theorem 1: g(g(a())) = a()
```

```
Proof:

  Lemma 1: f(g(u),f(u,z)) = z

    f(g(u),f(u,z)) = f(f(g(u),u),z) by Axiom 4 RL at e with {z <- z, y <- u, x <- g(u)}
                   = f(e(),z) by Axiom 3 LR at 1 with {x <- u}
                   = z by Axiom 1 LR at e with {x <- z}

  Lemma 2: y = g(g(y))

    y = f(g(g(y)),f(g(y),y)) by Lemma 1 RL at e with {z <- y, u <- g(y)}
      = f(g(g(y)),e()) by Axiom 3 LR at 2 with {x <- y}
      = g(g(y)) by Axiom 2 LR at e with {x <- g(g(y))}

  Theorem 1: g(g(a())) = a()

    g(g(a())) = a() by Lemma 2 RL at e with {y <- a()}
```

6 Implementierte Programme

Mit dem bisher vorgestellten Konzepten läßt sich die Erzeugung und Bearbeitung von Gleichheitsbeweisen in vier Phasen unterteilen. In der ersten Phase wird der Beweis durch einen vervollständigungsbasierten Beweiser geführt. Der Beweiser hält sein Vorgehen in einem PCL-Protokoll fest. Damit liegt der Beweis in einer Form vor, in der sich alle vom Beweiser durchgeführten Schritte eindeutig nachvollziehen lassen.

Die zweite Phase dient der Extraktion des Beweises. Aus dem PCL-Protokoll werden die für den Beweis nicht benötigten Schritte entfernt. Alle weiteren Analysen können an dieser stark verkürzten Beweisdarstellung vorgenommen werden. Selbst für Protokolle, die in der vollständigen Form nicht mehr beherrschbar sind, ist das verkürzte Protokoll im allgemeinen noch gut überschaubar. Aus dem Vergleich der verwendeten Schritte mit allen durchgeführten Schritten läßt sich die Strategie des Beweisers bewerten und verbessern.

In der dritten Phase werden besonders wichtige Schritte in dem Beweisprotokoll identifiziert und als Lemmata ausgezeichnet. Diese Phase dient vor allem als Vorbereitung für die abschließende vierte Phase, in der das Beweisprotokoll in eine für Menschen geeignete Form transformiert wird. Stellt man die einzelnen Phasen graphisch da, so ergibt sich Abbildung 2.

Diese Unterteilung des Gesamtproblems ist die Grundlage für die Entwicklung der im Rahmen dieser Arbeit implementierten Programme. Die Rolle des Beweisers wird dabei von dem parallelen DISCOUNT-System ([Pi92], [DP92]) ausgefüllt. Dieses System kann bisher keine echten PCL-Protokolle erzeugen, bei sequentiellem Lauf kann es aber ein eindeutiges Protokoll schreiben. Dieses Protokoll wird von dem Programm `trans` in echtes PCL übersetzt.

Die Entfernung der nicht notwendigen Beweisschritte wird von den Programmen `extract`, `mextract` oder `revert` und `rextract` vorgenommen. Diese verschiedenen Programme für die Teilaufgabe der Beweisextraktion erwiesen sich als notwendig, weil die PCL-Protokolle für die vom DISCOUNT-System geführten Beweise wesentlich größer als erwartet ausfielen.

Die Suche nach Lemmata wird von dem Programm `lemma` vorgenommen. Es unterstützt alle in Kapitel 4 vorgestellten Kriterien und erlaubt eine weitgehend freie Kombination dieser Kriterien. Schließlich wird von dem Programm `proof` die in Kapitel 5 vorgestellte Transformation in begründete Gleichungen vorgenommen. Diese werden in Form von Gleichheitsketten ausgegeben und liefern so eine relativ gut verständliche Darstellung des Beweises. Das Zusammenspiel der verschiedenen Programme ist in Abbildung 3 wiedergegeben.

Als Implementierungssprache für die im Rahmen dieser Arbeit erstellten Programme wurde GNU-C gewählt. GNU-C ist eine Implementierung der Programmiersprache C, die im wesentlichen dem ANSI-Standard für C genügt. Der Compiler steht als freie Software auf vielen verschiedenen Systemen zur Verfügung. Damit ist eine hohe Portabilität der entwickelten Programme gegeben. Die Sprache C erlaubt eine sehr genaue Kontrolle über die verwendeten Datenstrukturen und wird von den vorliegenden Compilern in effizienten Code übersetzt.

Um die in der Regel mehrstufige Verarbeitung von PCL-Protokollen zu erleichtern, lassen sich alle entwickelten Programme als UNIX-Filter einsetzen. Sie können also zur Bearbeitung von Dateien dienen, Eingaben von einem Terminal lesen oder in mehrstufigen Pipes verwendet werden. Werden die Programme zur Bearbeitung von Dateien eingesetzt, so werden sie alle durch einen Aufruf in der Kommandozeile einer UNIX-Shell gestartet. Dabei wird der Name des Programmes von einem oder mehreren Dateinamen und eventuellen Optionen gefolgt. Werden mehrere Dateinamen angegeben, so werden die Dateien wie eine zusammenhängende Datei behandelt.

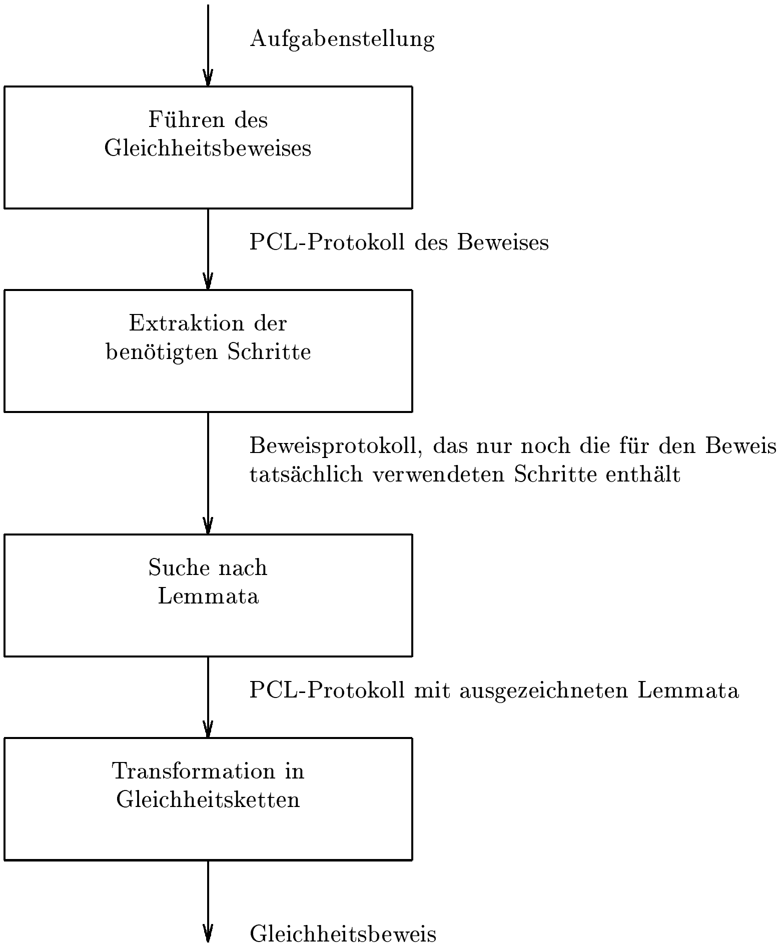

Abbildung 2: Phasen bei der Erzeugung und Aufbereitung von Beweisen

Die entwickelten Programme werden im folgenden näher beschrieben.

6.1 Erzeugung von PCL-Protokollen: `trans`

Wie bereits oben erwähnt wurde, kann das DISCOUNT-System bisher keine PCL-Protokolle erzeugen. Statt dessen kann es bei sequentiellen Läufen ein eindeutiges Protokoll in einem eigenen Format schreiben. Ein kurzes Beispiel für dieses Format befindet sich in Anhang A.1.2. Das Programm `trans` übersetzt dieses Format in echtes PCL. In naher Zukunft soll das DISCOUNT-System echte PCL-Protokolle direkt erzeugen, bei dem heute verwendeten Format handelt es sich daher nur um eine Übergangslösung, die hier nicht näher dokumentiert werden soll.

Neben der direkten Übersetzung erlaubt `trans` auch eine Kommentierung der entstehenden PCL-Protokolle. Dabei wird die Herkunft der PCL-Schritte beschrieben, eine eindeutige Zuordnung zu den im DISCOUNT-Format geschriebenen Schritten ist möglich. Die Ergebnisse der Analyse des PCL-Protokolls sind also auf einfache Art auf den Beweiser übertragbar. `trans` unterstützt die folgenden Optionen:

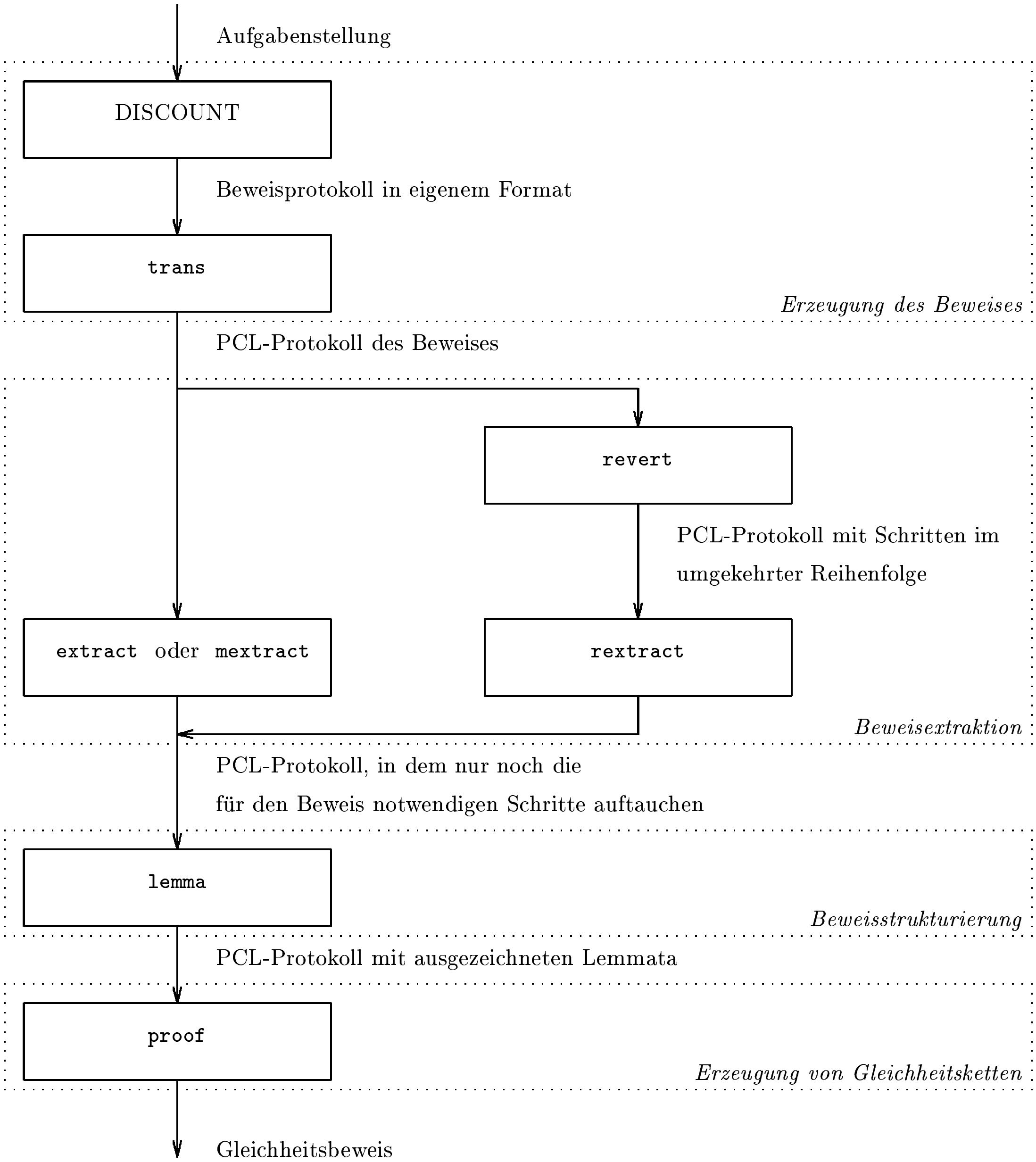

Abbildung 3: Programme für die Erzeugung und Bearbeitung von Beweisen

Option	Bedeutung
–v	Diese Option bewirkt eine Ausgabe der Informationen über das Programm vor Ablauf des Übersetzungsalgorithmus.
–h	Bei Verwendung der Option –h wird das Programm nach Ausgabe der Programminformationen abgebrochen. Eine Kopie der Programminformationen befindet sich in Anhang B.1
–c	Ist diese Option gesetzt, so werden Kommentare in dem DISCOUNT-Protokoll in das entstehende PCL-Protokoll übernommen. Zusätzlich wird die Zuordnung von PCL- und DISCOUNT-Bezeichnern sowie weitere Informationen über das Ausgangsprotokoll in neuen Kommentaren festgehalten.
–o `<outfile>`	Die Option –o dient der Festlegung einer Datei für die Ausgabe des Programmes. Ist diese Option gesetzt, so wird die Ausgabe in die Datei mit Namen `<outfile>` geschrieben, sonst wird sie auf der Standard-Ausgabe ausgegeben.

6.2 Extraktion der Beweisstruktur: `extract` und `mextract`

Das Programm `extract` realisiert den Algorithmus aus 3.1 und entfernt alle nicht benötigten Schritte aus dem eingegebenen PCL-Protokoll. Es verwendet zum Einlesen des Protokolls den für alle mit PCL arbeitenden Programme entwickelten Parser und die auch für `lemma` und `proof` verwendeten Datenstrukturen. Diese repräsentieren die in den PCL-Schritten auftauchenden Terme durch rekursive Strukturen, auf denen alle für die Termersetzung notwendigen Operationen definiert sind. Diese Operationen sind für die weitergehende Beweisanalyse und Transformation notwendig, die Darstellung erfordert aber relativ viel Speicherplatz.

Es zeigte sich sehr bald, daß die Protokolle der vom DISCOUNT-System gefundenen Beweise sehr groß sind. Der Algorithmus aus 3.1 untersucht die einzelnen Schritte des Protokolls von hinten nach vorne. Da die Schritte aber typischerweise in der Reihenfolge ihrer Erzeugung zur Verfügung stehen, muß das gesamte Protokoll zwischengespeichert werden. Damit ist die relativ speicherintensive Darstellung der PCL-Schritte selbst für Beispiele mittlerer Komplexität nicht mehr praktikabel.

Um diese Probleme zu lösen wurde das Programm `mextract` entwickelt. Es verwendet einen eigenen Parser und legt die PCL-Schritte in einem wesentlich kompakteren Format im Speicher ab. Insbesondere werden die Terme nicht durch rekursive Strukturen, sondern durch eine ihrer externen Darstellung entsprechende Zeichenfolge repräsentiert. Damit kann das Programm `mextract` die meisten der bisher vorliegenden Protokolle verarbeiten. Ein Seiteneffekt der kompakteren Darstellung ist auch ein erheblicher Geschwindigkeitsgewinn.

Bis auf die verschiedenen internen Darstellungen des Protokolls und die daraus resultierenden Unterschiede in der Leistungsfähigkeit unterscheiden sich `extract` und `mextract` nicht. Beide stellen im Rahmen ihrer Leistungsfähigkeit die selbe Funktionalität zur Verfügung. Im allgemeinen sollte das effizientere Programm `mextract` verwendet werden.

Die Programme unterstützen die folgenden Optionen:

Option	Bedeutung
–v	Diese Option bewirkt eine Ausgabe von Informationen über das Programm und einzelne Phasen während des Programmlaufs.
–h	Bei Verwendung der Option –h wird das Programm nach Ausgabe der Programminformationen abgebrochen. Eine Kopie der Programminformationen befindet sich in Anhang B.2 beziehungsweise B.3.
–c	Ist diese Option gesetzt, so werden im Eingabeprotokoll enthaltene Kommentare beibehalten, sofern sie PCL-Schritten in der Ausgabe zugeordnet werden können.
–o `<outfile>`	Die Option –o dient der Festlegung einer Datei für die Ausgabe des Programmes. Ist diese Option gesetzt, so wird die Ausgabe in die Datei mit Namen `<outfile>` geschrieben, sonst wird sie auf der Standard-Ausgabe ausgegeben.

6.3 Werkzeuge für sehr große Beweise: `revert` und `rextract`

Mit dem DISCOUNT-System gelang der Beweis einiger besonders anspruchsvoller Probleme. Ein Beispiel dafür ist der Beweis für das 6. Problem aus [LO82], der auch in Anhang A.4 behandelt wird. Das Protokoll für den von DISCOUNT gefundenen Beweis für dieses Problem hat eine Größe von etwa 50 Megabyte. Bei Protokollen dieser Größe ist eine Analyse auch mit `mextract` nicht mehr möglich. Zur Behandlung dieser extremen Fälle dienen die Programme `revert` und `rextract`. `revert` liest ein PCL-Protokoll und gibt die Schritte dieses Protokolles in umgekehrter Reihenfolge zurück. Die Entwicklung dieses Programmes war notwendig, weil die vorhandenen UNIX-Programme zum Umordnen von Dateien mit derart großen Datenmengen nicht angemessen umgehen konnten.

Das Programm `revert` kann prinzipiell Protokolle jeder Größe bearbeiten. Wird die Ausgabe des Programms direkt in einer Pipeline verwendet, so ist die Datenmenge durch den virtuellen Speicher begrenzt. Werden dagegen Dateien bearbeitet, so ist die maximale Größe dieser Dateien nur durch die Größe und Partitionierung der vorhandenen Massenspeicher begrenzt. In diesem Fall generiert `revert` bei Überlauf des virtuellen Speichers mehrere Dateien. Die Verkettung dieser Dateien ergibt dann das endgültige Ergebnis. Die Namen der erzeugten Dateien entstehen durch Verkettung des gewünschten Ausgabenamens mit je einem Suffix der Form `.rev`x, wobei x eine mit der Anzahl der Ausgabedateien wachsende Zahl ist. Ist kein Dateiname angegeben, so wird der Name `stdout` angenommen.

`revert` unterstützt diese Optionen:

Option	Bedeutung
–v	Diese Option bewirkt eine Ausgabe von Informationen über das Programm und einzelne Phasen während des Programmlaufs.
–h	Bei Verwendung der Option –h wird das Programm nach Ausgabe der Programminformationen abgebrochen. Eine Kopie der Programminformationen befindet sich in Anhang B.4.
–o `<outfile>`	Die Option –o dient der Festlegung einer Datei für die Ausgabe des Programmes. Ist diese Option gesetzt, so wird die Ausgabe in die Datei mit Namen `<outfile>` geschrieben, sonst wird sie auf der Standard-Ausgabe ausgegeben. Reicht der virtuelle Speicher für den Ablauf von `revert` nicht aus, so werden Dateien mit Namen der Form `<outfile>.rev*` beziehungsweise `stdout.rev*` geschrieben.

`rextract` dient zur Analyse von mit `revert` vorbehandelten PCL-Protokollen. Da in diesen Protokollen die PCL-Schritte in der für die Analyse notwendigen Reihenfolge vorliegen, ist ein Speichern des gesamten Protokolles wie bei den anderen Extraktionsprogrammen nicht mehr notwendig. Die von `rextract` zu verwaltende Datenmenge ist proportional zum Umfang des extrahierten Protokolles. Dieser Umfang hält sich für alle untersuchten Beispiele in problemlos beherschbaren Grenzen. Mit der Kombination von `revert` und `rextract` konnten alle bisher vorliegenden Protokolle analysiert werden.

Die von `rextract` unterstützten Optionen:

Option	Bedeutung
–v	Diese Option bewirkt eine Ausgabe von Informationen über das Programm und einzelne Phasen während des Programmlaufs.
–h	Bei Verwendung der Option –h wird das Programm nach Ausgabe der Programminformationen abgebrochen. Eine Kopie der Programminformationen befindet sich in Anhang B.5.
–r	Die Option –r bewirkt eine sofortige Ausgabe der als notwendig erkannten PCL-Schritte. Damit entsteht auch als Ausgabe von `rextract` ein PCL-Protokoll, in dem die Reihenfolge der Schritte vertauscht ist. Es lassen sich so auch Protokolle bearbeiten, bei denen der virtuelle Speicher auch das extrahierte Protokoll nicht mehr aufnehmen kann. Bisher liegt kein Beispiel vor, bei dem diese Option notwendig ist.
–c	Ist diese Option gesetzt, so werden im Eingabeprotokoll enthaltene Kommentare beibehalten, sofern sie PCL-Schritten in der Ausgabe zugeordnet werden können. Diese Option ist bei `rextract` nur sinnvoll, wenn Kommentare und PCL-Schritte in einer gemeinsamen Zeile stehen.
–o `<outfile>`	Die Option –o dient der Festlegung einer Datei für die Ausgabe des Programmes. Ist diese Option gesetzt, so wird die Ausgabe in die Datei mit Namen `<outfile>` geschrieben, sonst wird sie auf der Standard-Ausgabe ausgegeben.

Konstante	**Default**	**Option**
Konstanten aus 4.1 (Häufig verwendete Zwischenergebnisse)		
MINUSED	4	-o_min_used
Konstanten aus 4.2 (Wichtige Teilergebnisse)		
MINWEIGHT	11	-i_lemma_weight
Konstanten aus 4.3 (Isolierte Teilergebnisse)		
WEIGHTFACTOR	0.5	-t_weight_factor
OFFSET	2	-t_offset
Konstanten aus 4.4 (Syntaktische Kriterien)		
MAXSIZE	1	-s_average_size
AVERAGESIZE	2	-s_max_size
MINFAK	5	-s_min_fak
MAXFAK	0	-s_max_fak
Konstanten aus 4.5 (Analyse der verwendeten Inferenzschritte)		
INITW	1	-c_init_weight
HYPOW	0	-c_hypo_weight
QUOTEW	0	-c_quot_weight
ORIENTW	0	-c_orient_weight
CPW	3	-c_cp_weight
REDW	2	-c_redu_weight
INSW	0	-c_inst_weight
MINWEIGHT	15	-c_lemma_weight
Konstanten aus 4.6 (Unterteilung langer Beweise)		
MAXLEN	10	-p_max_length
Konstanten aus 4.7 (Negative Kriterien)		
MINUSED	1	-u_min_used
MINLEN	2	-u_min_length

Tabelle 6: Konstanten und Optionen bei `lemma`

6.4 Strukturierung des Beweises: `lemma`

Das Programm `lemma` sucht in einem PCL-Protokoll nach wichtigen Schritten und zeichnet diese als Lemmata aus. Es realisiert den Algorithmus aus 4.9 und erlaubt die Verwendung aller in Kapitel 4 vorgestellten Kriterien. Zur Konfigurierung dieser Kriterien unterstützt `lemma` eine Vielzahl von Optionen, mit denen alle in den Kriterien verwendeten Konstanten beeinflußt werden können. Die Zuordnung der Konstanten zu den Optionen und die Vorgabewerte für die Konstanten finden sich in Tabelle 6.

Die Kriteriengruppen in dieser Tabelle können einzeln oder in Kombinationen für die Suche nach Lemmata angegeben werden. Sind in einer Gruppe mehrere Kriterien zusammengefaßt – dieses ist bei den Kriterien nach 4.4 und 4.7 der Fall –, so können die Kriterien durch Setzen der zugehörigen Konstanten auf den Wert 0 aus der Gruppe entfernt werden.

Um die verschiedenen rekursiven Funktionen aus Kapitel 4 effizient zu realisieren, repräsentiert `lemma` den Beweis intern durch einen bidirektional gerichteten Graphen, der die Verbindungen zwischen den einzelnen PCL-Schritten wiederspiegelt. Beim Aufbau dieses Graphen werden, ähnlich wie bei `extract`, überflüssige PCL-Schritte aus dem Protokoll entfernt. Trotzdem sollte `lemma` in der Regel nur zur Bearbeitung von bereits extrahierten Protokollen eingesetzt werden, da die reinen Extraktionsprogramme ihre Aufgabe wesentlich effizienter verrichten. Eine direkte Bearbeitung des Ausgangsprotokoll mit `lemma` ist nur für kurze Beweise zu empfehlen.

Neben den Optionen zum Setzen der Konstanten unterstützt `lemma` weitere Optionen:

Option	Bedeutung
–v	Diese Option bewirkt eine Ausgabe von Informationen über das Programm und einzelne Phasen während des Programmlaufs.
–h	Bei Verwendung der Option –h wird das Programm nach Ausgabe der Programminformationen abgebrochen. Eine Kopie der Programminformationen befindet sich in Anhang B.6.
–c	Ist diese Option gesetzt, so werden im Eingabeprotokoll enthaltene Kommentare beibehalten, sofern sie PCL-Schritten in der Ausgabe zugeordnet werden können. Weiterhin werden neue Kommentare erzeugt, die die für eine Entscheidung entscheidenden Kriterien dokumentieren. Um die Entscheidungen überprüfen zu können, werden auch einige charakteristische Werte in diesen Kommentaren festgehalten.
–o `<outfile>`	Die Option –o dient der Festlegung einer Datei für die Ausgabe des Programmes. Ist diese Option gesetzt, so wird die Ausgabe in die Datei mit Namen `<outfile>` geschrieben, sonst wird sie auf der Standard-Ausgabe ausgegeben.
–criteria `<w>`	Mit der Option –criteria werden die für die Suche zu verwendeten Kriterien angegeben. `<w>` ist dabei ein Wort, das nur Buchstaben aus der Menge $\{\mathtt{s},\mathtt{o},\mathtt{i},\mathtt{t},\mathtt{c},\mathtt{p}\}$ enthält. Die einzelnen Buchstaben repräsentieren jeweils eine Kriteriengruppe, die Zuordnung ist in Tabelle 7 wiedergegeben. Entfällt die Option, so werden die durch das Wort `iotp` repräsentierten Kriterien verwendet. Die negativen Kriterien aus 4.7 werden immer verwendet, sofern dieses nicht durch geeignete Wahl der Konstanten verhindert wird.
–iterate	Ist diese Option gesetzt, so werden die zu verwendeten Kriterien nach der Methode (2) aus 4.8 kombiniert, das komplette PCL-Protokoll wird also mit jedem Kriterium einzeln nach Lemmata durchsucht. Die Kriterien werden in der mit –criteria angegebenen Reihenfolge verwendet. Fehlt –iterate, so werden die Kriterien nach Methode (1) aus 4.8 kombiniert, das PCL-Protokoll wird also nur einmal mit allen Kriterien durchsucht.

6.5 Transformation in Gleichheitsketten: `proof`

`proof` ist das letzte im Rahmen dieser Arbeit erstellte Programm. Es verwendet den Algorithmus aus 5.4 und erzeugt zu einem eingegebenen PCL-Protokoll einen Beweis in Form von Gleichheitsketten. Diese Ketten werden in einem Format ausgegeben, das einem von Menschen geführten Beweis möglichst nahe kommt. Ein von `proof` erzeugter Beweis besteht aus einer Liste der Axiome, einer Behauptung oder einer Liste von Behauptungen, gefolgt von Teilbeweisen

Buchstabe	**Kriteriengruppe**
`o`	Häufig benutzte Zwischenergebnisse (4.1)
`i`	Wichtige Teilergebnisse (4.2)
`t`	Isolierte Teilergebnisse (4.3)
`s`	Syntaktische Kriterien(4.4)
`c`	Analyse der verwendeten Inferenzschritte (4.5)
`p`	Unterteilung langer Beweise (4.6)

Tabelle 7: Buchstabencodes für die Kriterien bei `lemma`

für verwendete Lemmata und die Beweisziele. Jeder Teilbeweis ist eine Gleichheitskette, bei den Umformung in dieser Kette werden Stelle, Substitution, verwendete Gleichung und die Richtung der Gleichungsanwendung angegeben.

`proof` unterstützt verschiedene Variationen dieses Ausgabeformates. Insbesondere können die Informationen über die bei der Umformung verwendeten Stellen und Substitutionen unterdrückt werden. Auch kann `proof` den entstehenden Gleichheitsbeweis direkt in der Textbeschreibungssprache LaTeX (citeLa85) schreiben. Dabei werden die ersetzten Teilterme durch Unterstreichen, die eingesetzten Terme durch Fettdruck gekennzeichnet.

Um die Verarbeitung von Protokollen zu verkürzen, ignoriert `proof` normalerweise die markierten Lemmata und sucht nach eigenen Lemmata. Dabei verwendet es den selben Algorithmus wie `lemma`, erlaubt aber keine Beeinflussung der Kriterien und ihrer Konstanten.

Zur Steuerung dieser Möglichkeiten unterstützt `proof` die folgende Liste von Optionen:

Option	Bedeutung
–v	Diese Option bewirkt eine Ausgabe von Informationen über das Programm und einzelne Phasen während des Programmlaufs.
–h	Bei Verwendung der Option –h wird das Programm nach Ausgabe der Programminformationen abgebrochen. Eine Kopie der Programminformationen befindet sich in Anhang B.7.
–o `<outfile>`	Die Option –o dient der Festlegung einer Datei für die Ausgabe des Programmes. Ist diese Option gesetzt, so wird die Ausgabe in die Datei mit Namen `<outfile>` geschrieben, sonst wird sie auf der Standard-Ausgabe ausgegeben.
–nobrackets	Ist diese Option gesetzt, so gibt `proof` Konstanten in den Termen ohne Klammern aus. Damit lassen sich Konstanten und Variablen nur noch durch Namenskonventionen unterscheiden. `proof` verwendet als Variablen nur $\{\mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{u}, \mathtt{v}, \mathtt{w}, \mathtt{p}, \mathtt{q}, \mathtt{x0}, \mathtt{x1}, \ldots\}$.
–nolemmas	Wird diese Option gesetzt, so verwendet `proof` die Lemmata aus dem Eingabeprotokoll und erzeugt keine eigenen Lemmata.
–noplace	Diese Option unterdrückt die Ausgabe der Stellenangaben in den erzeugten Beweisen.
–nosubst	Wird diese Option verwendet, so verzichtet `proof` auf die Angabe der für die Gleichheitsumformungen verwendeten Substitutionen. Insbesondere große Beweise gewinnen so Übersichtlichkeit.
–latex	Mit dieser Option wird der erzeugte Beweis in LaTeX geschrieben. Dabei wird keine vollständige LaTeX-Datei geschrieben, sondern eine Datei, die mittels `include` oder `input` in ein bestehendes Dokument eingebunden werden kann.

7 Zusammenfassung und Entwicklungsmöglichkeiten

In diesem kurzen Kapitel soll stichwortartig das Erreichte zusammengefaßt werden. Gleichzeitig soll auf offene Fragen und weitere Entwicklungsmöglichkeiten hingewiesen werden. Zunächst folgt die Zusammenfassung der Ergebnisse.

- Mit der Protokollsprache PCL wurde die Möglichkeit zur genauen Untersuchung von Gleichheitsbeweisen und Beweisern geschaffen.
- Untersuchungen an PCL-Protokollen haben bereits eine erhebliche Verbesserung in den Strategien des DISCOUNT-Systems ermöglicht. Weiter Verbesserungen sind für die Zukunft zu erwarten.
- Die entwickelten Kriterien für die Beweisstrukturierung ermöglichen eine automatische Gliederung von Beweisen, die mit vertretbarem Aufwand selbst von Menschen kaum verbessert werden kann.
- Die Darstellung der Beweise im Kalkül der begründeten Gleichungen ermöglicht ein sinnvolles Nachvollziehen der Beweise durch Menschen. Diese Beweise sind auch für Mathematiker ohne Kenntnis von automatischen Beweisern verständlich. Insbesondere auf den Gebieten der Ring- und Gruppentheorie ergeben sich sehr gute Ergebnisse.
- Die implementierten Programme laufen stabil und bisher ohne erkennbare Fehler. Sie haben sich als ausreichend für die Bearbeitung selbst extrem großer Beweise erwiesen.

Auch für weitere Arbeiten bieten sich viele Ansatzpunkte.

- Die Protokollsprache PCL kann auf andere Kalküle erweitert werden. Hier bieten sich zunächst Erweiterungen auf Vervollständiger mit unterliegenden Theorien oder Sortenhierarchien an, PCL kann aber zur die Darstellung fast aller Inferenzprozesse dienen. Der Vergleich der verschiedenen Kalküle kann Aufschluß über allgemeine Prinzipien geben.
- Für die nahe Zukunft ist die Implementierung einer PCL-Ausgabe für die parallele Version des DISCOUNT-Systems geplant. Damit kann auch die parallele Vervollständigung in vollem Ausmaß von der Analyse extrahierter Protokolle profitieren. Der Vergleich von Beweisen des parallelen Systems mit sequentiell gewonnenen Beweisen kann prinzipielle Unterschiede und Gemeinsamkeiten aufzeigen.
- Untersuchungen von extrahierten PCL-Protokollen können wichtige Teilstrukturen enthüllen. Beweiser können eventuell aus solchen Protokollen günstige Strategien selbstständig entnehmen, also lernen.
- Weitere Lemma-Kriterien können die Beweisstrukturierung verbessern. So bieten zum Beispiel die in Kapitel 4.8 erwähnten und noch nicht implementierten Verfahren zur Kombination verschiedener Kriterien das Potential zu deutlichen Verbesserungen in sich.
- Die Darstellung der Beweise durch `proof` kann weiter verbessert werden, indem man etwa eine Transformation in die in vielen Fällen übliche Infix-Notation vornimmt und den Beweis interaktiv um Kommentare ergänzen läßt.

- Auch für die Darstellung der oben erwähnten anderen Kalküle können Verfahren entwickelt werden, die der menschlichen Gewohnheit entsprechen.

A Beweisbeispiele

In disem Anhang werden einige Beispiele für analysierte Gleichheitsbeweise exemplarisch untersucht und in verschiedenen Stadien der Aufbereitung wiedergegeben.

A.1 Ein Beispiel aus der Ringtheorie

In [LO82] präsentieren die Autoren eine Reihe von Problemen für Gleichheitsbeweiser, darunter zwei verwandte Probleme aus der Ringtheorie. Da in dieser Domäne auch Menschen mit Gleichheitsumformungen arbeiten, sind diese Probleme für die Aufbereitung und Untersuchung besonders geeignet. Das einfachere dieser Beispiele ist von moderater Schwierigkeit, die Länge des Beweises erlaubt eine eingehendere Analyse.

A.1.1 Die Problemstellung

Zu zeigen ist das Theorem, nach dem ein Ring mit $\mathtt{x} * \mathtt{x} = \mathtt{x}$ kommutativ ist. Das DISCOUNT-System arbeitet mit folgender Spezifikation:

```
MODE        PROOF

NAME        Lusk3

ORDERING    XKBO
            f:5 > j:4 > g:3 > 0:1 > b:1 > a:1  # > 0:1

EQUATIONS   j (0,x)       = x                    # 0 ist linksneutral
            j (x,0)       = x                    # 0 ist rechtsneutral
            j (g (x),x)   = 0                    # Es existiert ein Linksinverses
            j (x,g (x))   = 0                    # Es existiert ein Rechtsinverses
            j (j (x,y),z) = j (x,j (y,z))        # Addition ist assoziativ
            j (x,y)       = j(y,x)               # Addition ist kommutativ
            f (f (x,y),z) = f (x,f (y,z))        # Multiplikation ist assoziativ
            f (x,j (y,z)) = j (f (x,y),f (x,z))  # Distributivgesetze
            f (j (x,y),z) = j (f (x,z),f (y,z))  #
            f (x,x)       = x                    # Zusaetzliches Axiom: x*x = x

CONCLUSION  f (a,b) = f (b,a)                    # Theorem
```

A.1.2 Der Beweislauf

Das DISCOUNT-System protokolliert den gefundenen Beweis in einem eigenen Format. Dieses Protokoll hat eine Länge von etwa 445 Kilobyte und 13000 Zeilen. Es wird deshalb nur im Auszug wiedergegeben.

```
CP10   : initial
         f (x,x) = x
CP1    : initial
         j (0(),x) = x
CP2    : initial
         j (x,0()) = x
CP3    : initial
         j (g (x),x) = 0()

...

CP9    : initial
         f (j (x,y),z) = j (f (x,z),f (y,z))
G1     : initial
         f (a(),b()) = f (b(),a())

R1     : <== CP10
         f (x,x) -> x
R2     : <== CP1
         j (0(),x) -> x
R3     : <== CP2
         j (x,0()) -> x
CP11   : buildcp (R3,L,R2,L)
         j (0(),0())
         0() = 0()
U11    : delete

...

R55    : reduce (E7,L,L)
         f (x,j (y,x)) -> j (x,f (y,x))
U55    : reduce (R37,L,L)
         j (x,f (x,y)) = j (x,f (y,x))
CP2004 : <== U55
         j (x,f (x,y)) = j (x,f (y,x))
G1     : subsum (E7)
         f (a(),b()) = f (b(),a())
```

Zur weiteren Bearbeitung wird das Protokoll mit `trans` in echtes PCL übersetzt. Dieses Protokoll hat eine Länge von rund 362 Kilobyte und 5009 Beweisschritte.

```
    0 : tes-eqn : f(x,x) = x : initial
    1 : tes-eqn : j(0(),x) = x : initial
    2 : tes-eqn : j(x,0()) = x : initial

  ...

    8 : tes-eqn : f(x,j(y,z)) = j(f(x,y),f(x,z)) : initial
    9 : tes-eqn : f(j(x,y),z) = j(f(x,z),f(y,z)) : initial
```

```
  10 : tes-goal : f(a(),b()) = f(b(),a()) : hypothesis

 ...

5006 : tes-eqn : f(x,j(y,x)) = j(x,f(y,x)) : tes-red(4435,L,4837,L)
5007 : tes-eqn : j(x,f(x,y)) = j(x,f(y,x)) : tes-red(5006,L,2859,L)
5008 : tes-final : f(a(),b()) = f(b(),a()) : instance(10,4837)
```

Werden aus diesem Protokoll die für den Beweis nicht verwendeten Schritte gelöscht, so bleiben 83 notwendige Schritte übrig. Da dieses Protokoll immer wieder als Beispiel verwendet wird, soll es hier vollständig wiedergegeben werden.

```
   0 : tes-eqn : f(x,x) = x : initial
   1 : tes-eqn : j(0(),x) = x : initial
   2 : tes-eqn : j(x,0()) = x : initial
   4 : tes-eqn : j(x,g(x)) = 0() : initial
   5 : tes-eqn : j(x,y) = j(y,x) : initial
   6 : tes-eqn : j(j(x,y),z) = j(x,j(y,z)) : initial
   8 : tes-eqn : f(x,j(y,z)) = j(f(x,y),f(x,z)) : initial
   9 : tes-eqn : f(j(x,y),z) = j(f(x,z),f(y,z)) : initial
  10 : tes-goal : f(a(),b()) = f(b(),a()) : hypothesis
  11 : tes-rule : f(x,x) -> x : orient(0,u)
  12 : tes-rule : j(0(),x) -> x : orient(1,u)
  13 : tes-rule : j(x,0()) -> x : orient(2,u)
  20 : tes-rule : j(x,g(x)) -> 0() : orient(4,u)
  45 : tes-rule : j(j(x,y),z) -> j(x,j(y,z)) : orient(6,u)
  53 : tes-eqn : j(x,j(g(x),y)) = j(0(),y) : cp(45,L.1,20,L)
  54 : tes-eqn : j(x,j(g(x),y)) = y : tes-red(53,R,12,L)
  59 : tes-eqn : j(x,j(y,z)) = j(y,j(z,x)) : cp(5,L,45,L)
  65 : tes-rule : j(x,j(g(x),y)) -> y : orient(54,u)
  71 : tes-eqn : g(g(x)) = j(x,0()) : cp(65,L.2,20,L)
  72 : tes-eqn : g(g(x)) = x : tes-red(71,R,13,L)
  87 : tes-eqn : x = j(y,j(x,g(y))) : cp(65,L.2,5,L)
  89 : tes-rule : g(g(x)) -> x : orient(72,u)
  93 : tes-eqn : x = j(g(y),j(y,x)) : cp(65,L.2.1,89,L)
  97 : tes-rule : j(x,j(y,g(x))) -> y : orient(87,x)
 115 : tes-eqn : x = j(g(y),j(x,y)) : cp(97,L.2.2,89,L)
 126 : tes-rule : j(g(x),j(x,y)) -> y : orient(93,x)
 160 : tes-rule : j(g(x),j(y,x)) -> y : orient(115,x)
 181 : tes-eqn : g(x) = j(g(j(x,y)),y) : cp(160,L.2,126,L)
 182 : tes-eqn : g(x) = j(y,g(j(x,y))) : tes-red(181,R,5,L)
 298 : tes-rule : j(x,g(j(y,x))) -> g(y) : orient(182,x)
2615 : tes-rule : j(f(x,y),f(x,z)) -> f(x,j(y,z)) : orient(8,x)
2616 : tes-eqn : f(x,j(x,y)) = j(x,f(x,y)) : cp(2615,L.1,11,L)
2617 : tes-eqn : f(x,j(y,x)) = j(f(x,y),x) : cp(2615,L.2,11,L)
2618 : tes-eqn : f(x,j(y,x)) = j(x,f(x,y)) : tes-red(2617,R,5,L)
2656 : tes-rule : f(x,j(x,y)) -> j(x,f(x,y)) : orient(2616,u)
2669 : tes-eqn : j(x,f(x,0())) = f(x,x) : cp(2656,L.2,13,L)
2670 : tes-eqn : j(x,f(x,0())) = x : tes-red(2669,R,11,L)
2707 : tes-rule : j(x,f(x,0())) -> x : orient(2670,u)
```

```
2721 : tes-eqn : f(g(x),0()) = j(x,g(x)) : cp(65,L.2,2707,L)
2722 : tes-eqn : f(g(x),0()) = 0() : tes-red(2721,R,20,L)
2767 : tes-rule : f(g(x),0()) -> 0() : orient(2722,u)
2788 : tes-eqn : 0() = f(x,0()) : cp(2767,L.1,89,L)
2791 : tes-rule : f(x,0()) -> 0() : orient(2788,x)
2859 : tes-rule : f(x,j(y,x)) -> j(x,f(x,y)) : orient(2618,u)
2883 : tes-eqn : j(g(x),f(g(x),x)) = f(g(x),0()) : cp(2859,L.2,20,L)
2884 : tes-eqn : j(g(x),f(g(x),x)) = 0() : tes-red(2883,R,2791,L)
2944 : tes-rule : j(g(x),f(g(x),x)) -> 0() : orient(2884,u)
2961 : tes-eqn : f(g(x),x) = j(x,0()) : cp(65,L.2,2944,L)
2962 : tes-eqn : f(g(x),x) = x : tes-red(2961,R,13,L)
3044 : tes-rule : f(g(x),x) -> x : orient(2962,u)
3464 : tes-rule : j(f(x,y),f(z,y)) -> f(j(x,z),y) : orient(9,x)
3465 : tes-eqn : f(j(x,y),x) = j(x,f(y,x)) : cp(3464,L.1,11,L)
3466 : tes-eqn : f(j(x,y),y) = j(f(x,y),y) : cp(3464,L.2,11,L)
3467 : tes-eqn : f(j(x,y),y) = j(y,f(x,y)) : tes-red(3466,R,5,L)
3610 : tes-rule : f(j(x,y),x) -> j(x,f(y,x)) : orient(3465,u)
3630 : tes-eqn : j(x,f(0(),x)) = f(x,x) : cp(3610,L.1,13,L)
3631 : tes-eqn : j(x,f(0(),x)) = x : tes-red(3630,R,11,L)
3632 : tes-eqn : j(x,f(g(x),x)) = f(0(),x) : cp(3610,L.1,20,L)
3633 : tes-eqn : j(x,x) = f(0(),x) : tes-red(3632,L.2,3044,L)
3682 : tes-rule : j(x,f(0(),x)) -> x : orient(3631,u)
3867 : tes-eqn : x = j(x,j(x,x)) : cp(3682,L.2,3633,R)
3943 : tes-rule : j(x,j(x,x)) -> x : orient(3867,x)
3983 : tes-eqn : g(x) = j(j(x,x),g(x)) : cp(298,L.2.1,3943,L)
3984 : tes-eqn : g(x) = j(g(x),j(x,x)) : tes-red(3983,R,5,L)
3985 : tes-eqn : g(x) = j(x,j(x,g(x))) : tes-red(3984,R,59,L)
3986 : tes-eqn : g(x) = j(x,0()) : tes-red(3985,R.2,20,L)
3987 : tes-eqn : g(x) = x : tes-red(3986,R,13,L)
4117 : tes-rule : g(x) -> x : orient(3987,u)
4132 : tes-eqn : j(x,j(x,y)) = y : tes-red(65,L.2.1,4117,L)
4259 : tes-rule : j(x,j(x,y)) -> y : orient(4132,u)
4263 : tes-eqn : j(j(x,y),f(j(x,y),x)) = f(j(x,y),y) : cp(2859,L.2,4259,L)
4264 : tes-eqn : j(j(x,y),j(x,f(y,x))) = f(j(x,y),y) : tes-red(4263,L.2,3610,L)
4265 : tes-eqn : j(x,j(f(y,x),j(x,y))) = f(j(x,y),y) : tes-red(4264,L,59,L)
4266 : tes-eqn : j(x,j(j(x,y),f(y,x))) = f(j(x,y),y) : tes-red(4265,L.2,5,L)
4267 : tes-eqn : j(x,j(x,j(y,f(y,x)))) = f(j(x,y),y) : tes-red(4266,L.2,45,L)
4268 : tes-eqn : j(x,f(x,y)) = f(j(y,x),x) : tes-red(4267,L,4259,L)
4435 : tes-rule : f(j(x,y),y) -> j(y,f(x,y)) : orient(3467,u)
4800 : tes-eqn : j(x,f(x,y)) = j(x,f(y,x)) : tes-red(4268,R,4435,L)
4834 : tes-eqn : j(x,f(j(x,y),x)) = j(x,j(x,f(x,y))) : cp(4800,L.2,2656,L)
4835 : tes-eqn : j(x,j(x,f(y,x))) = j(x,j(x,f(x,y))) : tes-red(4834,L.2,3610,L)
4836 : tes-eqn : f(x,y) = j(y,j(y,f(y,x))) : tes-red(4835,L,4259,L)
4837 : tes-eqn : f(x,y) = f(y,x) : tes-red(4836,R,4259,L)
5008 : tes-final : f(a(),b()) = f(b(),a()) : instance(10,4837)
```

A.1.3 Darstellung des Beweises mit den Default-Einstellungen

In diesem Abschnitt wird der Beweis mit den Lemmata untersucht und dargestellt, die die Werkzeuge `lemma` beziehungsweise `proof` ohne Änderung der voreingestellten Parameter findet. Mit der Default-Strategie findet man die folgenden Lemmata in dem Protokoll:

```
  65 : tes-lemma : j(x,j(g(x),y)) -> y : orient(54,u)
  89 : tes-lemma : g(g(x)) -> x : orient(72,u)
2884 : tes-lemma : j(g(x),f(g(x),x)) = 0() : tes-red(2883,R,2791,L)
3610 : tes-lemma : f(j(x,y),x) -> j(x,f(y,x)) : orient(3465,u)
3867 : tes-lemma : x = j(x,j(x,x)) : cp(3682,L.2,3633,R)
3985 : tes-lemma : g(x) = j(x,j(x,g(x))) : tes-red(3984,R,59,L)
4259 : tes-lemma : j(x,j(x,y)) -> y : orient(4132,u)
4267 : tes-lemma : j(x,j(x,j(y,f(y,x)))) = f(j(x,y),y) : tes-red(4266,L.2,45,L)
4836 : tes-lemma : f(x,y) = j(y,j(y,f(y,x))) : tes-red(4835,L,4259,L)
```

Um den Beweis übersichtlich zu halten wurde bei der folgenden Darstellung des Beweises auf die Angabe der Substitutionen verzichtet. Es werden die oben angegebenen Lemmata verwendet.

Consider the following set of axioms:

Axiom 1: f(x, x) = x
Axiom 2: j(0, x) = x
Axiom 3: j(x, 0) = x
Axiom 4: j(x, g(x)) = 0
Axiom 5: j(x, y) = j(y, x)
Axiom 6: j(j(x, y), z) = j(x, j(y, z))
Axiom 7: f(x, j(y, z)) = j(f(x, y), f(x, z))
Axiom 8: f(j(x, y), z) = j(f(x, z), f(y, z))

This theorem holds true:

Theorem 1: f(a, b) = f(b, a)

Proof:

Lemma 1: j(u, j(g(u), z)) = z

$\underline{\mathrm{j(u, j(g(u), z))}}$	=	$\mathbf{j(\underline{j(u, g(u))}, z)}$	by Axiom 6 RL
	=	$\underline{\mathrm{j}(\mathbf{0}, \mathrm{z})}$	by Axiom 4 LR
	=	$\mathbf{z}$	by Axiom 2 LR

Lemma 2: g(g(u)) = u

$\underline{\mathrm{g(g(u))}}$	=	$\mathbf{j(u, \underline{j(g(u), g(g(u)))})}$	by Lemma 1 RL
	=	$\underline{\mathrm{j}(\mathrm{u}, \mathbf{0})}$	by Axiom 4 LR
	=	$\mathbf{u}$	by Axiom 3 LR

Lemma 3: $j(g(z), f(g(z), z)) = 0$

$$\begin{aligned}
\underline{j(g(z), f(g(z), z))} &= \mathbf{j(f(g(z), z), \underline{g(z)})} && \text{by Axiom 5 RL}\\
&= \underline{j(f(g(z), z), \mathbf{f(g(z), g(z))})} && \text{by Axiom 1 RL}\\
&= \mathbf{f(g(z), \underline{j(z, g(z))})} && \text{by Axiom 7 RL}\\
&= f(\underline{g(z)}, \mathbf{0}) && \text{by Axiom 4 LR}\\
&= \underline{f(\mathbf{g(g(g(z)))}, 0)} && \text{by Lemma 2 RL}\\
&= \mathbf{j(g(g(z)), j(\underline{g(g(g(z)))}, f(g(g(g(z))), 0)))} && \text{by Lemma 1 RL}\\
&= j(g(g(z)), \underline{j(\mathbf{f(g(g(g(z))), g(g(g(z))))}, f(g(g(g(z))), 0))}) && \text{by Axiom 1 RL}\\
&= j(g(g(z)), \mathbf{f(g(g(g(z))), \underline{j(g(g(g(z))), 0)})}) && \text{by Axiom 7 RL}\\
&= j(g(g(z)), \underline{f(g(g(g(z))), \mathbf{g(g(g(z)))})}) && \text{by Axiom 3 LR}\\
&= \underline{j(g(g(z)), \mathbf{g(g(g(z)))})} && \text{by Axiom 1 LR}\\
&= \mathbf{0} && \text{by Axiom 4 LR}
\end{aligned}$$

Lemma 4: $f(j(v, y), v) = j(v, f(y, v))$

$$\begin{aligned}
\underline{f(j(v, y), v)} &= \mathbf{j(\underline{f(v, v)}, f(y, v))} && \text{by Axiom 8 LR}\\
&= j(\mathbf{v}, f(y, v)) && \text{by Axiom 1 LR}
\end{aligned}$$

Lemma 5: $g(y) = j(j(y, y), g(y))$

$$\begin{aligned}
\underline{g(y)} &= \mathbf{j(g(j(y, j(y, y))), \underline{j(g(g(j(y, j(y, y)))), g(y))})} && \text{by Lemma 1 RL}\\
&= j(g(j(y, j(y, y))), \mathbf{j(g(y), \underline{g(g(j(y, j(y, y))))})}) && \text{by Axiom 5 LR}\\
&= j(g(j(y, j(y, y))), j(g(y), \mathbf{j(\underline{y}, j(y, y))})) && \text{by Lemma 2 LR}\\
&= j(g(j(y, j(y, y))), \underline{j(g(y), j(\mathbf{g(g(y))}, j(y, y)))}) && \text{by Lemma 2 RL}\\
&= \underline{j(g(j(y, j(y, y))), \mathbf{j(y, y)})} && \text{by Lemma 1 LR}\\
&= \mathbf{j(j(y, y), g(j(y, j(y, \underline{y}))))} && \text{by Axiom 5 LR}\\
&= j(j(y, y), g(j(y, j(y, \mathbf{j(y, \underline{0})})))) && \text{by Axiom 3 RL}\\
&= j(j(y, y), g(j(y, j(y, \underline{j(y, \mathbf{j(g(y), f(g(y), y))})})))) && \text{by Lemma 3 RL}\\
&= j(j(y, y), g(j(y, \underline{j(y, \mathbf{f(g(y), y)})}))) && \text{by Lemma 1 LR}\\
&= j(j(y, y), g(j(y, \mathbf{f(\underline{j(y, g(y))}, y)}))) && \text{by Lemma 4 RL}\\
&= j(j(y, y), g(\underline{j(y, f(\mathbf{0}, y))})) && \text{by Axiom 4 LR}\\
&= j(j(y, y), g(\mathbf{f(\underline{j(y, 0)}, y)})) && \text{by Lemma 4 RL}\\
&= j(j(y, y), g(\underline{f(\mathbf{y}, y)})) && \text{by Axiom 3 LR}\\
&= j(j(y, y), g(\mathbf{y})) && \text{by Axiom 1 LR}
\end{aligned}$$

Lemma 6: $j(u, j(u, z)) = z$

$$\begin{aligned}
j(u, j(\underline{u}, z)) &= j(u, j(\mathbf{j(u, \underline{0})}, z)) && \text{by Axiom 3 RL}\\
&= j(u, j(\underline{j(u, \mathbf{j(u, g(u))})}, z)) && \text{by Axiom 4 RL}\\
&= j(u, j(\underline{\mathbf{j(j(u, u), g(u))}}, z)) && \text{by Axiom 6 RL}\\
&= \underline{j(u, j(\mathbf{g(u)}, z))} && \text{by Lemma 5 RL}\\
&= \mathbf{z} && \text{by Lemma 1 LR}
\end{aligned}$$

Lemma 7: j(z, f(z, w)) = j(z, f(w, z))

j(z, f(z, w))	=	**j(w, j(w, j(z, f(z, w))))**	by Lemma 6 RL
	=	j(w, **j(j(w, z), f(z, w))**)	by Axiom 6 RL
	=	j(w, **j(f(z, w), j(w, z))**)	by Axiom 5 RL
	=	**j(j(w, f(z, w)), j(w, z))**	by Axiom 6 RL
	=	**j(j(w, z), j(w, f(z, w)))**	by Axiom 5 LR
	=	j(j(w, z), **f(j(w, z), w)**)	by Lemma 4 RL
	=	**j(f(j(w, z), w), j(w, z))**	by Axiom 5 RL
	=	j(f(j(w, z), w), **f(j(w, z), j(w, z))**)	by Axiom 1 RL
	=	**f(j(w, z), j(w, j(w, z)))**	by Axiom 7 RL
	=	f(j(w, z), **z**)	by Lemma 6 LR
	=	**j(f(w, z), f(z, z))**	by Axiom 8 LR
	=	j(f(w, z), **z**)	by Axiom 1 LR
	=	**j(z, f(w, z))**	by Axiom 5 LR

Theorem 1: f(a, b) = f(b, a)

f(a, b)	=	**j(b, j(b, f(a, b)))**	by Lemma 6 RL
	=	j(b, **f(j(b, a), b)**)	by Lemma 4 RL
	=	**j(b, f(b, j(b, a)))**	by Lemma 7 RL
	=	j(b, **j(f(b, b), f(b, a))**)	by Axiom 7 LR
	=	j(b, j(**b**, f(b, a)))	by Axiom 1 LR
	=	**f(b, a)**	by Lemma 6 LR

A.1.4 Kurzdarstellungen des Beweises mit anderen Lemmata

Die Qualität der Lemmata zur Beweisgliederung läßt sich nur subjektiv beurteilen. Dafür ist eine Darstellung des Beweises in Form von Gleichheitsketten weit besser geeignet als das PCL-Protokoll. Leider sind diese Beweise relativ lang. Um den Umfang dieser Darstellung in einem vertretbaren Ramen zu halten, werden die Beweise im folgenden in verkürzter Form wiedergegeben. Als Bewertungskriterien werden dabei jeweils die Länge der Teilbeweise und der längste Term angegeben.

Lemmata wurden nur mit dem angegebenen Kriterium und der zusätzlichen Forderung gesucht, daß die Länge der Beweiskette für ein Lemma mindestens 2 sein muß (man vergleiche 4.7). Dabei wurden die Defaults aus 6.4 verwendet. Grundlage war jeweils das PCL-Protokoll aus A.1.2

Lemmata nach 4.1 (Häufig verwendete Zwischenergebnisse)

Durchsucht man den Beweis nach besonder häufig zitierten Schritten, so erhält man folgende Lemmata.

```
  65 : tes-lemma : j(x,j(g(x),y)) -> y : orient(54,u)
3610 : tes-lemma : f(j(x,y),x) -> j(x,f(y,x)) : orient(3465,u)
4259 : tes-lemma : j(x,j(x,y)) -> y : orient(4132,u)
```

Damit ergibt sich folgender Beweis:

Consider the following set of axioms:

Axiom 1: f(x, x) = x
Axiom 2: j(0, x) = x
Axiom 3: j(x, 0) = x
Axiom 4: j(x, g(x)) = 0
Axiom 5: j(x, y) = j(y, x)
Axiom 6: j(j(x, y), z) = j(x, j(y, z))
Axiom 7: f(x, j(y, z)) = j(f(x, y), f(x, z))
Axiom 8: f(j(x, y), z) = j(f(x, z), f(y, z))

This theorem holds true:

Theorem 1: f(a, b) = f(b, a)

Proof:

Lemma 1: j(u, j(g(u), z)) = z
3 Gleichheitsumformungen, längster Term:

j(u, j(g(u), z))

Lemma 2: f(j(v, y), v) = j(v, f(y, v))
2 Gleichheitsumformungen, längster Term:

j(f(v, v), f(y, v))

Lemma 3: j(u, j(u, z)) = z
34 Gleichheitsumformungen, längster Term:

j(u, j(j(j(u, u), g(j(u, j(u, j(u, j(g(g(u)), j(f(g(g(g(u))), g(g(g(u)))), f(g(g(g(u))), 0)))))))), z))

Theorem 1: f(a, b) = f(b, a)
18 Gleichheitsumformungen, längster Term:

j(f(j(j(b, a), b), j(b, a)), f(j(j(b, a), b), j(j(b, a), b)))

Lemmata nach 4.2 (Wichtige Zwischenergebnisse)

Sucht man nach Schritten, die besonders viele Gleichheitsanwendungen repräsentieren, so ergeben sich die folgenden Lemmata:

```
  65 : tes-lemma : j(x,j(g(x),y)) -> y : orient(54,u)
 182 : tes-lemma : g(x) = j(y,g(j(x,y))) : tes-red(181,R,5,L)
2884 : tes-lemma : j(g(x),f(g(x),x)) = 0() : tes-red(2883,R,2791,L)
3867 : tes-lemma : x = j(x,j(x,x)) : cp(3682,L.2,3633,R)
4259 : tes-lemma : j(x,j(x,y)) -> y : orient(4132,u)
4267 : tes-lemma : j(x,j(x,j(y,f(y,x)))) = f(j(x,y),y) : tes-red(4266,L.2,45,L)
4836 : tes-lemma : f(x,y) = j(y,j(y,f(y,x))) : tes-red(4835,L,4259,L)
```

Damit ergibt sich folgender Beweis:

Consider the following set of axioms:

Axiom 1: f(x, x) = x
Axiom 2: j(0, x) = x
Axiom 3: j(x, 0) = x
Axiom 4: j(x, g(x)) = 0
Axiom 5: j(x, y) = j(y, x)
Axiom 6: j(j(x, y), z) = j(x, j(y, z))
Axiom 7: f(x, j(y, z)) = j(f(x, y), f(x, z))
Axiom 8: f(j(x, y), z) = j(f(x, z), f(y, z))

This theorem holds true:

Theorem 1: f(a, b) = f(b, a)

Proof:

Lemma 1: j(u, j(g(u), z)) = z
3 Gleichheitsumformungen, längster Term:

j(u, j(g(u), z))

Lemma 2: g(w) = j(v, g(j(w, v)))
10 Gleichheitsumformungen, längster Term:

j(g(w), j(j(w, v), j(g(j(w, v)), g(g(j(w, v))))))

Lemma 3: j(g(z), f(g(z), z)) = 0
12 Gleichheitsumformungen, längster Term:

j(g(g(z)), j(f(g(g(g(z))), g(g(g(z)))), f(g(g(g(z))), 0)))

Lemma 4: z = j(z, j(z, z))
10 Gleichheitsumformungen, längster Term:

j(z, j(z, j(z, j(g(z), f(g(z), z)))))

Lemma 5: j(u, j(u, z)) = z
6 Gleichheitsumformungen, längster Term:

j(u, j(j(j(u, u), g(j(u, j(u, u)))), z))

Lemma 6: j(w, j(w, j(z, f(z, w)))) = f(j(w, z), z)
10 Gleichheitsumformungen, längster Term:

j(j(w, z), j(f(w, w), f(z, w)))

Lemma 7: f(u, v) = j(v, j(v, f(v, u)))
10 Gleichheitsumformungen, längster Term:

j(j(v, u), j(j(v, u), j(v, f(v, j(v, u)))))

Theorem 1: f(a, b) = f(b, a)
2 Gleichheitsumformungen, längster Term:

j(b, j(b, f(b, a)))

Lemmata nach 4.3 (Isolierte Teilbeweise)

Sucht man im Beweis nach isolierten Teilergebnissen, so findet man die folgenden Lemmata:

```
  65 : tes-lemma : j(x,j(g(x),y)) -> y : orient(54,u)
  89 : tes-lemma : g(g(x)) -> x : orient(72,u)
3610 : tes-lemma : f(j(x,y),x) -> j(x,f(y,x)) : orient(3465,u)
3983 : tes-lemma : g(x) = j(j(x,x),g(x)) : cp(298,L.2.1,3943,L)
4259 : tes-lemma : j(x,j(x,y)) -> y : orient(4132,u)
4837 : tes-lemma : f(x,y) = f(y,x) : tes-red(4836,R,4259,L)
```

Damit ergibt sich der Beweis wie folgt:

Consider the following set of axioms:

Axiom 1: f(x, x) = x
Axiom 2: j(0, x) = x
Axiom 3: j(x, 0) = x
Axiom 4: j(x, g(x)) = 0
Axiom 5: j(x, y) = j(y, x)
Axiom 6: j(j(x, y), z) = j(x, j(y, z))
Axiom 7: f(x, j(y, z)) = j(f(x, y), f(x, z))
Axiom 8: f(j(x, y), z) = j(f(x, z), f(y, z))

This theorem holds true:

Theorem 1: f(a, b) = f(b, a)

Proof:

Lemma 1: j(u, j(g(u), z)) = z
3 Gleichheitsumformungen, längster Term:

j(j(u, g(u)), z)

Lemma 2: g(g(u)) = u
3 Gleichheitsumformungen, längster Term:

j(u, j(g(u), g(g(u))))

Lemma 3: f(j(v, y), v) = j(v, f(y, v))
2 Gleichheitsumformungen, längster Term:

j(f(v, v), f(y, v))

Lemma 4: g(y) = j(j(y, y), g(y))
24 Gleichheitsumformungen, längster Term:

j(j(y, y), g(j(y, j(y, j(y, j(g(g(y)), j(f(g(g(g(y))), g(g(y)))), f(g(g(g(y))), 0))))))))

Lemma 5: j(u, j(u, z)) = z
5 Gleichheitsumformungen, längster Term:

j(u, j(j(u, j(u, g(u))), z))

Lemma 6: f(u, v) = f(v, u)
18 Gleichheitsumformungen, längster Term:

j(f(j(j(v, u), v), j(v, u)), f(j(j(v, u), v), j(j(v, u), v)))

Theorem 1: f(a, b) = f(b, a)
1 Gleichheitsumformung, längster Term:

f(a,b)

Lemmata nach 4.4 (Syntaktische Kriterien)

Durchsucht man den Beweis nach syntaktischen Kriterien, so erhält man relativ viele Lemmata:

```
  54 : tes-lemma : j(x,j(g(x),y)) = y : tes-red(53,R,12,L)
  87 : tes-lemma : x = j(y,j(x,g(y))) : cp(65,L.2,5,L)
  93 : tes-lemma : x = j(g(y),j(y,x)) : cp(65,L.2.1,89,L)
 115 : tes-lemma : x = j(g(y),j(x,y)) : cp(97,L.2.2,89,L)
2670 : tes-lemma : j(x,f(x,0())) = x : tes-red(2669,R,11,L)
2884 : tes-lemma : j(g(x),f(g(x),x)) = 0() : tes-red(2883,R,2791,L)
3631 : tes-lemma : j(x,f(0(),x)) = x : tes-red(3630,R,11,L)
3867 : tes-lemma : x = j(x,j(x,x)) : cp(3682,L.2,3633,R)
3987 : tes-lemma : g(x) = x : tes-red(3986,R,13,L)
4132 : tes-lemma : j(x,j(x,y)) = y : tes-red(65,L.2.1,4117,L)
```

Der zugehörige Beweis:

Consider the following set of axioms:

Axiom 1: f(x, x) = x
Axiom 2: j(0, x) = x
Axiom 3: j(x, 0) = x
Axiom 4: j(x, g(x)) = 0
Axiom 5: j(x, y) = j(y, x)
Axiom 6: j(j(x, y), z) = j(x, j(y, z))
Axiom 7: f(x, j(y, z)) = j(f(x, y), f(x, z))
Axiom 8: f(j(x, y), z) = j(f(x, z), f(y, z))

This theorem holds true:

Theorem 1: f(a, b) = f(b, a)

Proof:

Lemma 1: j(u, j(g(u), z)) = z
3 Gleichheitsumformungen, längster Term:

j(j(u, g(u)), z)

Lemma 2: x = j(u, j(x, g(u)))
2 Gleichheitsumformungen, längster Term:

j(u, j(x, g(u)))

Lemma 3: z = j(g(y), j(y, z))
4 Gleichheitsumformungen, längster Term:

j(g(y), j(j(y, j(g(y), g(g(y)))), z))

Lemma 4: x = j(g(z), j(x, z))
4 Gleichheitsumformungen, längster Term:

j(g(z), j(x, j(z, j(g(z), g(g(z))))))

Lemma 5: j(x, f(x, 0)) = x
4 Gleichheitsumformungen, längster Term:

j(f(x, x), f(x, 0))

Lemma 6: j(g(z), f(g(z), z)) = 0
10 Gleichheitsumformungen, längster Term:

j(g(g(z)), j(g(g(g(z))), f(g(g(g(z))), 0)))

Lemma 7: j(x, f(0, x)) = x
4 Gleichheitsumformungen, längster Term:

j(f(x, x), f(0, x))

Lemma 8: z = j(z, j(z, z))
7 Gleichheitsumformungen, längster Term:

j(z, j(z, j(z, j(g(z), f(g(z), z)))))

Lemma 9: g(y) = y
7 Gleichheitsumformungen, längster Term:

j(g(j(y, j(y, y))), j(g(y), j(y, j(y, y))))

Lemma 10: j(u, j(u, z)) = z
2 Gleichheitsumformungen, längster Term:

j(u, j(g(u), z))

Theorem 1: f(a, b) = f(b, a)
20 Gleichheitsumformungen, längster Term:

j(j(j(b, a), b), j(f(j(b, a), j(b, a)), f(b, j(b, a))))

Lemmata nach 4.5 (Analyse der Inferenzschritte)

Nach diesem Kriterium ergeben sich die folgenden Lemmata:

```
  72 : tes-lemma : g(g(x)) = x : tes-red(71,R,13,L)
 115 : tes-lemma : x = j(g(y),j(x,y)) : cp(97,L.2.2,89,L)
 182 : tes-lemma : g(x) = j(y,g(j(x,y))) : tes-red(181,R,5,L)
2721 : tes-lemma : f(g(x),0()) = j(x,g(x)) : cp(65,L.2,2707,L)
2884 : tes-lemma : j(g(x),f(g(x),x)) = 0() : tes-red(2883,R,2791,L)
3633 : tes-lemma : j(x,x) = f(0(),x) : tes-red(3632,L.2,3044,L)
3867 : tes-lemma : x = j(x,j(x,x)) : cp(3682,L.2,3633,R)
3986 : tes-lemma : g(x) = j(x,0()) : tes-red(3985,R.2,20,L)
4263 : tes-lemma : j(j(x,y),f(j(x,y),x)) = f(j(x,y),y) : cp(2859,L.2,4259,L)
4266 : tes-lemma : j(x,j(j(x,y),f(y,x))) = f(j(x,y),y) : tes-red(4265,L.2,5,L)
4268 : tes-lemma : j(x,f(x,y)) = f(j(y,x),x) : tes-red(4267,L,4259,L)
4834 : tes-lemma : j(x,f(j(x,y),x)) = j(x,j(x,f(x,y))) : cp(4800,L.2,2656,L)
4836 : tes-lemma : f(x,y) = j(y,j(y,f(y,x))) : tes-red(4835,L,4259,L)
4837 : tes-lemma : f(x,y) = f(y,x) : tes-red(4836,R,4259,L)
```

Damit erhält man folgenden Beweis:

Consider the following set of axioms:

Axiom 1: f(x, x) = x
Axiom 2: j(0, x) = x
Axiom 3: j(x, 0) = x
Axiom 4: j(x, g(x)) = 0
Axiom 5: j(x, y) = j(y, x)
Axiom 6: j(j(x, y), z) = j(x, j(y, z))
Axiom 7: f(x, j(y, z)) = j(f(x, y), f(x, z))
Axiom 8: f(j(x, y), z) = j(f(x, z), f(y, z))

This theorem holds true:

Theorem 1: f(a, b) = f(b, a)

Proof:

Lemma 1: g(g(u)) = u
5 Gleichheitsumformungen, längster Term:

j(u, j(g(u), g(g(u))))

Lemma 2: x = j(g(z), j(x, z))
5 Gleichheitsumformungen, längster Term:

j(j(g(z), g(g(z))), x)

Lemma 3: g(w) = j(v, g(j(w, v)))
6 Gleichheitsumformungen, längster Term:

j(g(j(w, v)), j(g(w), j(g(g(w)), v)))

Lemma 4: f(g(u), 0) = j(u, g(u))
7 Gleichheitsumformungen, längster Term:

j(u, j(f(g(u), g(u)), f(g(u), 0)))

Lemma 5: j(g(z), f(g(z), z)) = 0
7 Gleichheitsumformungen, längster Term:

j(f(g(z), z), f(g(z), g(z)))

Lemma 6: j(z, z) = f(0, z)
8 Gleichheitsumformungen, längster Term:

j(z, j(z, j(g(z), f(g(z), z))))

Lemma 7: z = j(z, j(z, z))
8 Gleichheitsumformungen, längster Term:

j(f(z, z), f(0, z))

Lemma 8: g(y) = j(y, 0)
4 Gleichheitsumformungen, längster Term:

j(j(y, y), g(j(y, j(y, y))))

Lemma 9: j(j(w, z), f(j(w, z), w)) = f(j(w, z), z)
8 Gleichheitsumformungen, längster Term:

j(f(j(w, z), w), f(j(w, z), j(w, z)))

Lemma 10: j(w, j(j(w, z), f(z, w))) = f(j(w, z), z)
7 Gleichheitsumformungen, längster Term:

j(j(w, z), j(f(w, w), f(z, w)))

Lemma 11: j(z, f(z, w)) = f(j(w, z), z)
7 Gleichheitsumformungen, längster Term:

j(w, j(j(w, 0), j(z, f(z, w))))

Lemma 12: j(v, f(j(v, u), v)) = j(v, j(v, f(v, u)))
8 Gleichheitsumformungen, längster Term:

j(v, j(f(v, v), f(u, v)))

Lemma 14: f(u, v) = f(v, u)
6 Gleichheitsumformungen, längster Term:

j(v, j(j(v, 0), f(v, u)))

Theorem 1: f(a, b) = f(b, a)
1 Gleichheitsumformung, längster Term:

f(a, b)

Lemmata nach 4.5 (Unterteilung langer Beweise)

Wählt man nur Lemmata, die übermäßig lang Beweisketten vermeiden sollen, so ergeben sich folgende Lemmta:

```
 181 : tes-lemma : g(x) = j(g(j(x,y)),y) : cp(160,L.2,126,L)
2788 : tes-lemma : 0() = f(x,0()) : cp(2767,L.1,89,L)
3633 : tes-lemma : j(x,x) = f(0(),x) : tes-red(3632,L.2,3044,L)
3986 : tes-lemma : g(x) = j(x,0()) : tes-red(3985,R.2,20,L)
4265 : tes-lemma : j(x,j(f(y,x),j(x,y))) = f(j(x,y),y) : tes-red(4264,L,59,L)
4800 : tes-lemma : j(x,f(x,y)) = j(x,f(y,x)) : tes-red(4268,R,4435,L)
4837 : tes-lemma : f(x,y) = f(y,x) : tes-red(4836,R,4259,L)
```

Der entsprechende Beweis:

Consider the following set of axioms:

Axiom 1: f(x, x) = x
Axiom 2: j(0, x) = x
Axiom 3: j(x, 0) = x
Axiom 4: j(x, g(x)) = 0
Axiom 5: j(x, y) = j(y, x)
Axiom 6: j(j(x, y), z) = j(x, j(y, z))
Axiom 7: f(x, j(y, z)) = j(f(x, y), f(x, z))
Axiom 8: f(j(x, y), z) = j(f(x, z), f(y, z))

This theorem holds true:

Theorem 1: f(a, b) = f(b, a)

Proof:

Lemma 1: g(w) = j(g(j(w, v)), v)
17 Gleichheitsumformungen, längster Term:

j(g(j(w, v)), j(g(w), j(j(j(w, v), g(j(w, v))), g(g(j(w, v))))))

Lemma 2: 0 = f(y, 0)
13 Gleichheitsumformungen, längster Term:

j(g(y), j(f(g(g(y)), g(g(y))), f(g(g(y)), 0)))

Lemma 3: j(z, z) = f(0, z)
12 Gleichheitsumformungen, längster Term:

j(z, j(z, j(f(g(z), z), f(g(z), g(z)))))

Lemma 4: g(y) = j(y, 0)
9 Gleichheitsumformungen, längster Term:

j(j(y, y), g(j(f(y, y), f(0, y))))

Lemma 5: j(w, j(f(z, w), j(w, z))) = f(j(w, z), z)
12 Gleichheitsumformungen, längster Term:

j(f(j(w, z), w), f(j(w, z), j(w, z)))

Lemma 6: j(z, f(z, w)) = j(z, f(w, z))
11 Gleichheitsumformungen, längster Term:

j(w, j(j(w, 0), j(z, f(z, w))))

Lemma 7: f(u, v) = f(v, u)
15 Gleichheitsumformungen, längster Term:

j(v, j(f(v, v), f(v, u)))

Theorem 1: f(a, b) = f(b, a)
1 Gleichheitsumformung, längster Term:

f(b, a)

A.2 Die Vervollständigung der Gruppenaxiome

Die Vervollständigung der Gruppenaxiome wurde bereits in [KB70] als Beispiel herangezogen. Dabei wird aus den 3 Axiomen der Gruppe eine Menge von 10 Regeln berechnet, die eine konvergente Reduktionsrelation beschreiben. Die folgende Spezifikation dieses Problems dient als Eingabe für das DISCOUNT-System:

```
MODE        COMPLETION

NAME        Gruppe

ORDERING    LPO
            inv > f > 1

EQUATIONS   f (x,1) = x
            f (x,inv (x)) = 1
            f (f (x,y),z) = f (x,f (y,z))
```

Das extrahierte PCL-Protokoll zeigt die Herleitung des vervollständigten Regelsystems und die automatisch erzeugten Lemmata.

```
  0 : tes-eqn : f(x,1()) = x : initial
  1 : tes-eqn : f(x,inv(x)) = 1() : initial
  2 : tes-eqn : f(f(x,y),z) = f(x,f(y,z)) : initial
  3 : tes-rule : f(x,1()) -> x : orient(0,u)
  4 : tes-rule : f(x,inv(x)) -> 1() : orient(1,u)
  5 : tes-rule : f(f(x,y),z) -> f(x,f(y,z)) : orient(2,u)
  8 : tes-eqn : f(x,f(1(),y)) = f(x,y) : cp(5,L.1,3,L)
  9 : tes-eqn : f(x,f(y,inv(f(x,y)))) = 1() : cp(5,L,4,L)
 10 : tes-eqn : f(x,f(inv(x),y)) = f(1(),y) : cp(5,L.1,4,L)
 15 : tes-rule : f(x,f(1(),y)) -> f(x,y) : orient(8,u)
 39 : tes-rule : f(x,f(y,inv(f(x,y)))) -> 1() : orient(9,u)
 67 : tes-rule : f(x,f(inv(x),y)) -> f(1(),y) : orient(10,u)
 71 : tes-eqn : f(1(),inv(inv(x))) = f(x,1()) : cp(67,L.2,4,L)
 72 : tes-eqn : f(1(),inv(inv(x))) = x : tes-red(71,R,3,L)
 97 : tes-rule : f(1(),inv(inv(x))) -> x : orient(72,u)
 99 : tes-eqn : f(x,inv(inv(y))) = f(x,y) : cp(15,L.2,97,L)
101 : tes-lemma : f(x,inv(inv(y))) -> f(x,y) : orient(99,u)
102 : tes-eqn : f(1(),x) = x : tes-red(97,L,101,L)
116 : tes-lemma : f(1(),x) -> x : orient(102,u)
118 : tes-rule : f(x,f(inv(x),y)) -> y : tes-red(67,R,116,L)
120 : tes-eqn : inv(1()) = 1() : cp(116,L,4,L)
124 : tes-eqn : inv(inv(x)) = f(1(),x) : cp(116,L,101,L)
125 : tes-lemma : inv(inv(x)) = x : tes-red(124,R,116,L)
136 : tes-rule : inv(1()) -> 1() : orient(120,u)
148 : tes-rule : inv(inv(x)) -> x : orient(125,u)
153 : tes-eqn : 1() = f(inv(x),x) : cp(4,L.2,148,L)
154 : tes-eqn : x = f(inv(y),f(y,x)) : cp(118,L.2.1,148,L)
157 : tes-rule : f(inv(x),x) -> 1() : orient(153,x)
172 : tes-lemma : f(inv(x),f(x,y)) -> y : orient(154,x)
199 : tes-eqn : f(x,inv(f(y,x))) = f(inv(y),1()) : cp(172,L.2,39,L)
200 : tes-eqn : f(x,inv(f(y,x))) = inv(y) : tes-red(199,R,3,L)
206 : tes-rule : f(x,inv(f(y,x))) -> inv(y) : orient(200,u)
236 : tes-eqn : inv(f(x,y)) = f(inv(y),inv(x)) : cp(172,L.2,206,L)
331 : tes-rule : inv(f(x,y)) -> f(inv(y),inv(x)) : orient(236,u)
404 : tes-final : inv(1()) -> 1() : 136
405 : tes-final : inv(inv(x)) -> x : 148
406 : tes-final : inv(f(x,y)) -> f(inv(y),inv(x)) : 331
407 : tes-final : f(x,1()) -> x : 3
408 : tes-final : f(1(),x) -> x : 116
409 : tes-final : f(x,inv(x)) -> 1() : 4
410 : tes-final : f(inv(x),x) -> 1() : 157
411 : tes-final : f(f(x,y),z) -> f(x,f(y,z)) : 5
412 : tes-final : f(x,f(inv(x),y)) -> y : 118
413 : tes-final : f(inv(x),f(x,y)) -> y : 172
```

Damit ergibt sich der folgende Gleichheitsbeweis:

Consider the following set of axioms:

Axiom 1: f(x, 1) = x
Axiom 2: f(x, inv(x)) = 1
Axiom 3: f(f(x, y), z) = f(x, f(y, z))

These theorems hold true:

Theorem 1: inv(1) = 1
Theorem 2: inv(inv(y)) = y
Theorem 3: inv(f(v, w)) = f(inv(w), inv(v))
Theorem 4: f(x, 1) = x
Theorem 5: f(1, u) = u
Theorem 6: f(x, inv(x)) = 1
Theorem 7: f(inv(y), y) = 1
Theorem 8: f(f(x, y), z) = f(x, f(y, z))
Theorem 9: f(u, f(inv(u), z)) = z
Theorem 10: f(inv(y), f(y, z)) = z

Proof:

Lemma 1: f(v, inv(inv(y))) = f(v, y)

$$\begin{array}{llll}
\mathrm{f}(\underline{\mathrm{v}}, \mathrm{inv}(\mathrm{inv}(\mathrm{y}))) & = & \underline{\mathrm{f}(\mathbf{f(v,\ 1)}, \mathrm{inv}(\mathrm{inv}(\mathrm{y})))} & \text{by Axiom 1 RL} \\
 & = & \mathbf{f(v,\ f(\underline{1},\ inv(inv(y))))} & \text{by Axiom 3 LR} \\
 & = & \mathrm{f}(\mathrm{v}, \underline{\mathrm{f}(\mathbf{f(y,\ inv(y))}, \mathrm{inv}(\mathrm{inv}(\mathrm{y})))}) & \text{by Axiom 2 RL} \\
 & = & \mathrm{f}(\mathrm{v}, \underline{\mathbf{f(y,\ f(inv(y),\ inv(inv(y))))}}) & \text{by Axiom 3 LR} \\
 & = & \mathrm{f}(\mathrm{v}, \underline{\mathrm{f}(\mathrm{y}, \mathbf{1})}) & \text{by Axiom 2 LR} \\
 & = & \mathrm{f}(\mathrm{v}, \mathbf{y}) & \text{by Axiom 1 LR}
\end{array}$$

Lemma 2: f(1, u) = u

$$\begin{array}{llll}
\underline{\mathrm{f}(1, \mathrm{u})} & = & \mathbf{f(\underline{1},\ inv(inv(u)))} & \text{by Lemma 1 RL} \\
 & = & \underline{\mathrm{f}(\mathbf{f(u,\ inv(u))}, \mathrm{inv}(\mathrm{inv}(\mathrm{u})))} & \text{by Axiom 2 RL} \\
 & = & \mathbf{f(u,\ \underline{f(inv(u),\ inv(inv(u)))})} & \text{by Axiom 3 LR} \\
 & = & \underline{\mathrm{f}(\mathrm{u}, \mathbf{1})} & \text{by Axiom 2 LR} \\
 & = & \mathbf{u} & \text{by Axiom 1 LR}
\end{array}$$

Lemma 3: inv(inv(y)) = y

$$\begin{array}{llll}
\underline{\mathrm{inv}(\mathrm{inv}(\mathrm{y}))} & = & \underline{\mathbf{f(1,\ inv(inv(y)))}} & \text{by Lemma 2 RL} \\
 & = & \underline{\mathbf{f(1,\ y)}} & \text{by Lemma 1 LR} \\
 & = & \mathbf{y} & \text{by Lemma 2 LR}
\end{array}$$

Lemma 4: f(inv(y), f(y, z)) = z

$$\begin{array}{llll}
\mathrm{f}(\mathrm{inv}(\mathrm{y}), \mathrm{f}(\underline{\mathrm{y}}, \mathrm{z})) & = & \underline{\mathrm{f}(\mathrm{inv}(\mathrm{y}), \mathrm{f}(\mathbf{inv(inv(y))}, \mathrm{z}))} & \text{by Lemma 3 RL} \\
 & = & \mathbf{f(\underline{f(inv(y),\ inv(inv(y)))},\ z)} & \text{by Axiom 3 RL} \\
 & = & \underline{\mathrm{f}(\mathbf{1}, \mathrm{z})} & \text{by Axiom 2 LR} \\
 & = & \mathbf{z} & \text{by Lemma 2 LR}
\end{array}$$

Theorem 1: inv(1) = 1

$$
\begin{aligned}
\underline{\mathrm{inv}(1)} &= \underline{\mathbf{f(1,\ inv(1))}} && \text{by Lemma 2 RL}\\
&= \mathbf{1} && \text{by Axiom 2 LR}
\end{aligned}
$$

Theorem 2: inv(inv(y)) = y

$$
\begin{aligned}
\underline{\mathrm{inv}(\mathrm{inv}(y))} &= \mathbf{y} && \text{by Lemma 3 LR}
\end{aligned}
$$

Theorem 3: inv(f(v, w)) = f(inv(w), inv(v))

$$
\begin{aligned}
\underline{\mathrm{inv}(f(v,\ w))} &= \mathbf{f(inv(w),\ \underline{f(w,\ inv(f(v,\ w)))})} && \text{by Lemma 4 RL}\\
&= \mathrm{f(inv(w),\ }\mathbf{f(inv(v),\ \underline{f(v,\ f(w,\ inv(f(v,\ w))))})}) && \text{by Lemma 4 RL}\\
&= \mathrm{f(inv(w),\ f(inv(v),\ }\underline{\mathbf{f(f(v,\ w),\ inv(f(v,\ w)))}})) && \text{by Axiom 3 RL}\\
&= \mathrm{f(inv(w),\ }\underline{\mathrm{f(inv(v),\ }\mathbf{1})}) && \text{by Axiom 2 LR}\\
&= \mathrm{f(inv(w),\ }\mathbf{inv(v)}) && \text{by Axiom 1 LR}
\end{aligned}
$$

Theorem 4: f(x, 1) = x

$$
\begin{aligned}
\underline{f(x,\ 1)} &= \mathbf{x} && \text{by Axiom 1 LR}
\end{aligned}
$$

Theorem 5: f(1, u) = u

$$
\begin{aligned}
\underline{f(1,\ u)} &= \mathbf{u} && \text{by Lemma 2 LR}
\end{aligned}
$$

Theorem 6: f(x, inv(x)) = 1

$$
\begin{aligned}
\underline{f(x,\ \mathrm{inv}(x))} &= \mathbf{1} && \text{by Axiom 2 LR}
\end{aligned}
$$

Theorem 7: f(inv(y), y) = 1

$$
\begin{aligned}
f(\mathrm{inv}(y),\ \underline{y}) &= \underline{\mathrm{f(inv(y),\ }\mathbf{inv(inv(y))})} && \text{by Lemma 3 RL}\\
&= \mathbf{1} && \text{by Axiom 2 LR}
\end{aligned}
$$

Theorem 8: f(f(x, y), z) = f(x, f(y, z))

$$
\begin{aligned}
\underline{f(f(x,\ y),\ z)} &= \mathbf{f(x,\ f(y,\ z))} && \text{by Axiom 3 LR}
\end{aligned}
$$

Theorem 9: f(u, f(inv(u), z)) = z

$$
\begin{aligned}
\underline{f(u,\ f(\mathrm{inv}(u),\ z))} &= \mathbf{f(\underline{f(u,\ inv(u))},\ z)} && \text{by Axiom 3 RL}\\
&= \underline{\mathrm{f(}\mathbf{1}\mathrm{,\ z)}} && \text{by Axiom 2 LR}\\
&= \mathbf{z} && \text{by Lemma 2 LR}
\end{aligned}
$$

Theorem 10: f(inv(y), f(y, z)) = z

$$
\begin{aligned}
\underline{f(\mathrm{inv}(y),\ f(y,\ z))} &= \mathbf{z} && \text{by Lemma 4 LR}
\end{aligned}
$$

A.3 Ein Beispiel aus der Bool'schen Algebra

Auch in Axiomatisierungen von Bool'schen Algebren können Gleichheitsbeweiser viele Probleme lösen. Bekannte Beispiele für Sätze in der Bool'schen Algebra sind die Sätze von DeMorgan. Im folgenden wird einer dieser Sätze bewiesen.

```
MODE        PROOF

NAME        bool7

ORDERING    LPO
            n > a > o > 1 > 0 > y0 > x0

# ORDERING    LPO
#             a > o > n > 1 > 0 > y0 > x0

EQUATIONS   o (x,y) = o (y,x)                  # Kommutativitaet
            a (x,y) = a (y,x)

            a (x,o (y,z)) = o (a (x,y),a (x,z)) # Distributivitaet
            o (x,a (y,z)) = a (o (x,y),o (x,z))

            o (x,0) = x                        # Neutrale Elemente
            a (x,1) = x

            a (x,n (x)) = 0                    # Komplement
            o (x,n (x)) = 1

CONCLUSION  a (n (x0),n (y0)) = n (o (x0,y0))   # DeMorgan
```

Auf den Abdruck des PCL-Protokolles soll hier auf Grund der Länge des Protokolles verzichtet werden. Lemmata wurden mit den Optionen `-criteria toi -u_min_length 3` gesucht, es sind also isolierte Teilergebnisse, oft benutzte Schritte und wichtige Schritte als Lemmata gewählt worden. Als Randbedingung wurde gefordert, daß die Länge der Teilbeweise wenigstens 3 Schritte beträgt. Damit ergab sich der folgende Beweis.

Consider the following set of axioms:

Axiom 1: o(x, y) = o(y, x)
Axiom 2: a(x, y) = a(y, x)
Axiom 3: a(x, o(y, z)) = o(a(x, y), a(x, z))
Axiom 4: o(x, a(y, z)) = a(o(x, y), o(x, z))
Axiom 5: o(x, 0) = x
Axiom 6: a(x, 1) = x
Axiom 7: a(x, n(x)) = 0
Axiom 8: o(x, n(x)) = 1

This theorem holds true:

Theorem 1: $a(n(x0), n(y0)) = n(o(x0, y0))$

Proof:

Lemma 1: $o(a(x, u), a(x, n(u))) = x$

$$\begin{aligned} \underline{o(a(x, u), a(x, n(u)))} &= \mathbf{a(x, \underline{o(u, n(u))})} && \text{by Axiom 3 RL}\\ &= \underline{a(x, \mathbf{1})} && \text{by Axiom 8 LR}\\ &= \mathbf{x} && \text{by Axiom 6 LR}\end{aligned}$$

Lemma 2: $o(o(x, a(x, y)), a(o(x, y), z)) = o(x, a(y, z))$

$$\begin{aligned} & o(o(\underline{x}, a(x, y)), a(o(x, y), z))\\ &= o(o(\mathbf{o(a(x, x), \underline{a(x, n(x))})}, a(x, y)), a(o(x, y), z)) && \text{by Lemma 1 RL}\\ &= o(o(\underline{o(a(x, x), \mathbf{0})}, a(x, y)), a(o(x, y), z)) && \text{by Axiom 7 LR}\\ &= o(\underline{o(\mathbf{a(x, x)}, a(x, y))}, a(o(x, y), z)) && \text{by Axiom 5 LR}\\ &= o(\underline{\mathbf{a(x, o(x, y))}}, a(o(x, y), z)) && \text{by Axiom 3 RL}\\ &= \underline{o(\mathbf{a(o(x, y), x)}, a(o(x, y), z))} && \text{by Axiom 2 RL}\\ &= \underline{\mathbf{a(o(x, y), o(x, z))}} && \text{by Axiom 3 RL}\\ &= \mathbf{o(x, a(y, z))} && \text{by Axiom 4 RL}\end{aligned}$$

Lemma 3: $n(n(u)) = u$

$$\begin{aligned} \underline{n(n(u))} &= \mathbf{o(a(n(n(u)), u), \underline{a(n(n(u)), n(u))})} && \text{by Lemma 1 RL}\\ &= o(a(n(n(u)), u), \underline{\mathbf{a(n(u), n(n(u)))}}) && \text{by Axiom 2 RL}\\ &= o(\underline{a(n(n(u)), u)}, \mathbf{0}) && \text{by Axiom 7 LR}\\ &= \underline{o(\mathbf{a(u, n(n(u)))}, 0)} && \text{by Axiom 2 LR}\\ &= \mathbf{o(\underline{0}, a(u, n(n(u))))} && \text{by Axiom 1 LR}\\ &= \underline{o(\mathbf{a(u, n(u))}, a(u, n(n(u))))} && \text{by Axiom 7 RL}\\ &= \mathbf{u} && \text{by Lemma 1 LR}\end{aligned}$$

Lemma 4: $o(u, a(n(u), z)) = o(u, z)$

$$\begin{aligned} \underline{o(u, a(n(u), z))} &= \mathbf{o(o(u, \underline{a(u, n(u))}), a(o(u, n(u)), z))} && \text{by Lemma 2 RL}\\ &= o(\underline{o(u, \mathbf{0})}, a(o(u, n(u)), z)) && \text{by Axiom 7 LR}\\ &= o(\mathbf{u}, a(\underline{o(u, n(u))}, z)) && \text{by Axiom 5 LR}\\ &= o(u, \underline{a(\mathbf{1}, z)}) && \text{by Axiom 8 LR}\\ &= o(u, \underline{\mathbf{a(z, 1)}}) && \text{by Axiom 2 RL}\\ &= o(u, \mathbf{z}) && \text{by Axiom 6 LR}\end{aligned}$$

Lemma 5: o(x, o(x, y)) = o(x, y)

$$\begin{array}{rcll}
\mathrm{o(x,\ \underline{o(x,\ y)})} & = & \mathrm{o(\underline{x},\ \mathbf{a(o(x,\ y),\ 1)})} & \text{by Axiom 6 RL} \\
 & = & \mathrm{o(\mathbf{a(x,\ \underline{1})},\ a(o(x,\ y),\ 1))} & \text{by Axiom 6 RL} \\
 & = & \mathrm{o(a(x,\ \mathbf{o(y,\ \underline{n(y)})}),\ a(o(x,\ y),\ 1))} & \text{by Axiom 8 RL} \\
 & = & \mathrm{o(a(x,\ \underline{o(y,\ \mathbf{a(n(y),\ 1)})}),\ a(o(x,\ y),\ 1))} & \text{by Axiom 6 RL} \\
 & = & \mathrm{o(\underline{a(x,\ \mathbf{o(y,\ 1)})},\ a(o(x,\ y),\ 1))} & \text{by Lemma 4 LR} \\
 & = & \mathrm{o(\underline{\mathbf{o(a(x,\ y),\ a(x,\ 1))}},\ a(o(x,\ y),\ 1))} & \text{by Axiom 3 LR} \\
 & = & \mathrm{o(\underline{o(a(x,\ y),\ \mathbf{x})},\ a(o(x,\ y),\ 1))} & \text{by Axiom 6 LR} \\
 & = & \mathrm{\underline{o(\mathbf{o(x,\ a(x,\ y))},\ a(o(x,\ y),\ 1))}} & \text{by Axiom 1 LR} \\
 & = & \mathbf{o(x,\ \underline{a(y,\ 1)})} & \text{by Lemma 2 LR} \\
 & = & \mathrm{o(x,\ \mathbf{y})} & \text{by Axiom 6 LR}
\end{array}$$

Lemma 6: o(a(y, x), n(y)) = o(x, n(y))

$$\begin{array}{rcll}
\mathrm{o(a(\underline{y},\ x),\ n(y))} & = & \mathrm{\underline{o(a(\mathbf{n(n(y))},\ x),\ n(y))}} & \text{by Lemma 3 RL} \\
 & = & \underline{\mathbf{o(n(y),\ a(n(n(y)),\ x))}} & \text{by Axiom 1 LR} \\
 & = & \underline{\mathbf{o(n(y),\ x)}} & \text{by Lemma 4 LR} \\
 & = & \mathbf{o(x,\ n(y))} & \text{by Axiom 1 LR}
\end{array}$$

Lemma 7: a(o(p, n(y)), y) = a(y, p)

$$\begin{array}{rcll}
\mathrm{\underline{a(o(p,\ n(y)),\ y)}} & = & \underline{\mathbf{a(y,\ o(p,\ n(y)))}} & \text{by Axiom 2 RL} \\
 & = & \mathbf{o(a(y,\ p),\ \underline{a(y,\ n(y))})} & \text{by Axiom 3 LR} \\
 & = & \mathrm{\underline{o(a(y,\ p),\ \mathbf{0})}} & \text{by Axiom 7 LR} \\
 & = & \mathbf{a(y,\ p)} & \text{by Axiom 5 LR}
\end{array}$$

Lemma 8: a(v, a(v, x)) = a(v, x)

$$\begin{array}{rcll}
\mathrm{\underline{a(v,\ a(v,\ x))}} & = & \mathbf{a(\underline{o(a(v,\ x),\ n(v))},\ v)} & \text{by Lemma 7 RL} \\
 & = & \mathrm{\underline{a(\mathbf{o(x,\ n(v))},\ v)}} & \text{by Lemma 6 LR} \\
 & = & \mathbf{a(v,\ x)} & \text{by Lemma 7 LR}
\end{array}$$

Lemma 9: a(u, a(v, u)) = a(u, v)

$$\begin{array}{rcll}
\mathrm{\underline{a(u,\ a(v,\ u))}} & = & \mathbf{a(o(\underline{a(v,\ u)},\ n(u)),\ u)} & \text{by Lemma 7 RL} \\
 & = & \mathrm{a(\underline{o(\mathbf{a(u,\ v)},\ n(u))},\ u)} & \text{by Axiom 2 RL} \\
 & = & \mathrm{\underline{a(\mathbf{o(v,\ n(u))},\ u)}} & \text{by Lemma 6 LR} \\
 & = & \underline{\mathbf{a(u,\ o(v,\ n(u)))}} & \text{by Axiom 2 LR} \\
 & = & \mathbf{o(a(u,\ v),\ \underline{a(u,\ n(u))})} & \text{by Axiom 3 LR} \\
 & = & \mathrm{\underline{o(a(u,\ v),\ \mathbf{0})}} & \text{by Axiom 7 LR} \\
 & = & \mathbf{a(u,\ v)} & \text{by Axiom 5 LR}
\end{array}$$

Lemma 10: a(w, n(a(v, n(w)))) = w

a(w, n(a(v, n(w)))) = **o(a(w, n(a(v, n(w)))), 0)**
by Axiom 5 RL
= **o(0, a(w, n(a(v, n(w)))))**
by Axiom 1 LR
= o(**a(w, n(w))**, a(w, n(a(v, n(w)))))
by Axiom 7 RL
= o(**a(n(w), w)**, a(w, n(a(v, n(w)))))
by Axiom 2 LR
= o(a(**o(a(n(w), v), a(n(w), n(v)))**, w), a(w, n(a(v, n(w)))))
by Lemma 1 RL
= o(a(o(**a(v, n(w))**, a(n(w), n(v))), w), a(w, n(a(v, n(w)))))
by Axiom 2 LR
= o(a(**o(a(v, n(w)), o(a(v, n(w)), a(n(w), n(v))))**, w), a(w, n(a(v, n(w)))))
by Lemma 5 RL
= o(a(o(a(v, n(w)), o(**a(n(w), v)**, a(n(w), n(v)))), w), a(w, n(a(v, n(w)))))
by Axiom 2 RL
= o(a(o(a(v, n(w)), **n(w)**), w), a(w, n(a(v, n(w)))))
by Lemma 1 LR
= o(**a(w, a(v, n(w)))**, a(w, n(a(v, n(w)))))
by Lemma 7 LR
= **w**
by Lemma 1 LR

Lemma 11: a(o(a(w, n(u)), u), n(a(w, n(u)))) = u

a(o(a(w, n(u)), u), n(a(w, n(u))))
= **a(n(a(w, n(u))), o(a(w, n(u)), u))** by Axiom 2 RL
= **o(a(n(a(w, n(u))), a(w, n(u))), a(n(a(w, n(u))), u))** by Axiom 3 LR
= o(**a(a(w, n(u)), n(a(w, n(u))))**, a(n(a(w, n(u))), u)) by Axiom 2 RL
= o(**0**, a(n(a(w, n(u))), u)) by Axiom 7 LR
= **o(a(n(a(w, n(u))), u), 0)** by Axiom 1 RL
= **a(n(a(w, n(u))), u)** by Axiom 5 LR
= **a(n(a(w, n(u))), a(u, n(a(w, n(u)))))** by Lemma 9 RL
= **a(a(u, n(a(w, n(u)))), n(a(w, n(u))))** by Axiom 2 RL
= **a(a(u, n(a(w, n(u)))), a(n(a(w, n(u))), a(u, n(a(w, n(u))))))** by Lemma 9 RL
= a(a(u, n(a(w, n(u)))), **a(n(a(w, n(u))), u)**) by Lemma 9 LR
= a(**u**, a(n(a(w, n(u))), u)) by Lemma 10 LR
= **a(u, n(a(w, n(u))))** by Lemma 9 LR
= **u** by Lemma 10 LR

Lemma 12: a(v, n(o(v, y))) = 0

a(v, n(o(v, y)))	=	**a(n(o(v, y)), v)**	by Axiom 2 LR
	=	**o(a(n(o(v, y)), v), 0)**	by Axiom 5 RL
	=	o(a(n(o(v, y)), v), **a(o(v, y), n(o(v, y)))**)	by Axiom 7 RL
	=	o(a(n(o(v, y)), v), **a(n(o(v, y)), o(v, y))**)	by Axiom 2 LR
	=	**a(n(o(v, y)), o(v, o(v, y)))**	by Axiom 3 RL
	=	**a(o(v, o(v, y)), n(o(v, y)))**	by Axiom 2 LR
	=	a(**o(v, y)**, n(o(v, y)))	by Lemma 5 LR
	=	**0**	by Axiom 7 LR

Lemma 13: a(o(u, w), n(a(w, n(u)))) = u

a(o(u, w), n(a(w, n(u))))

=	a(o(**n(n(u))**, w), n(a(w, n(u))))	by Lemma 3 RL
=	a(**o(n(n(u)), a(n(n(u)), w))**, n(a(w, n(u))))	by Lemma 4 RL
=	a(o(n(n(u)), **a(w, n(n(u)))**), n(a(w, n(u))))	by Axiom 2 LR
=	a(o(n(n(u)), a(w, **n(u)**)), n(a(w, n(u))))	by Lemma 3 LR
=	a(**o(a(w, n(u)), n(n(u)))**, n(a(w, n(u))))	by Axiom 1 LR
=	a(o(a(w, n(u)), **u**), n(a(w, n(u))))	by Lemma 3 LR
=	**u**	by Lemma 11 LR

Lemma 14: n(a(v, n(u))) = o(a(v, u), a(n(a(v, n(u))), n(v)))

n(a(v, n(u))) = **o(a(n(a(v, n(u))), v), a(n(a(v, n(u))), n(v)))**
by Lemma 1 RL

= o(**a(v, n(a(v, n(u))))**, a(n(a(v, n(u))), n(v)))
by Axiom 2 LR

= o(**o(a(v, n(a(v, n(u)))), 0)**, a(n(a(v, n(u))), n(v)))
by Axiom 5 RL

= o(o(a(v, n(a(v, n(u)))), **a(v, n(o(v, u)))**), a(n(a(v, n(u))), n(v)))
by Lemma 12 RL

= o(**a(v, o(n(a(v, n(u))), n(o(v, u))))**, a(n(a(v, n(u))), n(v)))
by Axiom 3 RL

= o(**a(o(n(a(v, n(u))), n(o(v, u))), v)**, a(n(a(v, n(u))), n(v)))
by Axiom 2 RL

= o(a(**o(a(o(v, u), n(a(v, n(u)))), n(o(v, u)))**, v), a(n(a(v, n(u))), n(v)))
by Lemma 6 RL

= o(**a(v, o(a(o(v, u), n(a(v, n(u)))), n(o(v, u))))**, a(n(a(v, n(u))), n(v)))
by Axiom 2 RL

= o(**o(a(v, a(o(v, u), n(a(v, n(u))))), a(v, n(o(v, u))))**, a(n(a(v, n(u))), n(v)))
by Axiom 3 LR

= o(o(a(v, a(o(v, u), n(a(v, n(u))))), **0**), a(n(a(v, n(u))), n(v)))
by Lemma 12 LR

= o(**a(v, a(o(v, u), n(a(v, n(u)))))**, a(n(a(v, n(u))), n(v)))
by Axiom 5 LR

= o(a(v, a(**o(u, v)**, n(a(v, n(u))))), a(n(a(v, n(u))), n(v)))
by Axiom 1 RL

= o(a(v, **u**), a(n(a(v, n(u))), n(v)))
by Lemma 13 LR

Lemma 15: a(v, y) = n(o(n(y), n(v)))

a(v, y) = **n(n(a(v, y)))**
by Lemma 3 RL
= n(n(a(v, **n(n(y))**)))
by Lemma 3 RL
= n(**o(a(v, n(y)), a(n(a(v, n(n(y)))), n(v)))**)
by Lemma 14 LR
= n(o(a(v, n(y)), **n(n(a(n(a(v, n(n(y)))), n(v))))**))
by Lemma 3 RL
= n(o(a(v, n(y)), n(**a(n(a(n(a(v, n(n(y)))), n(v))), 1)**)))
by Axiom 6 RL
= n(o(a(v, n(y)), n(**a(1, n(a(n(a(v, n(n(y)))), n(v))))**)))
by Axiom 2 LR
= n(o(a(v, n(y)), n(a(**o(a(v, n(n(y))), n(a(v, n(n(y)))))**, n(a(n(a(v, n(n(y)))), n(v)))))))
by Axiom 8 RL
= n(o(a(v, n(y)), n(a(o(**a(v, a(v, n(n(y))))**, n(a(v, n(n(y))))), n(a(n(a(v, n(n(y)))), n(v)))))))
by Lemma 8 RL
= n(o(a(v, n(y)), n(a(o(**a(a(v, n(n(y))), v)**, n(a(v, n(n(y))))), n(a(n(a(v, n(n(y)))), n(v)))))))
by Axiom 2 RL
= n(o(a(v, n(y)), n(a(**o(v, n(a(v, n(n(y)))))**, n(a(n(a(v, n(n(y)))), n(v)))))))
by Lemma 6 LR
= n(o(a(v, n(y)), n(**v**)))
by Lemma 13 LR
= n(**o(n(y), n(v))**)
by Lemma 6 LR

Lemma 16: a(n(y), n(z)) = n(o(z, y))

a(n(y), n(z))	=	**n(o(n(n(z)), n(n(y))))**	by Lemma 15 LR
	=	n(o(**z**, n(n(y))))	by Lemma 3 LR
	=	n(o(z, **y**))	by Lemma 3 LR

Theorem 1: a(n(x0), n(y0)) = n(o(x0, y0))

a(n(x0), n(y0))	=	**a(n(y0), n(x0))**	by Axiom 2 LR
	=	**n(o(x0, y0))**	by Lemma 16 LR

A.4 Ein großes Beispiel aus der Ringtheorie

Dieses Beispiel wird in [LO82] als ein sehr anspruchsvolles Problem vorgestellt. Es ist von besonderem Interesse, weil bisher kein Beweis, der ausschließlich auf Rewrite-Techniken ohne unterliegende AC-Theorie basiert, publiziert ist. Ein solcher Beweis gelang dem DISCOUNT-System auf einem SUN4/370 Server in 8188.2 Sekunden. Nähere Einzelheiten dieses Beweislaufes befinden sich in [Pi92]. Bei einem späteren Lauf wurde ein Protokoll erzeugt, aus dem der hier gezeigte Beweis gewonnen wurde. Zu beweisen ist, daß ein Ring, in dem $x^3 = x$ für alle x gilt, kommutativ ist.

```
MODE        PROOF

NAME        lusk6

ORDERING    XKBO
            f:5 > j:4 > g:3 > 0:1 > b:1 > a:1

EQUATIONS   j (0,x)       = x                   # 0 ist a left identity for sum
            j (x,0)       = x                   # 0 ist a right identity for sum
            j (g (x),x)   = 0                   # there exists a left inverse for sum
            j (x,g (x))   = 0                   # there exists a right inverse for sum
            j (j (x,y),z) = j (x,j (y,z))       # associativity of addition
            j (x,y)       = j(y,x)              # commutativity of addition
            f (f (x,y),z) = f (x,f (y,z))       # associativity of multiplication
            f (x,j (y,z)) = j (f (x,y),f (x,z)) # distributivity axioms
            f (j (x,y),z) = j (f (x,z),f (y,z)) #
            f (f(x,x),x) = x                    # special hypothesis

CONCLUSION  f (a,b) = f (b,a)                   # theorem
```

Das Beweisprotokoll für dieses Beispiel umfaßt knapp 400000 Schritte, von denen nur 190 im Beweis verwendet werden. Der Beweis wurde nicht mit den Default-Einstellungen strukturiert. Das Programm `lemma` wude mit den Optionen `-criteria toi -u_min_length 3 -iterate` aufgerufen, es wurden also das Protokoll in mehreren Phasen nach isolierten Teilbeweisen, oft verwendeten Lemmata und wichtigen Lemmata durchsucht. Zusätzlich wurde für Lemmata eine Mindestbeweislänge von 3 Schritten gefordert. Damit ergab sich der folgende Beweis:

Consider the following set of axioms:

Axiom 1: j(0, x) = x
Axiom 2: j(x, 0) = x
Axiom 3: j(x, g(x)) = 0
Axiom 4: j(j(x, y), z) = j(x, j(y, z))
Axiom 5: j(x, y) = j(y, x)
Axiom 6: f(f(x, y), z) = f(x, f(y, z))
Axiom 7: f(x, j(y, z)) = j(f(x, y), f(x, z))
Axiom 8: f(j(x, y), z) = j(f(x, z), f(y, z))

Axiom 9: f(f(x, x), x) = x

This theorem holds true:

Theorem 1: f(a, b) = f(b, a)

Proof:

Lemma 1: j(u, j(g(u), z)) = z

$$\begin{aligned}
j(u, j(g(u), z)) &= j(j(u, g(u)), z) && \text{by Axiom 4 RL}\\
&= j(0, z) && \text{by Axiom 3 LR}\\
&= z && \text{by Axiom 1 LR}
\end{aligned}$$

Lemma 2: g(g(u)) = u

$$\begin{aligned}
g(g(u)) &= j(u, j(g(u), g(g(u)))) && \text{by Lemma 1 RL}\\
&= j(u, 0) && \text{by Axiom 3 LR}\\
&= u && \text{by Axiom 2 LR}
\end{aligned}$$

Lemma 3: j(v, g(j(w, v))) = g(w)

$$\begin{aligned}
j(v, g(j(w, v))) &= j(g(j(w, v)), v) && \text{by Axiom 5 RL}\\
&= j(g(j(w, v)), j(g(w), j(g(g(w)), v))) && \text{by Lemma 1 RL}\\
&= j(g(j(w, v)), j(g(w), j(w, v))) && \text{by Lemma 2 LR}\\
&= j(g(j(w, v)), j(g(w), g(g(j(w, v))))) && \text{by Lemma 2 RL}\\
&= j(g(j(w, v)), j(g(g(j(w, v))), g(w))) && \text{by Axiom 5 RL}\\
&= g(w) && \text{by Lemma 1 LR}
\end{aligned}$$

Lemma 4: f(y, 0) = 0

$$\begin{aligned}
f(y, 0) &= f(g(g(y)), 0) && \text{by Lemma 2 RL}\\
&= j(g(y), j(g(g(y)), f(g(g(y)), 0))) && \text{by Lemma 1 RL}\\
&= j(g(y), j(f(f(g(g(y)), g(g(y))), g(g(y))), f(g(g(y)), 0))) && \text{by Axiom 9 RL}\\
&= j(g(y), j(f(g(g(y)), f(g(g(y)), g(g(y)))), f(g(g(y)), 0))) && \text{by Axiom 6 LR}\\
&= j(g(y), f(g(g(y)), j(f(g(g(y)), g(g(y))), 0))) && \text{by Axiom 7 RL}\\
&= j(g(y), f(g(g(y)), f(g(g(y)), g(g(y))))) && \text{by Axiom 2 LR}\\
&= j(g(y), f(f(g(g(y)), g(g(y))), g(g(y)))) && \text{by Axiom 6 RL}\\
&= j(g(y), g(g(y))) && \text{by Axiom 9 LR}\\
&= 0 && \text{by Axiom 3 LR}
\end{aligned}$$

Lemma 5: $f(0, y) = 0$

$$\begin{aligned}
f(0, y) &= f(0, j(y, 0)) && \text{by Axiom 2 RL}\\
&= f(0, j(y, j(y, g(y)))) && \text{by Axiom 3 RL}\\
&= f(0, j(j(y, y), g(y))) && \text{by Axiom 4 RL}\\
&= j(f(0, j(y, y)), f(0, g(y))) && \text{by Axiom 7 LR}\\
&= j(j(f(0, y), f(0, y)), f(0, g(y))) && \text{by Axiom 7 LR}\\
&= j(f(j(0, 0), y), f(0, g(y))) && \text{by Axiom 8 RL}\\
&= j(f(0, y), f(0, g(y))) && \text{by Axiom 1 LR}\\
&= f(0, j(y, g(y))) && \text{by Axiom 7 RL}\\
&= f(0, 0) && \text{by Axiom 3 LR}\\
&= 0 && \text{by Lemma 4 LR}
\end{aligned}$$

Lemma 6: $f(z, f(y, f(z, f(y, f(z, y))))) = f(z, y)$

$$\begin{aligned}
f(z, f(y, f(z, f(y, f(z, y))))) &= f(z, f(y, f(f(z, y), f(z, y)))) && \text{by Axiom 6 RL}\\
&= f(f(z, y), f(f(z, y), f(z, y))) && \text{by Axiom 6 RL}\\
&= f(f(f(z, y), f(z, y)), f(z, y)) && \text{by Axiom 6 RL}\\
&= f(z, y) && \text{by Axiom 9 LR}
\end{aligned}$$

Lemma 7: $f(p, j(f(p, f(p, v)), z)) = f(p, j(v, z))$

$$\begin{aligned}
f(p, j(f(p, f(p, v)), z)) &= j(f(p, f(p, f(p, v))), f(p, z)) && \text{by Axiom 7 LR}\\
&= j(f(p, f(f(p, p), v)), f(p, z)) && \text{by Axiom 6 RL}\\
&= j(f(f(p, f(p, p)), v), f(p, z)) && \text{by Axiom 6 RL}\\
&= j(f(f(f(p, p), p), v), f(p, z)) && \text{by Axiom 6 RL}\\
&= j(f(p, v), f(p, z)) && \text{by Axiom 9 LR}\\
&= f(p, j(v, z)) && \text{by Axiom 7 RL}
\end{aligned}$$

Lemma 8: $f(j(v, g(f(x, f(x, v)))), x) = 0$

$$\begin{aligned}
&f(j(v, g(f(x, f(x, v)))), x)\\
&\quad = f(j(v, g(f(x, f(x, v)))), f(x, f(j(v, g(f(x, f(x, v)))), f(x, f(j(v, g(f(x, f(x, v)))), x)))))\\
&\quad\quad \text{by Lemma 6 RL}\\
&\quad = f(j(v, g(f(x, f(x, v)))), f(f(x, j(v, g(f(x, f(x, v))))), f(x, f(j(v, g(f(x, f(x, v)))), x))))\\
&\quad\quad \text{by Axiom 6 RL}\\
&\quad = f(j(v, g(f(x, f(x, v)))), f(f(x, j(f(x, f(x, v)), g(f(x, f(x, v))))), f(x, f(j(v, g(f(x, f(x, v)))), x))))\\
&\quad\quad \text{by Lemma 7 RL}\\
&\quad = f(j(v, g(f(x, f(x, v)))), f(f(x, 0), f(x, f(j(v, g(f(x, f(x, v)))), x))))\\
&\quad\quad \text{by Axiom 3 LR}\\
&\quad = f(j(v, g(f(x, f(x, v)))), f(0, f(x, f(j(v, g(f(x, f(x, v)))), x))))\\
&\quad\quad \text{by Lemma 4 LR}\\
&\quad = f(j(v, g(f(x, f(x, v)))), 0)\\
&\quad\quad \text{by Lemma 5 LR}\\
&\quad = 0\\
&\quad\quad \text{by Lemma 4 LR}
\end{aligned}$$

Lemma 9: f(z, g(u)) = g(f(z, u))

f(z, g(u))	=	j(0, f(z, g(u)))	by Axiom 1 RL
	=	j(f(z, g(u)), 0)	by Axiom 5 RL
	=	j(f(z, g(u)), j(0, g(0)))	by Axiom 3 RL
	=	j(f(z, g(u)), g(0))	by Axiom 1 LR
	=	j(f(z, g(u)), g(f(z, 0)))	by Lemma 4 RL
	=	j(f(z, g(u)), g(f(z, j(u, g(u)))))	by Axiom 3 RL
	=	j(f(z, g(u)), g(j(f(z, u), f(z, g(u)))))	by Axiom 7 LR
	=	g(f(z, u))	by Lemma 3 LR

Lemma 10: f(g(u), y) = g(f(u, y))

f(g(u), y)	=	j(0, f(g(u), y))	by Axiom 1 RL
	=	j(f(g(u), y), 0)	by Axiom 5 RL
	=	j(f(g(u), y), j(0, g(0)))	by Axiom 3 RL
	=	j(f(g(u), y), g(0))	by Axiom 1 LR
	=	j(f(g(u), y), g(f(0, y)))	by Lemma 5 RL
	=	j(f(g(u), y), g(f(j(u, g(u)), y)))	by Axiom 3 RL
	=	j(f(g(u), y), g(j(f(u, y), f(g(u), y))))	by Axiom 8 LR
	=	g(f(u, y))	by Lemma 3 LR

Lemma 11: g(f(u, w)) = f(w, f(w, g(f(u, w))))

g(f(u, w))	=	j(f(g(f(w, f(w, u))), w), g(j(f(u, w), f(g(f(w, f(w, u))), w))))	by Lemma 3 RL
	=	j(f(g(f(w, f(w, u))), w), g(f(j(u, g(f(w, f(w, u)))), w)))	by Axiom 8 RL
	=	j(f(g(f(w, f(w, u))), w), g(0))	by Lemma 8 LR
	=	j(f(f(w, g(f(w, u))), w), g(0))	by Lemma 9 RL
	=	j(f(w, f(g(f(w, u)), w)), g(0))	by Axiom 6 LR
	=	j(f(w, f(f(w, g(u)), w)), g(0))	by Lemma 9 RL
	=	j(f(w, f(w, f(g(u), w))), g(0))	by Axiom 6 LR
	=	j(f(w, f(w, f(g(u), w))), j(0, g(0)))	by Axiom 1 RL
	=	j(f(w, f(w, f(g(u), w))), 0)	by Axiom 3 LR
	=	j(0, f(w, f(w, f(g(u), w))))	by Axiom 5 LR
	=	f(w, f(w, f(g(u), w)))	by Axiom 1 LR
	=	f(w, f(w, g(f(u, w))))	by Lemma 10 LR

Lemma 12: f(v, f(v, f(z, v))) = f(z, v)

f(v, f(v, f(z, v)))	=	g(g(f(v, f(v, f(z, v)))))	by Lemma 2 RL
	=	g(f(v, g(f(v, f(z, v)))))	by Lemma 9 RL
	=	g(f(v, f(v, g(f(z, v)))))	by Lemma 9 RL
	=	g(g(f(z, v)))	by Lemma 11 RL
	=	f(z, v)	by Lemma 2 LR

Lemma 13: f(y, w) = f(w, f(y, f(w, f(y, f(w, f(w, f(y, w)))))))

f(y, w)	=	f(y, f(w, f(y, f(w, f(y, w)))))
		by Lemma 6 RL
	=	f(f(w, f(y, f(w, f(y, w)))), f(f(w, f(y, f(w, f(y, w)))), f(y, f(w, f(y, f(w, f(y, w)))))))
		by Lemma 12 RL
	=	f(f(w, f(y, f(w, f(y, w)))), f(f(w, f(y, f(w, f(y, w)))), f(y, w)))
		by Lemma 6 LR
	=	f(f(w, f(y, f(w, f(y, w)))), f(w, f(f(y, f(w, f(y, w))), f(y, w))))
		by Axiom 6 LR
	=	f(f(w, f(y, f(w, f(y, w)))), f(w, f(y, f(f(w, f(y, w)), f(y, w)))))
		by Axiom 6 LR
	=	f(f(w, f(y, f(w, f(y, w)))), f(w, f(y, f(w, f(f(y, w), f(y, w))))))
		by Axiom 6 LR
	=	f(f(w, f(y, f(w, f(y, w)))), f(w, f(y, f(w, f(y, f(w, f(y, w)))))))
		by Axiom 6 LR
	=	f(f(w, f(y, f(w, f(y, w)))), f(w, f(y, w)))
		by Lemma 6 LR
	=	f(w, f(f(y, f(w, f(y, w))), f(w, f(y, w))))
		by Axiom 6 LR
	=	f(w, f(y, f(f(w, f(y, w)), f(w, f(y, w)))))
		by Axiom 6 LR
	=	f(w, f(y, f(w, f(f(y, w), f(w, f(y, w))))))
		by Axiom 6 LR
	=	f(w, f(y, f(w, f(y, f(w, f(w, f(y, w)))))))
		by Axiom 6 LR

Lemma 14: f(j(p, f(p, g(f(x, x)))), x) = f(p, j(g(x), f(f(p, p), x)))

f(j(p, f(p, g(f(x, x)))), x)	=	f(j(f(f(p, p), p), f(p, g(f(x, x)))), x)	by Axiom 9 RL
	=	f(j(f(p, f(p, p)), f(p, g(f(x, x)))), x)	by Axiom 6 LR
	=	f(f(p, j(f(p, p), g(f(x, x)))), x)	by Axiom 7 RL
	=	f(p, f(j(f(p, p), g(f(x, x))), x))	by Axiom 6 LR
	=	f(p, f(j(f(p, p), j(g(f(p, p)), g(j(f(x, x), g(f(p, p)))))), x))	by Lemma 3 RL
	=	f(p, f(g(j(f(x, x), g(f(p, p)))), x))	by Lemma 1 LR
	=	f(p, g(f(j(f(x, x), g(f(p, p))), x)))	by Lemma 10 LR
	=	f(p, g(j(f(f(x, x), x), f(g(f(p, p)), x))))	by Axiom 8 LR
	=	f(p, g(j(x, f(g(f(p, p)), x))))	by Axiom 9 LR
	=	f(p, g(j(x, g(f(f(p, p), x)))))	by Lemma 10 LR
	=	f(p, j(f(f(p, p), x), j(g(f(f(p, p), x)), g(j(x, g(f(f(p, p), x)))))))	by Lemma 1 RL
	=	f(p, j(f(f(p, p), x), g(x)))	by Lemma 3 LR
	=	f(p, j(g(x), f(f(p, p), x)))	by Axiom 5 LR

Lemma 15: f(j(p, g(f(p, f(x, x)))), x) = 0

f(j(p, g(f(p, f(x, x)))), x)	=	f(j(p, f(p, g(f(x, x)))), x)	by Lemma 9 RL
	=	f(p, j(g(x), f(f(p, p), x)))	by Lemma 14 LR
	=	f(p, j(g(x), f(p, f(p, x))))	by Axiom 6 LR
	=	j(f(p, g(x)), f(p, f(p, f(p, x))))	by Axiom 7 LR
	=	j(f(p, g(x)), f(p, f(f(p, p), x)))	by Axiom 6 RL
	=	j(f(p, g(x)), f(f(p, f(p, p)), x))	by Axiom 6 RL
	=	j(f(p, g(x)), f(f(f(p, p), p), x))	by Axiom 6 RL
	=	j(f(p, g(x)), f(p, x))	by Axiom 9 LR
	=	f(p, j(g(x), x))	by Axiom 7 RL
	=	f(p, j(x, g(x)))	by Axiom 5 LR
	=	f(p, 0)	by Axiom 3 LR
	=	0	by Lemma 4 LR

Lemma 16: f(w, g(f(w, f(w, g(f(y, f(w, w))))))) = f(w, y)

f(w, g(f(w, f(w, g(f(y, f(w, w)))))))
= f(w, j(0, g(f(w, f(w, g(f(y, f(w, w))))))))
by Axiom 1 RL
= f(w, j(f(w, 0), g(f(w, f(w, g(f(y, f(w, w))))))))
by Lemma 4 RL
= f(w, j(f(w, f(j(w, g(f(j(y, g(f(y, f(w, w)))), f(j(y, g(f(y, f(w, w)))), w)))), j(y, g(f(y, f(w, w)))))), g(f(w, f(w, g(f(y, f(w, w))))))))
by Lemma 8 RL
= f(w, j(f(w, f(j(w, g(f(j(y, g(f(y, f(w, w)))), 0))), j(y, g(f(y, f(w, w)))))), g(f(w, f(w, g(f(y, f(w, w))))))))
by Lemma 15 LR
= f(w, j(f(w, f(j(w, g(0)), j(y, g(f(y, f(w, w)))))), g(f(w, f(w, g(f(y, f(w, w))))))))
by Lemma 4 LR
= f(w, j(f(w, f(j(w, j(0, g(0))), j(y, g(f(y, f(w, w)))))), g(f(w, f(w, g(f(y, f(w, w))))))))
by Axiom 1 RL
= f(w, j(f(w, f(j(w, 0), j(y, g(f(y, f(w, w)))))), g(f(w, f(w, g(f(y, f(w, w))))))))
by Axiom 3 LR
= f(w, j(f(w, f(w, j(y, g(f(y, f(w, w)))))), g(f(w, f(w, g(f(y, f(w, w))))))))
by Axiom 2 LR
= f(w, j(j(y, g(f(y, f(w, w)))), g(f(w, f(w, g(f(y, f(w, w))))))))
by Lemma 7 LR
= f(w, j(y, j(g(f(y, f(w, w))), g(f(w, f(w, g(f(y, f(w, w)))))))))
by Axiom 4 LR
= j(f(w, y), f(w, j(g(f(y, f(w, w))), g(f(w, f(w, g(f(y, f(w, w)))))))))
by Axiom 7 LR
= j(f(w, y), f(w, j(f(w, f(w, g(f(y, f(w, w))))), g(f(w, f(w, g(f(y, f(w, w)))))))))
by Lemma 7 RL
= j(f(w, y), f(w, 0))
by Axiom 3 LR
= j(f(w, y), 0)
by Lemma 4 LR
= j(0, f(w, y))
by Axiom 5 LR
= f(w, y)
by Axiom 1 LR

Lemma 17: f(w, f(y, f(w, w))) = f(w, y)

f(w, f(y, f(w, w)))	=	f(w, g(g(f(y, f(w, w)))))	by Lemma 2 RL
	=	f(w, g(g(f(f(y, w), w))))	by Axiom 6 RL
	=	f(w, g(g(f(w, f(w, f(f(y, w), w))))))	by Lemma 12 RL
	=	f(w, g(g(f(w, f(w, f(y, f(w, w)))))))	by Axiom 6 LR
	=	f(w, g(f(w, g(f(w, f(y, f(w, w)))))))	by Lemma 9 RL
	=	f(w, g(f(w, f(w, g(f(y, f(w, w)))))))	by Lemma 9 RL
	=	f(w, y)	by Lemma 16 LR

Lemma 18: $f(z, x) = f(f(z, f(x, x)), f(z, f(x, z)))$

$$\begin{array}{lcll}
f(z, x) & = & f(z, f(f(x, x), x)) & \text{by Axiom 9 RL} \\
 & = & f(f(z, f(x, x)), x) & \text{by Axiom 6 RL} \\
 & = & f(f(z, f(x, x)), f(x, f(f(z, f(x, x)), f(z, f(x, x))))) & \text{by Lemma 17 RL} \\
 & = & f(f(z, f(x, x)), f(x, f(f(f(z, f(x, x)), z), f(x, x)))) & \text{by Axiom 6 RL} \\
 & = & f(f(z, f(x, x)), f(x, f(f(z, f(x, x)), z))) & \text{by Lemma 17 LR} \\
 & = & f(f(z, f(x, x)), f(x, f(z, f(f(x, x), z)))) & \text{by Axiom 6 LR} \\
 & = & f(f(z, f(x, x)), f(x, f(z, f(x, f(x, z))))) & \text{by Axiom 6 LR} \\
 & = & f(f(z, f(x, x)), f(f(x, z), f(x, f(x, z)))) & \text{by Axiom 6 RL} \\
 & = & f(f(z, f(x, x)), f(f(z, f(z, f(x, z))), f(x, f(x, z)))) & \text{by Lemma 12 RL} \\
 & = & f(f(z, f(x, x)), f(z, f(f(z, f(x, z)), f(x, f(x, z))))) & \text{by Axiom 6 LR} \\
 & = & f(f(z, f(x, x)), f(z, f(z, f(f(x, z), f(x, f(x, z)))))) & \text{by Axiom 6 LR} \\
 & = & f(f(z, f(x, x)), f(z, f(z, f(x, f(z, f(x, f(x, z))))))) & \text{by Axiom 6 LR} \\
 & = & f(f(z, f(x, x)), f(z, f(z, f(x, f(z, f(x, f(z, f(z, f(x, z))))))))) & \text{by Lemma 12 RL} \\
 & = & f(f(z, f(x, x)), f(z, f(x, z))) & \text{by Lemma 13 RL}
\end{array}$$

Lemma 19: $f(z, x) = f(z, f(x, f(x, f(z, f(x, z)))))$

$$\begin{array}{lcll}
f(z, x) & = & f(f(z, f(x, x)), f(z, f(x, z))) & \text{by Lemma 18 LR} \\
 & = & f(z, f(f(x, x), f(z, f(x, z)))) & \text{by Axiom 6 LR} \\
 & = & f(z, f(x, f(x, f(z, f(x, z))))) & \text{by Axiom 6 LR}
\end{array}$$

Theorem 1: $f(a, b) = f(b, a)$

$$\begin{array}{lcll}
f(a, b) & = & f(a, f(b, f(b, f(a, f(b, a))))) & \text{by Lemma 19 LR} \\
 & = & f(a, f(b, f(b, f(f(a, b), a)))) & \text{by Axiom 6 RL} \\
 & = & f(a, f(b, f(f(b, f(a, b)), a))) & \text{by Axiom 6 RL} \\
 & = & f(a, f(f(b, f(b, f(a, b))), a)) & \text{by Axiom 6 RL} \\
 & = & f(a, f(f(a, b), a)) & \text{by Lemma 12 LR} \\
 & = & f(a, f(a, f(b, a))) & \text{by Axiom 6 LR} \\
 & = & f(b, a) & \text{by Lemma 12 LR}
\end{array}$$

A.5 Die Spezifikationen anderer verwendeter Probleme

A.5.1 Das Beispiel Lusk 2

Dieses einfache Problem stammt aus [LO82]. Es wird gezeigt, daß in jeder Gruppe das Inverse des Inversen eines Elementes wieder das Element selbst ist. Das PCL-Protokoll diese Beweises befindet sich in 3.3, ein Gleichheitsbeweis in 5.5.

```
MODE            PROOF

NAME            GroupLusk2

ORDERING        LPO
                g > f > e > a

EQUATIONS       f(e,x)      = x             # e is linksneutral
                f(x,e)      = x             # e is rechtsneutral
                f(g(x),x)   = e             # Es existiert ein Linksinverses
                f(x,g(x))   = e             # Es existiert ein Rinksinverses
                f(f(x,y),z) = f(x,f(y,z))   # f ist assoziativ

CONCLUSION      g(g(a)) = a                 # Theorem
```

A.5.2 Das Beispiel Self Inverse

Dieses Beispiel entstand aus Beobachtungen bei der Analyse des Beispieles aus A.1. Es verwendet ebenfalls eine Axiomatisierung eines Ringes mit der zusätzlichen Hypothese $x^2 = x$. Gezeigt wird, daß mit dieser Hypothese jedes Element sein eigenes Inverses ist. Das Beispiel wird in Tabelle 1 zitiert.

```
MODE        PROOF

NAME        SelfInverse

ORDERING    XKBO
            f:5 > j:4 > g:3 > 0:1 > b:1 > a:1  # > 0:1

EQUATIONS   j (0,x)       = x                   # 0 ist a left identity for sum
            j (g (x),x)   = 0                   # there exists a left inverse for sum
            j (j (x,y),z) = j (x,j (y,z))       # associativity of addition
            j (x,y)       = j(y,x)              # commutativity of addition
            f (f (x,y),z) = f (x,f (y,z))       # associativity of multiplication
            f (x,j (y,z)) = j (f (x,y),f (x,z)) # distributivity axioms
            f (j (x,y),z) = j (f (x,z),f (y,z)) #
            f (x,x)       = x                   # special hypothese: x*x = x

CONCLUSION  g(x) = x                            # theorem
```

A.5.3 Das Beispiel $6x = 0$

Dieses Beispiel verwendet im wesentlichen die selbe Axiomatisierung wie das Beispiel Lusk Overbeck 6 aus A.4. Die zu zeigende Hypothese ist hier aber, daß $6x = 0$ (genauer: $x+x+x+x+x+x = 0$) für alle x gilt.

```
MODE        PROOF

NAME        ring4

ORDERING    XKBO
            f:5 > j:4 > g:3 > 0:1 > a:1

EQUATIONS   j (0,x)       = x                   # 0 ist a left identity for sum
            j (x,0)       = x                   # 0 ist a right identity for sum
            j (g (x),x)   = 0                   # there exists a left inverse for sum
            j (x,g (x))   = 0                   # there exists a right inverse for sum
            j (j (x,y),z) = j (x,j (y,z))       # associativity of addition
            j (x,y)       = j(y,x)              # commutativity of addition
            f (f (x,y),z) = f (x,f (y,z))       # associativity of multiplication
            f (x,j (y,z)) = j (f (x,y),f (x,z)) # distributivity axioms
            f (j (x,y),z) = j (f (x,z),f (y,z)) #
            f (f(x,x),x) = x                    # special hypothesis

CONCLUSION  j(j(j(a,a),a),j(j(a,a),a)) = 0      # theorem
```

A.5.4 Das Beispiel Fibgroup

Bei diesem Beispiel handelt es sich um die Vervollständigung einer Zyklischen Gruppe. Das vervollständigte System berechnet die Kongruenzklassen der Struktur.

```
MODE        COMPLETION

NAME        FibGroup5

ORDERING    KBO
            a      : 6
            b      : 10
            c      : 6
            d      : 6
            e      : 1


EQUATIONS   a (e (x)) = d (x)
            b (a (x)) = e (x)
            c (b (x)) = a (x)
            d (c (x)) = b (x)
            e (d (x)) = c (x)
```

A.5.5 Das Beispiel Z22

Bei diesem größeren Beispiel einer Vervollständigung werden die Kongruenzklasen der ganzen Zahlen moulo 22 errechnet.

```
MODE        COMPLETION

NAME        Z22

ORDERING    LPO
            e1 > e > d1 > d > c1 > c > b1 > b > a1 > a

ORDERING    LPO
            a1 > a > b1 > b > c1 > c > d1 > d > e1 > e

EQUATIONS   a (b (c (x))) = d (x)
            b (c (d (x))) = e (x)
            c (d (e (x))) = a (x)
            d (e (a (x))) = b (x)
            e (a (b (x))) = c (x)
            a (a1 (x)) = x
            a1 (a (x)) = x
            b (b1 (x)) = x
            b1 (b (x)) = x
            c (c1 (x)) = x
            c1 (c (x)) = x
            d (d1 (x)) = x
            d1 (d (x)) = x
            e (e1 (x)) = x
            e1 (e (x)) = x
```

B Die Online-Hilfen der implementierten Programme

B.1 Das Programm trans

```
trans 1.1  vom 16.2.1992

Usage: trans [-v] [-h] [-c] [- o outfile] [infile1 ... infileN]

Geschrieben von Stephan Schulz

trans uebersetzt die von DISCOUNT erzeugten Protokolle
in echtes PCL-Format. Wird kein Eingabefile angegeben,
so wird die Standard-Eingabe gelesen, mehrere Eingabe-
files werden wie eine zusammenhaengende Datei behandelt.

 OPTIONEN:

 -v          Ausgabe der Programminformationen vor
             Beginn der Uebersetzung.
 -h          Ausgabe der Programminformationen, Abbruch.
 -c          Uebernahme von Kommentaren im Eingabefile,
             Ausgabe erlaeuternder Kommentare
 -o outfile  Schreibe Ausgabe in outfile. Entfaellt diese
             Option, so wird auf die Standard-Ausgabe ge-
             schrieben.
```

B.2 Das Programm `extract`

```
extract 1.1  vom 16.1.1992

Usage: extract [-v] [-h] [-c] [- o outfile] [infile1 ... infileN]

Geschrieben von Stephan Schulz

extract generiert aus eine PCL-Eingabe eine PCL-Ausgabe,
die nur noch die fuer das Ergebnis notwendigen Schritte
enthaelt. Wird kein Eingabefile angegeben, so wird die
Standard-Eingabe gelesen, mehrere Eingabefiles werden wie
eine zusammenhaengende Datei behandelt.

 OPTIONEN:

 -v          Ausgabe von Programm- und Ablaufinformationen
 -h          Ausgabe der Programminformationen, Abbruch.
 -c          Uebernahme von Kommentaren im Eingabefile.
 -o outfile  Schreibe Ausgabe in outfile. Entfaellt diese
             Option, so wird auf die Standard-Ausgabe ge-
             schrieben.
```

B.3 Das Programm `mextract`

```
mextract 1.1  vom 16.2.1992

Usage: rextract [-v] [-h] [-c] [- o outfile] [infile1 ... infileN]

Geschrieben von Stephan Schulz

mextract generiert aus einer PCL-Eingabe eine PCL-Ausgabe,
die nur noch die fuer das Ergebnis notwendigen Schritte
enthaelt. Wird kein Eingabefile angegeben, so wird die
Standard-Eingabe gelesen, mehrere Eingabefiles werden wie
eine zusammenhaengende Datei behandelt.

 OPTIONEN:

 -v          Ausgabe von Programm- und Ablaufinformationen
 -h          Ausgabe der Programminformationen, Abbruch.
 -c          Uebernahme von Kommentaren im Eingabefile.
 -o outfile  Schreibe Ausgabe in outfile. Entfaellt diese
             Option, so wird auf die Standard-Ausgabe ge-
             schrieben.
```

B.4 Das Programm `revert`

```
revert 1.0 vom 19.9.1991

Usage: revert [-v] [-h] [- o outfile] [infile1 ... infileN]

revert liest eine Textdatei und gibt die Zeilen in umgekehrter
Reihenfolge wieder aus. Wird kein Eingabefile angegeben,
so wird die Standard-Eingabe gelesen, mehrere Eingabefiles
werden wie eine zusammenhaengende Datei behandelt.
Falls der Speicherplatz zur Aufnahme der Datei nicht
ausreicht, wird die Ausgabe in Dateien mit den Namen
outfile.rev* beziehungsweise stdout.rev* geschrieben.

Geschrieben von Stephan Schulz

 -v          Ausgabe der Programminformationen vor
             Beginn der Uebersetzung.
 -h          Ausgabe der Programminformationen, Abbruch.
 -o outfile  Schreibe Ausgabe in outfile. Entfaellt diese
             Option, so wird auf die Standard-Ausgabe ge-
             schrieben.

 BUGS: Fuer die generierten Filenamen wird ein interner
       Puffer von nur 50 Zeichen Laenge verwendet.
```

B.5 Das Programm `rextract`

```
rextract 1.1  vom 16.2.1992

Usage: rextract [-v] [-h] [-c] [-r] [- o outfile] [infile1 ... infileN]

Geschrieben von Stephan Schulz

rextract generiert aus einer umgedrehten PCL-Eingabe eine PCL-
Ausgabe, die nur noch die fuer das Ergebnis notwendigen Schritte
enthaelt. Wird kein Eingabefile angegeben, so wird die
Standard-Eingabe gelesen, mehrere Eingabefiles werden wie
eine zusammenhaengende Datei behandelt.

 OPTIONEN:

 -v           Ausgabe von Programm- und Ablaufinformationen
 -h           Ausgabe der Programminformationen, Abbruch.
 -c           Uebernahme von Kommentaren im Eingabefile.
 -r           Ausgabe in umgedrehtem PCL-Format. Nur not-
              wendig bei Eingaben, bei denen der virtuelle
              Speicher nicht fuer den extrahierten Beweis
              ausreicht.
 -o outfile   Schreibe Ausgabe in outfile. Entfaellt diese
              Option, so wird auf die Standard-Ausgabe ge-
              schrieben.

BUGS: PCL-Instruktionen, die mehr als eine Zeile benoetigen,
      werden bereits beim Umdrehen zerlegt, sie koennen nicht
      verarbeitet werden. Aus dem selben Grund werden Kommen-
      tare, die in einer anderen Zeile als die PCL-Instruktion
      stehen, den falschen Schritten zugeordnet.
```

B.6 Das Programm lemma

```
lemma 2.01 vom 13.10.1992

Usage: lemma [-v] [-h] [-c] [- o outfile] ... [infile1 ... infileN]

Geschrieben von Stephan Schulz

lemma generiert aus einer PCL-Eingabe eine PCL-Ausgabe,
die nur noch die fuer das Ergebnis notwendigen Schritte
enthaelt. Dabei wird der Beweis durch Lemmata gegliedert.
Wird kein Eingabefile angegeben, so wird die Standard-
Eingabe gelesen, mehrere Eingabefiles werden wie eine
zusammenhaengende Datei behandelt.

 OPTIONEN:
 In der neuesten Version erlaubt lemma die Nutzung
 verschiedener Kriterien zur Lemma-Erzeugung. Dieses bedingt eine
 Vielzahl von Optionen, die hier in Gruppen zusammengefasst werden.

 (1) Optionen, die das Ein- und Ausgabeverhalten beeinflussen.

 -v                 Ausgabe von Programm- und Ablaufinformationen
 -h                 Ausgabe der Programminformationen, Abbruch.
 -c                 Uebernahme von Kommentaren im Eingabefile,
                    Generierung von neuen Kommentaren, die die ver-
                    wendeten Kriterien erlaeutern.
 -o outfile         Schreibe Ausgabe in outfile. Entfaellt diese
                    Option, so wird auf die Standard-Ausgabe ge-
                    schrieben.

(2) Optionen zur allgemeinen Steuerung

-criteria w         Gibt die fuer die Suche zu verwendenden Kriterien
                    an. w ist ein Wort, das nur die Buchstaben
                    'soitcp' enthalten darf. Sie stehen fuer:
                    's': Syntaktische Kriterien
                    'o': Oft benutzte Schritte
                    'i': Wichtige Schritte
                    't': Isolierte Teilbaeume
                    'c': Verwendete Vervollstaendigungsschritte
                    'p': Beweisunterteilung
                    Default ist "iotp".
```

```
-iterate             Suche Lemmata, indem die zu verwendenden
                     Kriterien nacheinander auf den kompletten
                     Beweis angewendet werden. Ansonsten werden
                     die Kriterien auf einzelne Schritte angewendet.

(3) Optionen zu abweisenden Lemma-Kriterien

-u_min_length n      Mindestlaenge fuer die Beweiskette eines
                     potentiellen Lemmas. Default ist 2, bei n=0
                     kommt das Kriterium nicht zur Wirkung.
-u_min_used n        Ein Schritt wird nur zu einem Lemma, wenn er
                     mindestens n mal im weiteren Beweis verwendet
                     wird. Default ist 1, bei n=0 kommt das
                     Kriterium nicht zur Wirkung.

(4) Optionen zu syntaktischen Kriterien

 -s_average_size n Ein Schritt wird Lemma, falls die
                     durchschnittliche Termgroesse kleiner oder gleich
                     n ist. Default ist 1, bei n=0 kommt das
                     Kriterium nicht zur Wirkung.
-s_max_size n        Ein Schritt wird Lemma, falls die
                     maximale Termgroesse kleiner oder gleich
                     n ist. Default ist 2, bei n=0 kommt das
                     Kriterium nicht zur Wirkung.
-s_min_fak r         Ein Schritt wird Lemma, falls das Verhaeltnis der
                     Termgroessen groesser oder gleich r ist. Default
                     ist 5.000000, bei r=0.0 wird das Kriterium nicht
                     verwendet.
-s_max_fak r         Ein Schritt wird Lemma, falls das Verhaeltnis der
                     Termgroessen kleiner oder gleich r ist. Default
                     ist 0.000000, bei r=0.0 wird das Kriterium nicht
                     verwendet.

(5) Optionen zur Suche nach haeufig verwendeten Lemmata

-o_min_used n        Ein Schritt wird zum Lemma, wenn er
                     oefter als n mal im weiteren Beweis
                     verwendet wird. Default ist 4.

(6) Optionen zur Suche nach wichtigen Lemmata

-i_lemma_weight n  Ein Schritt wird Lemma, falls das Produkt aus der
                     Laenge der Beweiskette und der Zahl der
                     Verwendungen groesser oder gleich n ist. Default
```

```
                   ist 11.

(7) Optionen zur Suche nach isolierten Teilbeweisen

-t_weight_factor r Setzt t_weight_factor, Default ist 0.500000.
-t_offset n        Setzt t_offset. Default ist 2. Die Verwendung
                   der Daten wird in der ausfuehrlichen Dokumentation
                   erlaeutert.

(8) Optionen zur Lemmasuche nach verwendeten Inferenzschritten

-c_init_weight n   Setzt den Wert, mit dem der PCL-Ausdruck
                   "initial" fuer die Lemma-Suche gewichtet wird,
                   auf den Wert n. Default ist 1.
-c_hypo_weight n   Ditto fuer "hypothesis". Default ist 0.
-c_orient_weight n Setzt den Wert, mit dem ein PCL-Ausdruck mit
                   Operation "orient" zum Gewicht des Ausdruckes
                   fuer die Lemma-Suche beitraegt auf den Wert n.
                   Default ist 0.
-c_cp_weight n     Ditto fuer "cp". Default ist 3.
-c_redu_weight n   Ditto fuer "tes-red". Default ist 2.
-c_inst_weight n   Ditto fuer "instance". Default ist 0.
-c_quot_weight n   Ditto fuer Ausdruecke, die nur Verweise auf
                   andere Schritte sind. Default ist 0.
-c_lemma_weight n  Setzt Schwellwert fuer Lemma-Generierung. Ein
                   PCL-Schritt, dessen Gewicht ueber diesem Wert
                   liegt, wird ein Lemma. Default ist 15.

(9) Optionen, die die Unterteilung langer Beweisketten steuern

-p_max_length n    Ein Schritt, dessen Beweislaenge groesser als n
                   ist, wird automatisch zum Lemma. Default ist 10.


BUGS: - lemma testet nur, ob Argumente vorhanden sind, der Typ wird
        nicht ueberprueft.
      - Jede Option muss fuer sich stehen.
      - Die Laenge der Beweiskette beruecksichtigt nicht die von proof
        vorgenommenen Verkuerzungen, wenn Beweisschritte durch
        Instanziierung ueberfluessig werden.
```

B.7 Das Programm `proof`

```
proof 1.1  vom 7.7.1992

Usage: proof [-v] [-h] [-nobrackets] [-nolemmas]
             [-o outfile] [infile1 ... infileN]

Geschrieben von Stephan Schulz

proof generiert aus eine PCL-Eingabe einen Gleichheitsbeweis,
der in einem an die menschliche Gewohnheit angepassten Format
ausgegeben wird.
Wird kein Eingabefile angegeben, so wird die Standard-
Eingabe gelesen, mehrere Eingabefiles werden wie eine
zusammenhaengende Datei behandelt.

 OPTIONEN:

 -v           Ausgabe von Programm- und Ablaufinformationen.
 -h           Ausgabe der Programminformationen, Abbruch.
 -nobrackets  Unterdrueckt die Ausgabe von Klammern hinter
              Konstanten.
 -nolemmas    Unterdrueckt die Suche nach Lemmata, nur die
              in der Eingabe spezifizierten Lemmata werden
              als solche behandelt.
 -noplace     Keine Ausgabe der Stellen, an denen Gleichungen
              angewandt wurden.
 -nosubst     Keine Ausgabe der verwendeten Substitututionen.
 -latex       Ausgabe in LaTeX. Es wird kein vollstaendiges
              Dokument, sondern nur ein zum Einbinden in
              bestehende Dokumente geeigneter Textbaustein
              geschrieben.
 -o outfile   Schreibe Ausgabe in outfile. Entfaellt diese
              Option, so wird auf die Standard-Ausgabe ge-
              schrieben.


 BUGS: Da PCL keine Reduktionsordnungen mehr kennt, koennen
       allgemeine Paramodulationsschritte als Reduktionen be-
       schrieben werden. proof kann verwirrt werden, wenn da-
       im ersetzten Term Variable auftauchen, die nicht beim
       Match gebunden wurden.
```

Literatur

[Av91] **Avenhaus, J.:** *Reduktionssysteme I, Skript zur Vorlesung im WS91,* Fachbereich Informatik an der Universität Kaiserslautern

[Av90] **Avenhaus, J.:** *Reduktionssysteme II, Skript zur Vorlesung im SS89,* Fachbereich Informatik an der Universität Kaiserslautern

[BDP89] **Bachmair, L.; Derschowitz, N.; Plaisted, D.A.:** *Completion without Failure,* Coll. on the Resolution of Equations in Algebraic Structures, Austin 1987, Academic Press 1989

[DP92] **Denzinger, J.; Pitz, W.:** *Das DISCOUNT-System Benutzerhandbuch,* Seki-Report

[KB70] **Knuth, D.E.; Bendix, P.B.:** *Simple Word Problems in Universal Algebras,* Computational Problems in Abstract Algebra, ed.: J.Leech, Pergamon Press 1970, pp 263-297

[La85] **Lamport, L.:** *LaTeX – A Document Preparation System,* Addison–Wesley Co., Inc., Reading, MA, 1985

[Li90] **Lingenfelder, C.:** *Structuring Computer generated Proofs,* Dissertation am Fachbereich Informatik der Universität Kaiserslautern, 1990

[LP90] **Lingefelder,C.; Präcklein, A.:** *Presentation of Proofs in an Equational Calculus,* Seki-Report 90/15

[LO82] **Lusk, E.L.; Overbeck, R.A.:** *A Short Problem Set for Testing Systems that Include Equality Reasoning,* Argonne National Laboratory, Illinois, 1982

[Pi92] **Pitz, W.:** *Realisierung eines Systems zum verteilten, wissensbasierten Gleichheitsbeweisen mit Hilfe der Teamwork Methode,* Diplomarbeit am Fachbereich Informatik der Universität Kaiserslautern, 1992